

Susanne Skjold Edvardsen
Malin Foss
Philip Morud

Knowledge in App

Graduate thesis in Bachelor i Programmering (BPROG)
Supervisor: Frode Haug
May 2023

Susanne Skjold Edvardsen
Malin Foss
Philip Morud

Knowledge in App

Graduate thesis in Bachelor i Programming (BPROG)
Supervisor: Frode Haug
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Summary of the Bachelor Thesis

Title: Knowledge in App

Date: 21 May 2023

Participants: Malin Foss, Philip Morud, Susanne Skjold Edvardsen

Mentors: Frode Haug and Tom Røise

Employer: Vitensenteret Innlandet

Keywords: Programming, mobile application, flutter, api

Number of pages: 86

Number of appendixes: 17

Availability: Open

Summary: Vitensenteret Innlandet, a high-profile science center, reached out to NTNU with a wish to make some of their services available online. To do this, the group settled on developing a mobile application together with an interface that can be accessed on any computer. This mobile application houses a ticketing system, blog, and a collection of some simple games. It focuses on a good user experience and in addition to this, the web interface gives administrative options like publishing and managing the content of the app, as well as managing tickets.

The solution is an application written in Flutter, in Android Studio, with a connection to a database in Firebase. This database uses both storage options and a real-time database. The application also uses a Stripe API for the payment solutions. The application has not been made available online, however, little work is needed to publish it as it is.

During the development of this app, the group put a high priority on working professionally. We used Scrum and Kanban as well as git issue boards to keep track of any tasks and wrote summaries of all meetings the group attended.

Sammendrag av Bacheloroppgaven

Tittel: Viten i App

Dato: 21 Mai 2023

Deltakere: Malin Foss, Philip Morud, Susanne Skjold Edvardsen

Mentorer: Frode Haug og Tom Røise

Oppdragsgiver: Vitensenteret Innlandet

Nøkkelord: Programmering, Mobilapplikasjon, Flutter, API

Antall sider: 86

Antall vedlegg: 17

Tilgjengelighet: Åpen

Sammendrag: Vitensenteret Innlandet, et høyprofilert vitenskapssenter, tok kontakt med NTNU med et ønske om å gjøre noen av deres tilbud tilgjengelig på nettet. For å gjennomføre dette ble gruppen enig om å utvikle en mobilapplikasjon med et nettside grensesnitt som kan bli brukt på enhver datamaskin. Denne mobilapplikasjonen inneholder et billettsystem, en blogg og en samling av noen enkle spill. Appen har fokus på en god brukeropplevelse og i tillegg gir internett grensesnittet administrative muligheter, som publisering og håndtering av innholdet i appen og behandling av billetter.

Løsningen er en applikasjon programmert i Flutter, i Android Studio, koblet til en database i Firebase. Denne lagrer data i sanntid og håndterer filer spesifikt. I tillegg bruker applikasjonen en Stripe API som betalingsløsning. Appen har ikke blitt gjort tilgjengelig på nettet ennå, men lite arbeid må til for å publisere den slik den er.

I løpet av utviklingsprosessen for denne appen ble det satt stort fokus på et profesjonelt arbeidsmiljø. Vi brukte Scrum, Kanban og Git "issue boards" for å ha oversikt over alle arbeidsoppgaver, og vi skrev sammendrag fra alle møtene gruppen holdt.

Preface

This Bachelor thesis was developed at NTNU by a group taking their bachelor in programming. The members of this group was Malin Foss, Philip Morud and Susanne Skjold Edvardsen.

We want to thank all those who made this thesis possible. Our Mentor Frode Haug always had our backs and pushed us towards a well-made project. Tom Røyse, who we could count on when difficult topics arose around the follow-through of the employer. We had an educated translator, Daniel Johan Bauge, go over any parts translated from English to Norwegian or vice versa, and we want to thank him. In addition to this, we want to mention those who helped perform user tests and always provided feedback during the development. And lastly, our employers at Vitensenteret, especially Gavin Robb, which came up with the task and helped provide a real-life experience for the group.

Thank you.

Table of Contents

List of Figures	vii
1 Introduction	1
1.1 Project Description	1
1.1.1 Background	1
1.1.2 Subject Area	1
1.1.3 Delimitation	2
1.1.4 Task Description	2
1.2 Goals and Constraints	3
1.2.1 Project goals	3
1.2.2 Constraints	4
1.3 Project Audience	4
1.3.1 Users of the Mobile Application	4
1.3.2 Users of the Web Interface of the Application	4
1.3.3 Readers of this Thesis	5
1.4 Project Organization	5
1.4.1 Group Members and Academic Background	5
1.4.2 Roles	5
1.5 Structure of the Report	6
1.6 Terminology	6
2 Theory	8
2.1 Subjects	8
2.1.1 Design	8
2.1.2 Database	8
2.1.3 API	8

2.2	Purpose	8
2.3	Subjects	9
2.3.1	Main Subjects	9
2.3.2	Relevant Subjects	9
3	Requirement Specifications	11
3.1	Requirements	11
3.1.1	Front End	11
3.1.2	Back End	11
3.1.3	Operational Requirements	11
3.1.4	Safety and Misuse Handling	11
3.1.5	Storing Personal Data in Accordance with GDPR	12
3.1.6	Licensing	14
3.1.7	Version updates	14
3.1.8	Interface Requirements	14
3.1.9	Testing	15
3.2	Initial Design	15
3.2.1	Low Fidelity Prototype for the Mobile Application	15
3.2.2	Low Fidelity Prototype for the Web Interface	16
3.3	Use Cases	17
3.3.1	Use Case Diagram	17
3.3.2	Use Case Descriptions	19
3.4	Product Backlog	22
3.5	Domain Model	25
4	Developmental Progress	27
4.1	Methodolgy	27
4.1.1	Choice of Software Development Methodology	27

4.1.2	Sprint Meetings before the Hybrid Solution	28
4.1.3	Sprint/Kanban Meetings after the Hybrid Solution	29
4.2	Meetings	29
4.2.1	Mentor Meetings	29
4.2.2	Project Owner Meetings	29
4.2.3	Internal Meetings	29
5	User Interface	30
5.1	Blog Page	30
5.2	Game Page	31
5.2.1	Viten Ord	31
5.2.2	Tower of Hanoi	32
5.2.3	Viten Kode	33
5.3	Authentication Page	34
5.4	Ticket Page	35
5.4.1	Customer Ticket Page	35
5.4.2	Employee Ticket Page	36
5.5	Settings Page	37
6	Technical Design	39
6.1	Technology	39
6.2	API	40
6.3	System Architecture	40
6.4	Data Storage	41
6.4.1	Access and Security	43
6.4.2	Limitations	44
7	Implementation	46

7.1	Database	46
7.2	API	51
7.2.1	Firebase Functions	51
7.2.2	Stripe API	52
7.3	Mobile Application	52
7.3.1	Log in and Register Page	52
7.3.2	Ticket Page	54
7.3.3	Settings	54
7.3.4	Blog	55
7.3.5	Payment Page	55
7.4	Web	56
7.4.1	Ticket Page for Admins	57
7.4.2	Blog Page for Admins	60
7.5	Game Implementations	61
7.5.1	Viten Ord	61
7.5.2	Tower of Hanoi	62
7.5.3	Viten Kode	66
8	Testing	68
8.1	Unit Testing	68
8.2	User Testing	68
9	Code Quality	71
9.1	Database Best Practices	71
9.2	Commenting Standards	71
9.3	Naming Conventions	72
9.4	Using Flutter's Built-in Linter to Minimize Redundant Code	72

10 Deployment	73
10.1 App Store for iPhone	73
10.2 Google Play Store for Android	73
10.3 Source Code	73
10.4 Terms and Conditions	73
11 Further Development	75
11.1 Deployment	75
11.2 Further Polish of Interface	75
11.3 Database	75
11.4 Games	76
11.4.1 Game Engines	76
11.4.2 Further development of "Viten Kode"	76
11.4.3 Adding a Scoreboard and Daily Rewards for Logged-in Users	77
11.5 Notification System	77
11.6 Other Additions	77
12 Discussion	78
12.1 Implementation and Follow Through	78
12.1.1 Worked Hours	78
12.1.2 Following the Gantt Chart	81
12.2 Alternative Technology and The Choices We Made	83
12.2.1 Firebase Realtime Database vs. Firebase Firestore vs. SQL	83
12.2.2 Kotlin vs. Flutter vs .Net vs React Native	84
12.2.3 Payment system	85
12.3 What Would We Have Done Different Today?	86
12.3.1 Database choice	86
12.3.2 Payment system	86

12.4 Evaluation of the Group Effort	87
12.5 Conclusion	87
13 Sources	88
Appendix	91
A Gantt	91
B Project plan	93
C Contract	112
D Status report 1	119
E Status report 2	122
F Status report 3	125
G Meeting Minutes	128
H Time Chart	142
I Wishes from Employer	143
J User tests	143
K Group rules	151
L Requirements specification	154
M Translated summary	171
N Design Handbook for Vitensenteret Innlandet	173
O Assets	214

P Backlog	216
Q User Manual	218

List of Figures

1	Paper model, low fidelity prototype.	16
2	Digital model of the Web interface, low fidelity prototype.	17
3	Model of the use cases	18
4	In Depth Use Cases	22
5	Model of the domain model.	25
6	Screenshot of issue 40.	28
7	Screenshot of a specific worked day in the time sheets.	28
8	Final design of web version of blog pages	30
9	Final design of web version of game pages	31
10	Screenshot of the game "Viten Ord".	32
11	Screenshot of the game "Hanoi's Tower".	33
12	Screenshot of the game "Viten Kode".	34
13	Final design of web version of login and registration pages	35
14	Final design of phone ticket pages	36
15	Final design of web version of admin ticket pages	37
16	Final design of web version of settings page	38
17	All technologies used in this project.	39
18	Model of the Database.	40
19	Model of the Database.	42
20	Rules for Firebase Realtime Database.	43
21	Rules for Firebase Storage	44
22	Screenshot of registering a user in Firebase Authenticate.	46

23	Screenshot of saving user data to Firebase Realtime Database.	46
24	Screenshot of ticket data class.	47
25	Screenshot of function for getting all tickets from a user.	48
26	Screenshot of function updating user email.	49
27	Screenshot of function deleting a user.	50
28	Screenshot of Firebase function	51
29	Screenshot of Stripe API call	52
30	Screenshot of email register field.	53
31	Screenshot of function for validating email formats.	53
32	Screenshot of page for editing email address.	54
33	Screenshot of the blog.	55
34	Screenshot of the payment page with and without the payment sheet . . .	56
35	Screenshot of the ticket page for Admin.	57
36	Screenshot of the search bar code.	58
37	Screenshot of the code that sorts users and tickets.	58
38	Screenshot of the code that includes or excludes expired tickets.	59
39	Screenshot of how the blog page looks for an admin.	60
40	Screenshot of the code handling correct letter guesses.	61
41	Screenshot of the code handling yellow cases and wrong letters.	62
42	Illustration of how the game board is set up in the Tower of Hanoi.	63
43	Code of DragTarget for Tower of Hanoi.	63
44	Code of Draggable with index 0 for Tower of Hanoi.	64
45	Code of function "checkLegalMove" Tower of Hanoi.	65
46	Screenshot of the main widget for Viten Kode	66
47	The game Viten Kode	67
48	Unit Tests, performed on all games.	68
49	User Test Results.	69

50	Example of a function comment	71
51	Time chart legend	78
52	Time Chart Malin Foss	79
53	Time Chart Philip Morud	80
54	Time Chart Susanne Skjold Edvardsen	81
55	Gantt Chart	82
56	Time Chart	142

1 Introduction

1.1 Project Description

1.1.1 Background

Vitensentrene is an association of many regional science centers around Norway [1]. These locations are open to visitors of all ages, and contain many exhibitions in which the visitors can play around with science experiments and educate themselves on fun and interesting topics. Vitensenteret Innlandet, VI, is one of these locations.

Entry to Vitensenteret Innlandet requires a ticket either purchased online and validated upon entry or bought at the entrance. If the visitor wishes to purchase a ticket for a longer duration they can also do so.

The current system is quite user unfriendly as the online purchase was actually just a gift card that had to be redeemed at the entrance by the cashier, meaning you barely saved time compared to just buying it at the entrance. In addition the cashier would have to manually type the gift card code, manually subtract the cost of an annual ticket from the balance of the gift card and then add the customers personal information as a ticket to the system. This left a lot of room for human error from the cashier, it was also inefficient and had a tendency to crash as inputs weren't checked before being pushed to the database.

Vitensenteret Innlandet were in need for a new and better system, and in these times where almost everyone is carrying their phone wherever they go, having an app was a natural choice to make the process easier for both visitors and employees. In this context, Vitensenteret contacted NTNU with a bachelor thesis topic, involving ticket purchase through a mobile application, as a part of a solution to this problem.

Vitensenteret asked that we develop a mobile solution so that users have a way to purchase and track tickets. Additionally, they also wished for a way to alert users of the app of events, sales, or other important things through notifications. Lastly, they wanted an interactive part of the app, a way to retain use of it even after a visit has been done.

1.1.2 Subject Area

A physical membership card is very common for members to gain entry to locations where a ticket or membership is needed for entry. The use of such cards is becoming less common in favor of phones. The reason for this may be the decrease in the use of a wallet, or simply the costs of manufacturing the cards, card scanners, and equipment to support this system [2].

Phones are already on a person at all times. This device holds many important apps like

social platforms, BankID, and games. Thus making an application for phones which will contain all that is needed during a visit to Vitensenteret, like tickets and an overview of events at the location, is a good way to solve this issue.

The subject of this project is the development of an application for mobile devices. This application is able to run on both Android and iOS devices. The team has also developed an interface for the web so that the employees at Vitensenteret have an easier time managing the app. During the process of developing this app, we touched on many subject areas, like intuitive design, databases and cloud technologies, game design and overall working on software for a mobile device.

In addition to the above, this project has taught the group members how to work together on a project of this scope. While the final product has an important role in the final evaluation, one of the first limitations we employed as a group was to focus on the developmental process and documentation around the project. What this meant for the project is that the wishes of the employer come second in regard to what we prioritized when developing this app.

1.1.3 Delimitation

Instead of using the outdated systems at Vitensenteret, our group has developed a mobile application that allows users to purchase tickets to gain entry to the location. In addition to this, the users get updates about Vitensenteret in the app and are able to play brain teaser games. The employees at Vitensenteret also have received an interface specific for their use in managing news, users, games and tickets. This has been developed by the group, with a focus on proper documentation and a solid workflow.

1.1.4 Task Description

- The group has made an application for mobile platforms with a web interface. This application runs on both Android and iOS, as well as browsers such as Edge, Chrome etc.
- This application communicates with a database in Firebase to administer the users who have a valid ticket to enter. The database also contains information on blog posts as well as containing task information for one of the games.
- The application allows users to pay for tickets through the app, through the API Stripe.
- The app allows administrators to push blogs with content from Vitensenteret into a feed that all users have available.

-
- The application was developed using design principles and has been adequately secured to make sure that there are no security breaches.
 - The application has a number of interactive games. These games are in line with other functions of Vitensenteret and consist of some form of brain teasers.

1.2 Goals and Constraints

1.2.1 Project goals

Impact Goals: The main goal of this project was to develop an application for users of Vitensenteret so that the users are not dependent on bringing anything other than their phones to the locations. The application will bring the presence of Vitensenteret to those who have the app, creating a familiarity with it. Vitensenteret has a goal to reach 50-60 000 visitors in 2023, and our goal is that 40% of those use the app before, during or after their visit.

Project Goals: The main aspect that we aimed to develop is a way to showcase tickets purchased, like a yearly one, as well as allow for purchasing these tickets through the app. In addition to this, we wanted to develop a notification system where the administrators of the app can post notifications so that normal users of the app are alerted to events or sales at Vitensenteret. Safety is an important factor in all of this. We also planned that if we reached our goals early, see our Gantt chart in the appendix [A] we would work on developing some games for the application.

It was also imperative that we made the app a base for further development as Vitensenteret has many plans for that.

Educational Goals: The goal of this task was to learn how to develop a bigger project than what has been previously worked on throughout this education. With a focus on the developmental process from day one, where proper documentation is vital, as well as having a fluid grasp and accomplishment of the agile development method that the group chose for this task. Up until this point, most, if not all, tasks that have been worked on have been internal, while now that the group is working with a team outside of NTNU, we knew we would experience and solve unique situations that the group members never had have encountered before.

Additionally, we aimed to take many of the subjects we have had through our education and combine them into this task. This encompasses everything from design, coding, task writing, and ethics. All of it was done together in a structured group where each member has shared and separate roles, for the educational plan on this subject from NTNU [3].

1.2.2 Constraints

- The app has to work on both Android and iOS. Target api 31 [4], and iOS 15 [5].
- The app has to be user-friendly, for kids and adults.
- We have to follow the Vitensenteret design handbook [N] when designing elements of the app.
- The app should work with the existing database in Firebase.
- The code has to be well documented and made for future development.
- The app will have a user manual [Q] so employees at Vitensenteret can learn the app.

1.3 Project Audience

1.3.1 Users of the Mobile Application

The mobile application is targeted toward those who travel to and visit Vitensenteret. The main purpose of the application is to provide an overview of tickets and be able to purchase them through the application. In addition to the tickets, we also focused on developing a blog so that users could get news about Vitensenteret. The other functions are add-ons to make the application feel better for the users and to make it so that the app is more likely to be used, and maybe even make the users of the application visit Vitensenteret more often.

Since any person can have a ticket in their name, the application is targeted at all audiences who visit Vitensenteret. The application has some games that younger children will be able to click around inside and use, as well as providing entertainment for adults.

1.3.2 Users of the Web Interface of the Application

The web interface is for those working at Vitensenteret. With different levels of admins, there are different rights. Those at the main desk will be running it similar to a cashier, where they can check tickets of attendees in the interface by searching for names and such. Should the user have inputted any wrong information, it will also allow those employees some editing of users. They also have some right to create tickets. Admins have the unique ability to access the blogs and add content there.

1.3.3 Readers of this Thesis

This thesis is documentation on how we worked to solve this project and will include the first project plans, and diagrams, as well as the final results. In addition to this, it will give an insight into developmental methods and allow others who may be developing similar projects to take inspiration from this.

This thesis is also for mentor, examiner and those at Vitensenteret who may wish to get an insight into the full development of the final product.

1.4 Project Organization

1.4.1 Group Members and Academic Background

The members of this group are Malin Foss, Philip Morud, and Susanne Skjold Edvardsen. All of the members of the group are attending NTNU completing the bachelor's degree in programming. We have had different elective courses so each member of the group has the competence to develop what we need, with a multitude of additional knowledge to draw from as well.

We have all coded in C++ and other object-oriented programming, so moving to Flutter, which we used for this project, was no problem. We all have had previous experiences with running bigger projects, however not in this scope. For this project, we were expected to put around 75% of our work week into this project, whereas in previous subjects it was around 25%.

1.4.2 Roles

The main roles we have for this project are project manager, research manager, and documentation manager. Malin Foss was the project manager, which includes having an overview of what should be done, keeping an eye on the backlog, and paying attention to deadlines. Malin also took on the responsibility of communications with our mentor and employer, as well as any other outside communication. Philip Morud was responsible for research, mostly in regard to the development of payment system. In addition to this, he was also the Scrum master and hosted all the scrum-related meetings. Susanne Skjold Edvardsen was responsible for all documentation. She made drafts of the documentation and delegated where and what should be written. In addition to this, she made meeting minutes of all gatherings the group attended.

1.5 Structure of the Report

The report is divided into the following document structure. The document is divided into 12 numbered chapters with the appendix at the end:

- 1. Introduction
- 2. Theory
- 3. Requirement Specifications
- 4. Developmental Progress
- 5. User Interface
- 6. Technical Design
- 7. Implementation
- 8. Testing
- 9. Code Quality
- 10. Deployment
- 11. Further Development
- 12. Discussion

Appendix

1.6 Terminology

API An API is a way for websites to open a specific page to gather information and use it within your own software.

GDPR This is a rule set active in all of the EU which makes the storage of a person's personal data keep to strict rules.

I/O I/O stands for input and, or output.

Kanban Is a team-based agile development method centered around a continuous developmental cycle for developing software.

Scrum Is a team-based agile development method centered around sprints and designated roles for developing software.

VI/Vitensenteret/Vitensenteret Innlandet/Project Owner These are synonymous and all refer to the employer who created this bachelor.

UI User Interface is the interface or the view that the user is presented with when using the application.

UX User Experience is the entire experience a user has when using the app. This encompasses everything from looks to functionality.

2 Theory

2.1 Subjects

Vitensenteret asked for a multi-platform solution, for the web, and for phones. This application would also be connected to a database and an API. This required competence within many different areas.

2.1.1 Design

First and foremost, creating a valid interface across a multi-platform application, with changing screen sizes, sometimes even dynamically, required a keen attention to detail. The application works on the smallest phones, to the web without loading specific versions of the application. We also made an emphasis on following the design specifications given to the group by Vitensenteret, and this sample can be found in the appendix [N].

2.1.2 Database

Databases are used to persistently store data remotely. For us, this data was for the most part user and ticket data, and especially ticket data was important to keep in a database. For this purpose, we use Firebase Realtime Database and Firebase Storage. The database also hosts a Google Cloud function responsible for initiating payments when a user buys tickets.

2.1.3 API

The mobile part of the application makes use of a Stripe API which handles purchases made in the app. The Stripe API allows for a test mode where developers can test their app's functionality without having to use actual funds.

2.2 Purpose

There are a couple of reasons why we chose this task for our bachelor's thesis. During the research phase and when the group first met, as we have not been in a group setting before, we all went over subjects we liked and wished to pursue in this bachelor thesis. One of these subjects was Mobile Programming. A subject where we learned mostly how to create Android applications through an IDE called Android Studio. This was something everyone in the group enjoyed, so we were mostly interested in tasks that could be solved

via a mobile application. In particular, one, where we could also develop for iOS as we had no experience with that and wanted to learn.

Furthermore, there was a wish to connect to some other resource, like an API or a database. We reasoned that the apps developed in the subject Mobile Programming, were standalone applications, and therefore wanted to create something with systems outside of "just a standalone app".

There was also a push to make something creative, so when all the different employers presented their tasks, we were especially interested in this one. When we talked with Gavin Rob, the contact person at Vitensenteret, we discussed some ideas back and forth and discovered that undertaking this project would align well with what we already knew, with an excellent opportunity to build on that knowledge and learn more.

2.3 Subjects

The structure of the study can be found here [6]. Under is a selection of subjects that were particularly relevant to this thesis.

2.3.1 Main Subjects

PROG2007 - Mobile Programming was the central subject that we wanted to further our knowledge around. This served as the basis for the bachelor thesis. With the knowledge here we had a basic application that we could further develop with the knowledge gained through this project.

IDG1362 - Introduction to user-centered design was very relevant to create an interface that felt natural for the user. This subject helped put a focus on a well-designed application, where you can easily navigate exactly where you want to go.

PROG2005 - Cloud Technologies were relevant for connecting the application to an API. The connection of the application to the API we use for the payment systems was done through our knowledge gained in this subject.

IDATG2204 - Data modeling and database systems were a subject about creating databases and using them. We had good use of this as we created our own database, and fetch information from it in the app.

2.3.2 Relevant Subjects

PROG2052 - The integration project was a subject where we created a software program on our own entirely from scratch. This was a way to do the entire process from start to

finish before the bachelor thesis and it helped us prepare for "the real thing".

IIKG1001 - Cybersecurity and computer networks as well as IIKG2001 - Software Security both helped in how we handled communication with the database and the API. It also affected how we stored user information.

IDATG2102 - Algorithmic methods and IMT3603 - Game Programming were two subjects centered around maths and fun. These two subjects were central in the brain teaser games developed inside the application.

3 Requirement Specifications

3.1 Requirements

3.1.1 Front End

To use this application we have developed an application that can be downloaded for phones, both Android and Apple devices. Additionally, we have an interface that is compatible with a desktop so that those who are working at Vitensenteret can use their computers to gain access to the application, as well as the more administrative parts of the page.

3.1.2 Back End

The application is connected to a database in Firebase and has general updates through this system. Flutterfire [7] is the interface between Flutter, the programming language that the group used to develop the app, and Firebase. We have some functionality in our database using Google Cloud. In addition to this, we have utilized the Stripe API for the payment solution.

3.1.3 Operational Requirements

The app was developed for both Android and iOS. For Android the target API is 31, the current standard target API for Android apps [4]. For iOS we aim for iOS 15, which is currently the version in use by 89% of all iOS devices on the App Store [5]. We will also develop the administrative part of the application for web, only used by employees at Vitensenteret Innlandet. The database in use is a Firebase database, which keeps user data, game data, blog data as well as ticket data.

3.1.4 Safety and Misuse Handling

Since the app will be handling sensitive user data it has to follow standards and laws around that, like GDPR [8]. We are using Firebase Authentication [9], which handles the safety surrounding the connection and the handling of passwords. This will ensure that the users and their tickets are safe. Tickets and gift codes are unique and only able to exist in one place at a time, otherwise, duplication and theft can happen. It is not possible to use someone else's ticket to gain access to Vitensenteret. This is ensured by security rules in both the database, and check done in the code. When buying tickets we also use existing systems to ensure that it is safe.

To ensure the integrity of the app, access to features is restricted by security levels. There are four security levels that are necessary; unregistered user, registered user, employee, and admin. This is also reflected in the use case diagram which can be found in "3.3 Use Cases".

Unregistered user can only view the blog and individual blog posts, as well as play games. For anything else they have to log in. Registered users will be able to buy tickets, change their own user data or delete their own account along with what an unregistered user can. Employees can add tickets to users, change basic user data and see all users and all tickets, as well as all the things registered users can. Admins can do everything employees do, as well as appoint new employees or admins, and publish, change or delete blog posts.

We store some data both locally and in the database. Data stored locally is mostly for speed, and accessibility within the app. Even if this data were to be changed by something or someone, it would quickly be overwritten by the data in the database, as we fetch this regularly. Data in transit is encrypted by Firebase, and all the services we use from Firebase also encrypt their data at rest [10].

3.1.5 Storing Personal Data in Accordance with GDPR

In 2018, the European Data Protection Regulation was applicable to all of the members of the European Union [8]. This General data Protection Regulation, GDPR for short, ensures that the data stored on individual people is not in any way used for malicious intents.

In the app that is being developed by us for Vitensenteret, it is a necessity to store certain details about its members to obtain some functions of the app. Most of the app can be accessed without the application storing anything about the user. What is stored about the user, with the user not being a member of the app, and signed in, are as follows:

If the app is idle, then the app will store in its cache what page the user was most recently using, before leaving the app in its idle mode. It will also store progress made in the games, for as long as they are not exited.

In relation to the definitions specified by the GDPR, personal data is the following: "‘personal data’ means any information relating to an identified or identifiable natural person (‘data subject’); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person; " [[8], Art. 4].

Abstracting from this, we can say that the data stored about unregistered users are not personal data, as the data cannot by any means recreate the persona they originated from.

Therefore this data can be stored without consideration of the GDPR.

However, in the instance where a user will need to log into the account to gain access to the other functionalities of the application, we will need to store the following about the user:

- Name of the user.
- Email of the user.
- Phone number of the user.
- Image of the user.
- Password of the user.

The reason why the app needs to store this data, is because the app will provide a paid service, and to ensure that only the user who paid for their ticket is using it, we need to store this information about the user. And as by the definition by the GDPR above, this clearly qualifies as personal data. This data, by the rules of the GDPR needs to be the following:

- The user will be made aware that by creating an account they consent to having this data stored about themselves.
- The user will be made aware that by deleting an account their personal data will be wiped from the databases, and is unrecoverable.
- The data will be stored in secure locations, and sensitive data, like the password, will be hashed.
- The data will not be processed in any other way than to ensure that the user is who they claim to be.

Other principles in regards to the GDPR can be read following the sources of the regulation, however, the points listed above are those that have a concrete and important factor in the development of this app. In addition, we want to mention that in order to conduct the payment methods, we are outsourcing the handling of the information the user inputs to buy tickets. This happens through the Stripe API, which then is the party responsible for handling said data in accordance with the GDPR.

3.1.6 Licensing

The product was developed by us, the student group at NTNU as a bachelor's thesis. The contract that we signed with Vitensenteret [C] states that we retain no rights to ownership of the final product, and can only use the content that we created as a way to show our skills to potential future employers. This means that Vitensenteret has the rights to the software that we develop during this course, for this thesis.

3.1.7 Version updates

At the point in which we deliver our product and software to Vitensenteret, we are no longer responsible for the upkeep of this software. Since the ownership is transferred to Vitensenteret, they are now responsible for updating the software and keeping it relevant for the users at Vitensenteret.

To make this task easier for Vitensenteret the code is written in modular steps which makes it easy for other developers to continue work on the software. Our design choices reflect this as well, as we created a manageable file system with an easy-to-understand structure.

We have also written the code with this in mind, and as such it has been commented and explained at crucial points in the code, what the different sections do. In addition to this, Vitensenteret will gain access to all our developmental documents, like the domain model, use case diagrams, and furthermore can expand on these as they see fit.

3.1.8 Interface Requirements

The design of the application will follow the design conventions as stated by Vitensenteret themselves [N]. In the early development of the application, the group received a pamphlet with these design conventions, which can be found in the appendix.

The app itself has been designed for the most common use cases of the app, buying tickets, seeing the blog, and playing games. Other functionalities lay easily accessible in the menu. All these functionalities are placed in a bar on the bottom of the screen for ease and reach. The design follows the common "Z" pattern [11], with an attention grab at the top, a scan of the middle, and ending up at the easily accessible lower bar. For the blog itself, we utilized the "F" pattern [12] instead, as that is better suited for a more text-centralized design.

3.1.9 Testing

After each issue was completed, it was comprehensively tested by the one who implemented the solution. Each developer worked on their own branch for any major tasks, and before it was pushed to the main source it was tested by them. In meetings, the group went over what has been implemented, to ensure a clean working tree at all times.

In addition to this, we created unit tests for many functions, more in this later in the chapter "Testing". We decided against using implementation tests as it would require a complicated setup and routine to work with Android Studio, and the scope of such was a little outside of the project. As a solution to this, we resolved to perform user tests at the end of the developmental period to weed out anything that we may not have seen ourselves, as well as find more for possible future development to work with.

3.2 Initial Design

3.2.1 Low Fidelity Prototype for the Mobile Application

The original proposition for the app can be seen in Figure: 1. Here the group sketched out a paper model for what we envisioned Vitensenteret would want. The model includes a news page as the main page, with a game tab leading to an example sudoku game, as well as a ticket tab that leads to the ticket page. Additionally, there is an image of the user, which when clicked leads to the settings page. The final application is not so different from this original proposition.

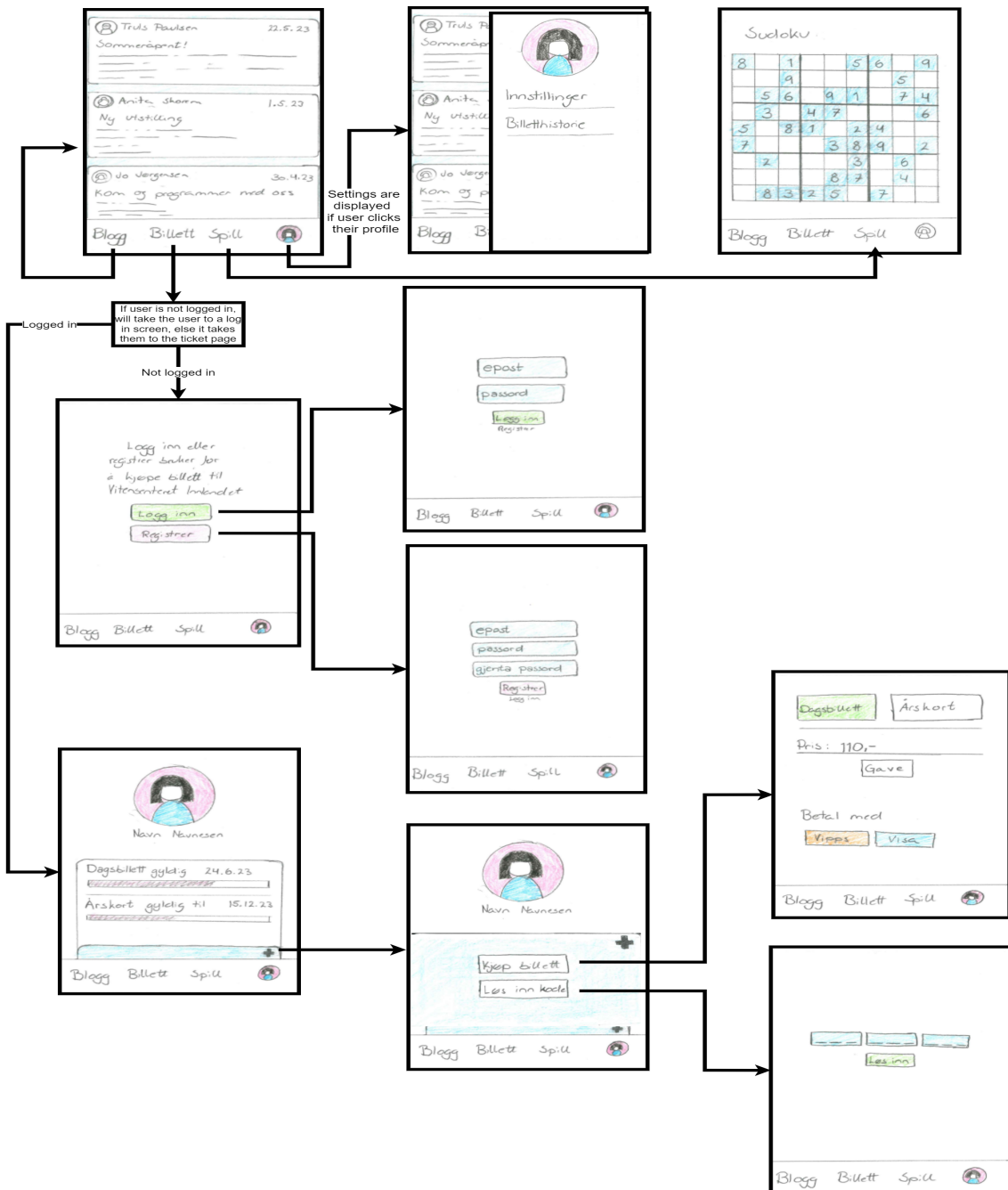


Figure 1: Paper model, low fidelity prototype.

3.2.2 Low Fidelity Prototype for the Web Interface

We also had to create a web interface for the application. We created these low-fidelity charts online, to showcase which functionality we may need. While most of the functionality was implemented, the final designs have deviated quite a bit from the first prototype.

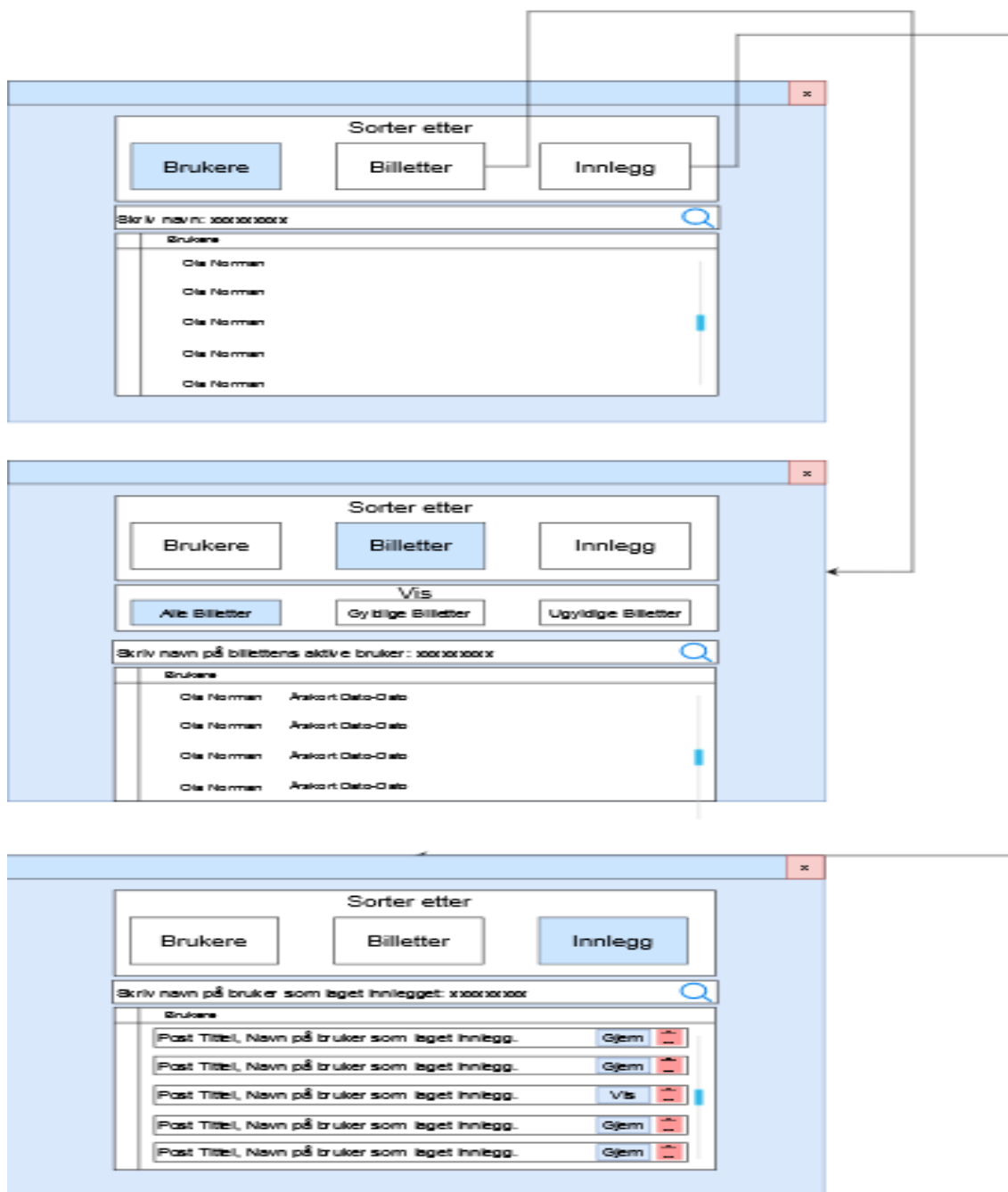


Figure 2: Digital model of the Web interface, low fidelity prototype.

3.3 Use Cases

3.3.1 Use Case Diagram

Figure: 3 shows the use cases for the application. There are a variety of users, all of which inherit rights, whereas the admin has all the rights of the user types underneath it.

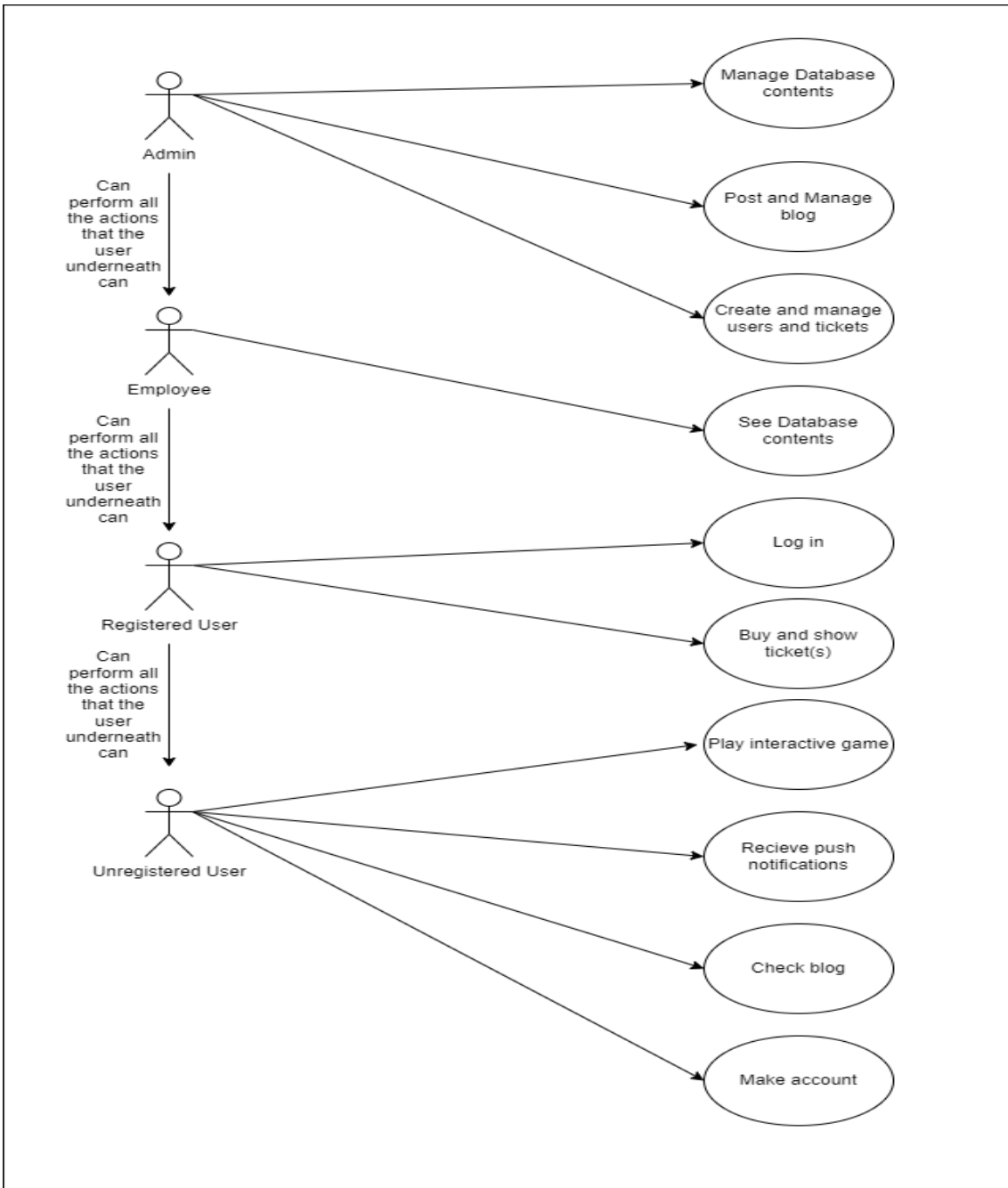


Figure 3: Model of the use cases

3.3.2 Use Case Descriptions

Action:	Post blog
Actor:	Admin
Goal:	Make a publication in the form of a blog
Description:	The admin navigates to their profile icon, and taps it. This will open an additional menu, where options, tickets etc are available. On this menu there is an option to create a blog post, which the admin can press. They will then be presented with a simple blog creation page, where they input the title, content, and potentially image with the blog. Additionally there is an option to create a notification at the publication of the blog, which the admin can cross off on a yes or no based on the situation.

Action:	Manage Database Contents
Actor:	Admin
Goal:	Remove Post from database
Description:	The admin will be on the desktop version of the app. They will navigate to the button with posts "Innlegg". Here they will be presented with the option to search for the post they wish to remove. After searching they are presented with a list of posts that match the search criteria. Selecting the post that is the target, the admin is prompted to hide it from the public, or outright delete it. The admin deletes the post.

Action:	Make account
Actor:	Unregistered User
Goal:	Create a new account with personal identification
Prerequisite:	The user does not already have an account.
Aftermath:	The user has made an account.
Description, normal flow:	The user clicks on the "register user" button and is brought to a page where they are prompted to input a profile picture, name, email and password. If the input is invalid the user will be prompted again. When the user has entered valid data, a new user is created in the database and the user is automatically logged in.
Description, alternative flow:	In the case where the user is logged into an account they dont want to have, they will navigate to their usertab on the right. This will open the profile of the user, where there is a button to log out. After this is completed, the user can follow the normal flow description.
Error situations:	Errors that may happen, or stop the progression is if the user has entered credentials that is registered to another user, or if they mistype something. This includes the likes of: email, name, password, or picture is the wrong format. Another error that may stop this is if the user has a poor connection to the database.

Action:	Log in
Actor:	Registered User
Goal:	Log in on the registered account
Prerequisite:	The user has an account registered to them already.
Aftermath:	The user has logged into the app with their credentials.
Description, normal flow:	The user clicks a "Login" button and is brought to a page where they are prompted for email and password. If either email or password are invalid the user will be informed of such, and will have to reenter valid information. After they have logged in they are brought to the blog and the icon showcasing the profile will be updated with their profile image.
Description, alternative flow:	If the user does not have an account, they will have to create one, following the use case example: Make account.
Error situations:	Errors that may happen, or stop the progression is if the user has entered credentials that is registered to another user, or if they mistype something. This includes the likes of: email, name, password, or picture is the wrong format. Another error that may stop this is if the user has a poor connection to the database.

Action:	Receive push notification
Actor:	All Users have access to this action
Goal:	Receive an update from Vitensenteret
Prerequisite:	Their phone is turned on, connected to some sort of internet and has the app downloaded.
Aftermath:	The user has a notification on their phone from the app.
Description, normal flow:	While the user is not actively using the application they receive a notification from the app, which may prompt them to open the app to gain access to more information through the blog.
Description, alternative flow:	In the case where a notification is not present on the phone, the only way to receive the update is to go into the app, and go to the blog page, this page is also the homepage. Here the notification will be displayed as a blog post.
Error situations:	The notification does not appear, or the content of the notification is wrong. Another error that may stop this is if the user has a poor connection to the database.

Action:	Buy Ticket
Actor:	Registered User
Goal:	Purchase a yearly ticket
Prerequisite:	The user is logged in, and has some way or means to purchase the ticket.
Aftermath:	The user now has a yearly ticket in their name, on their phone.
Description, normal flow:	The user navigates to the ticket button. They will then be taken to a page which showcases any purchased tickets, if there is any to showcase. There is then a plus icon they can press next to the tickets. Pressing this button slides up a new menu with the options "Buy Ticket" or "Use Code". In this case the user navigates to "Buy Ticket" and is sent to another page. They are prompted with the option of "day ticket" or "Yearly Ticket" and select the later option. The price will be displayed and underneath the price will be an option to make this ticket a gift. Without pressing the gift button the user navigates down to see that there are multiple payment options, they choose Vipps, and are then forwarded to that application to finish the payment. Once that has been processed the ticket page will show the updated yearly ticket that was purchased.
Description, alternative flow:	The user goes to buy a ticket by pressing the ticket button only to discover they already have one. So they dont need to buy another.
Error situations:	The user chooses the wrong kind of ticket, or the wrong kind of payment method and has to contact Vitensenteret to get it resolved. The user chooses the ticket to be a gift, even tho it isnt one.
	The payment handing was flawed and the payment was not processed. Error with the connection to the database, or lacking internet.

Figure 4: In Depth Use Cases

3.4 Product Backlog

We created our backlog on a reMarkable, which is a tablet that functions similarly to a notebook but enables the notes to be digital and be uploaded directly to the internet without scanning the notes. We wanted to do it like this to have a natural and intuitive way to figure out what to work with. This is also why it was created in a mix of Norwegian and English. This backlog can be found in its original entirety in appendix [P].

This backlog is divided into three main parts: administrative, app, and other. The tasks finished have been crossed out with a line passing through it, and those not finished are left. Tasks that were discarded have an x next to them. This is what the backlog looked like in pure text form:

Administrative:

- Kravspek
- Statusrapport x3 til Frode
- Endelig rapport
- Manual for admins
- GDPR
- Vipps
- Stripe

App

- Login
- Registration
- Create blog entries
- Delete blog entries
- Payment system/ duell vipps
- Gift code input
- Bottom bar
- Profile page
- Settings
- Game page + other game
- Fix issues with web from login and registration
- Display Tickets
- Bytte språk

-
- Riktige farger
 - Edit blog
 - Viten Ord
 - Hanois tårn
 - Viten Kode
 - Delete user

Other

- Connect to database
- Backend connection to database
- GitLab CL pipeline
- Gain access to duell API
- Access vipps
- Tests
- Confirm email

3.5 Domain Model

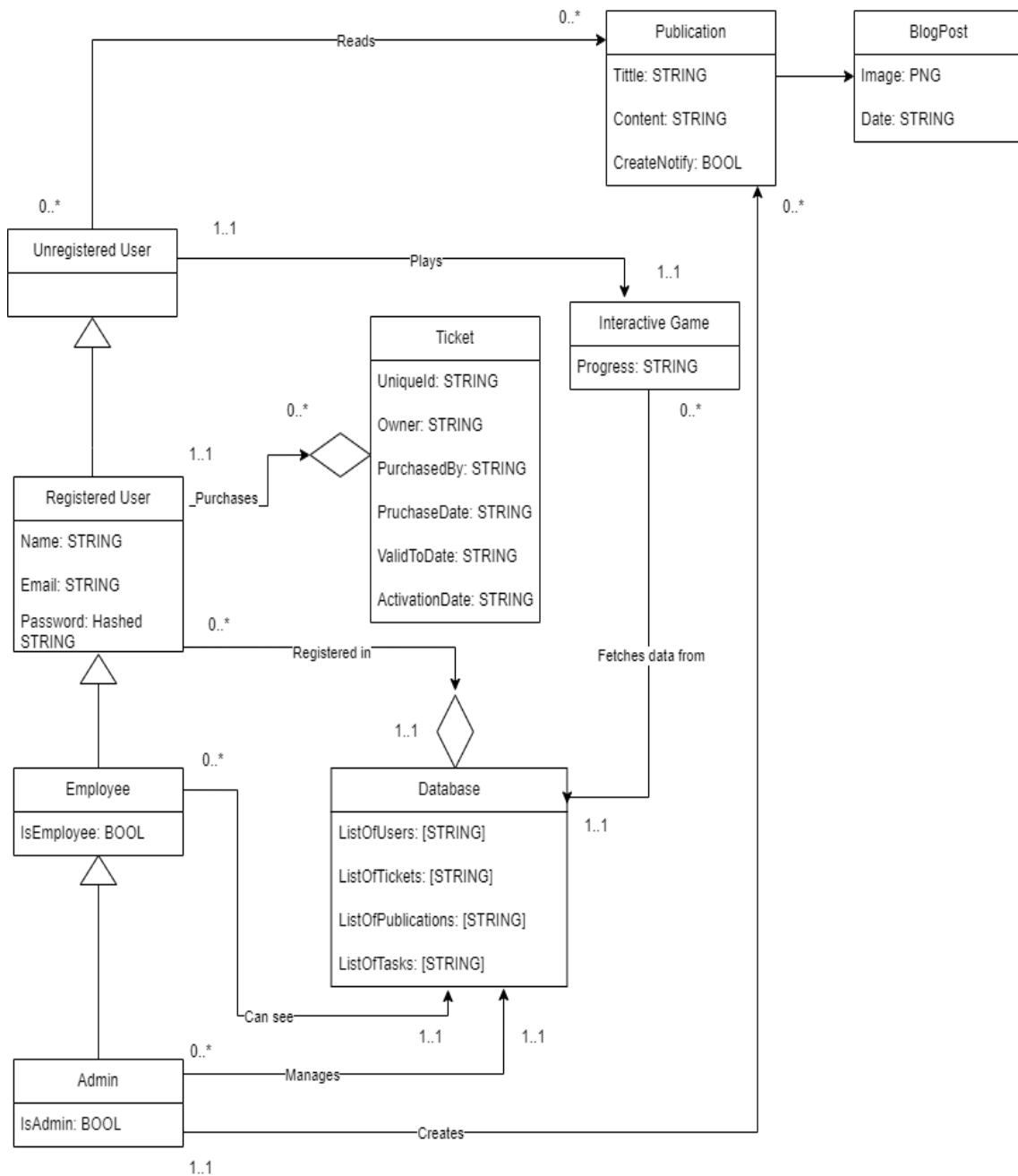


Figure 5: Model of the domain model.

This domain model was created to show how everything is connected in the application. The entry point used in this brief explanation will be the unregistered user. We see that this kind of user has no information stored, and this is meant to represent users who may only use the application to read blogs, play the games, or receive news. The unregistered user can then register their information and become a registered user, which inherits all possible actions from the unregistered user. This user also unlocks the ability to purchase tickets in their name. Furthermore, there are two additional users, this is the employee

and the admin. Employees can see the contents of the database, like tickets purchased, and their owners, but admins have the ability to manage this database. Admins have the ability to manage tickets that have been purchased, this includes removing a ticket from a user or giving them one, for instance in the case where a user may have accidentally used an activation code on themselves instead of gifting it. The admin is also able to upload publications which is a blogs. The interactive game is a feature any user has access to. It will connect to the database to pull any relevant data needed for the games.

4 Developmental Progress

4.1 Methodology

4.1.1 Choice of Software Development Methodology

We originally used the agile method of Scrum. We chose this because we wanted to be able to adapt our product to the product owner as we went, and this is best achieved by using an agile method. Scrum allowed us to keep a lot of structure to the development but also allows for changes to long-term plans easily.

Later on, as the schedules of the group became more jumbled as the planning phase was finished, we swapped over to a combination of Scrum and Kanban, which worked much better for our group. We chose to keep the role delegation from Scrum and the backlog, but instead of organizing implementation in sprints, we swapped over to the Kanban board. We utilized one of the functions of GitLab to create an issue board and used labels to keep track of the progression of each individual issue or task. We keep a weekly physical meeting, akin to a scrum retrospect, where we explained what we had worked on, and what we had finished, and occasionally we would here work together to resolve issues that required more attention.

This proved to be an excellent choice as we had to undergo changes to the application as we went, and our original Gantt chart schedule was essentially flipped so those things we had planned to do later were done early, while the earlier things were solved later.

We used Scrum with weekly sprints when we started out, and when we moved to our hybrid method, this length stayed for the status meetings. Our backlog always made it easy to see what should be made room for and we used the issue board on GitLab to show which tasks each individual on the group was working on. These issues had four categories "open, in development, testing, and important". Open acted as the sprint backlog, in development was for started tasks, testing for finished tasks that need review. Important was reserved for any tasks that need to be resolved quickly or in the case of a bottleneck. These issues were tracked in the commit messages, as well as in the time charts in Appendix [H]. For instance, we had an issue revolving around making content for the report, like this:

Write content for the report

#40 - created 2 weeks ago by Meat

Development

Figure 6: Screenshot of issue 40.

And this was reflected in the time sheets we had, here we commented on what we worked on that day, color coordinated with the most fitting category, and added in hours worked. For this example, we can see that it's green, which means the work day was mostly related to documentation. The hours spent working that day was 8.

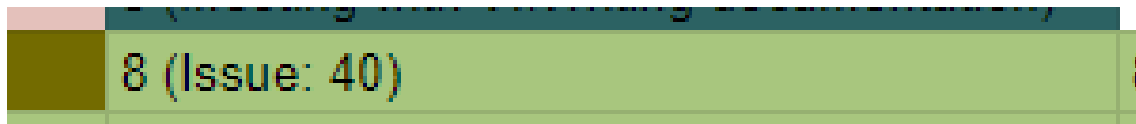


Figure 7: Screenshot of a specific worked day in the time sheets.

4.1.2 Sprint Meetings before the Hybrid Solution

Sprint planning meeting Sprint start was every Tuesday from 11.30-13.00 as we had week-long sprints. Here we worked out what the focus of the coming sprint would be, and pulled tasks from the backlog. We also identified any issues that needed to be classified as important, or if any issue required several people to be worked on.

Sprint review meeting Sprint review meetings were held Tuesdays from 10.00-11.30. The goal of these meetings were to review our work in the past sprint, especially to identify potential problems. We also used what we learned to better plan ahead and create a better product. If anyone learned something useful during the sprint, this was an arena to share that.

Sprint retrospective meeting Sprint retrospective meetings were held every other sprint before sprint reviews. The focus of the meeting was to review the sprints, to see if the length will need to be changed, or if issues are too big or small. It was during one such meeting that the group decided to swap to a hybrid Kanban solution.

4.1.3 Sprint/Kanban Meetings after the Hybrid Solution

Progress Meeting This meeting was held Tuesdays from 10.00-13.00. The goal of these meetings was to showcase the work done so far and discuss further development and delegations.

4.2 Meetings

4.2.1 Mentor Meetings

We had weekly meetings with our mentor Frode Haug. These meetings were primarily used as a forum to discuss methods, report writing, and anything else we need someone else to spar with. On the occasion when we had little or nothing to report, these meetings were dropped.

4.2.2 Project Owner Meetings

It was important for us to keep an open and continuous dialogue with the project owner. We aimed to have meetings with Vitensenteret every other week. This way we could show progress made, and get pointers as to whether we were heading in the right direction product-wise. We were also able to catch potential issues, problems, or misunderstandings during these meetings.

4.2.3 Internal Meetings

All our meetings were held in accordance with our schedules, and in the case where one person was absent, the other group members sent a summary of what each meeting entailed. The group also wrote meeting minutes for anytime we converged and they can be found in the appendix [G]. From there, you can see we tracked progress, as well as the development of the app. This documentation keeps track of all major decisions we made, and any grievances we had during development.

5 User Interface

User interface is a big part of this application. The user can navigate through four major categories: "Vitensenteret", "Spill", "Billett" and "Innstillinger", using the navigation bar in the application. As the application is meant to be useable on both PC and mobile, the application bar will move based on your screen size. Bottom for phones and left for larger screens.

5.1 Blog Page

The first category "Vitensenteret" we refer to as the blog page. The blog page contains a list of articles that employees at Vitensenteret have written. Clicking on an entry will expand the card and show the full image and text of the entry. If the user is logged in as an employee of Vitensenteret they can see a top bar to enter the admin part of the page. On the admin page the user has access to create, edit and delete blog entries.

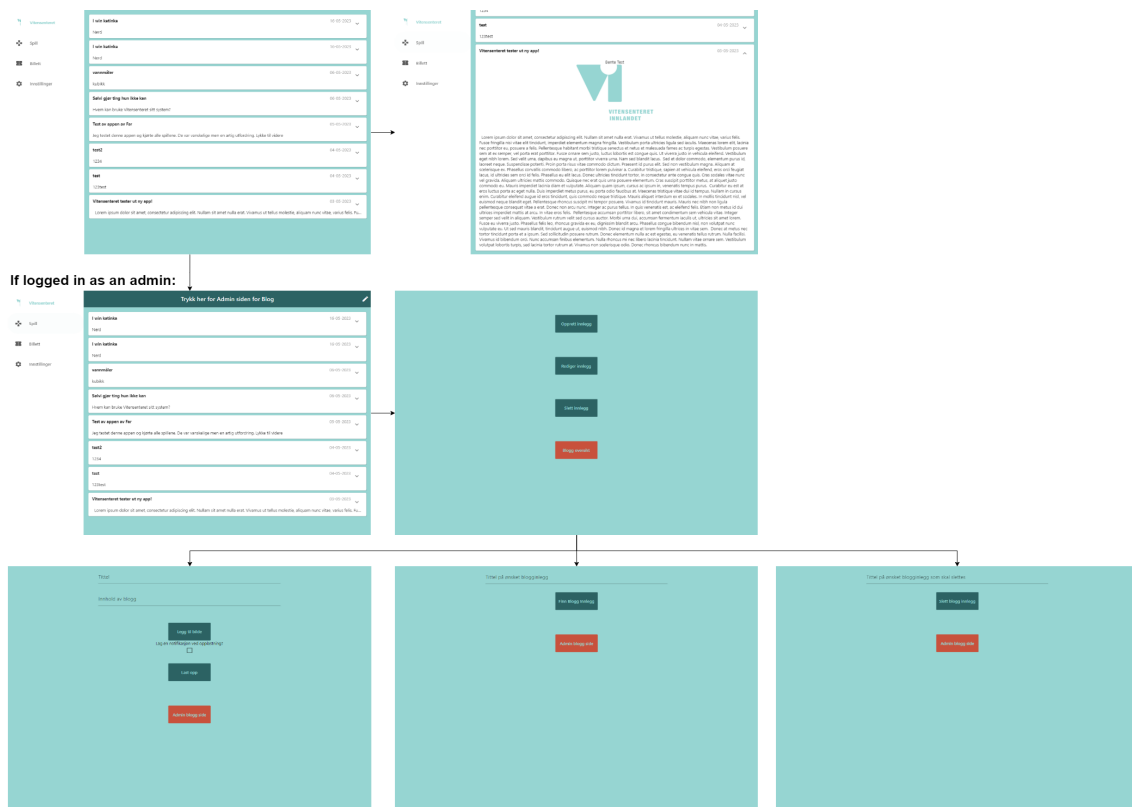


Figure 8: Final design of web version of blog pages

5.2 Game Page

The game page contains the brain teasers of the application. Clicking on one of the large buttons will bring you to the individual game pages.



Figure 9: Final design of web version of game pages

5.2.1 Viten Ord

Viten Ord works similarly to a game called Mastermind[13]. This game is about guessing a pattern of colors, and while doing so, being given hints of whether the pieces you placed are the correct color and the correct placement. Instead of using colors, this game uses letters. The app fetches a random word and lets the user guess it. If the user guesses the wrong letter, it will turn dark blue. If the user guesses the right letter, but places it at the wrong location, it turns yellow. And lastly, if the user guesses the right letter in the right location it turns green. Below is a screenshot of the game in action.



Figure 10: Screenshot of the game "Viten Ord".

5.2.2 Tower of Hanoi

In addition to this game, we created a replication of Hanoi's Tower[14]. This game presents the player with three rods. On the rod to the left are four pieces in ascending sizes. The player has to move the pieces to the rightmost rod. There are two rules, you cannot place a bigger piece on top of a smaller piece, and you cannot move a piece that is underneath another piece.

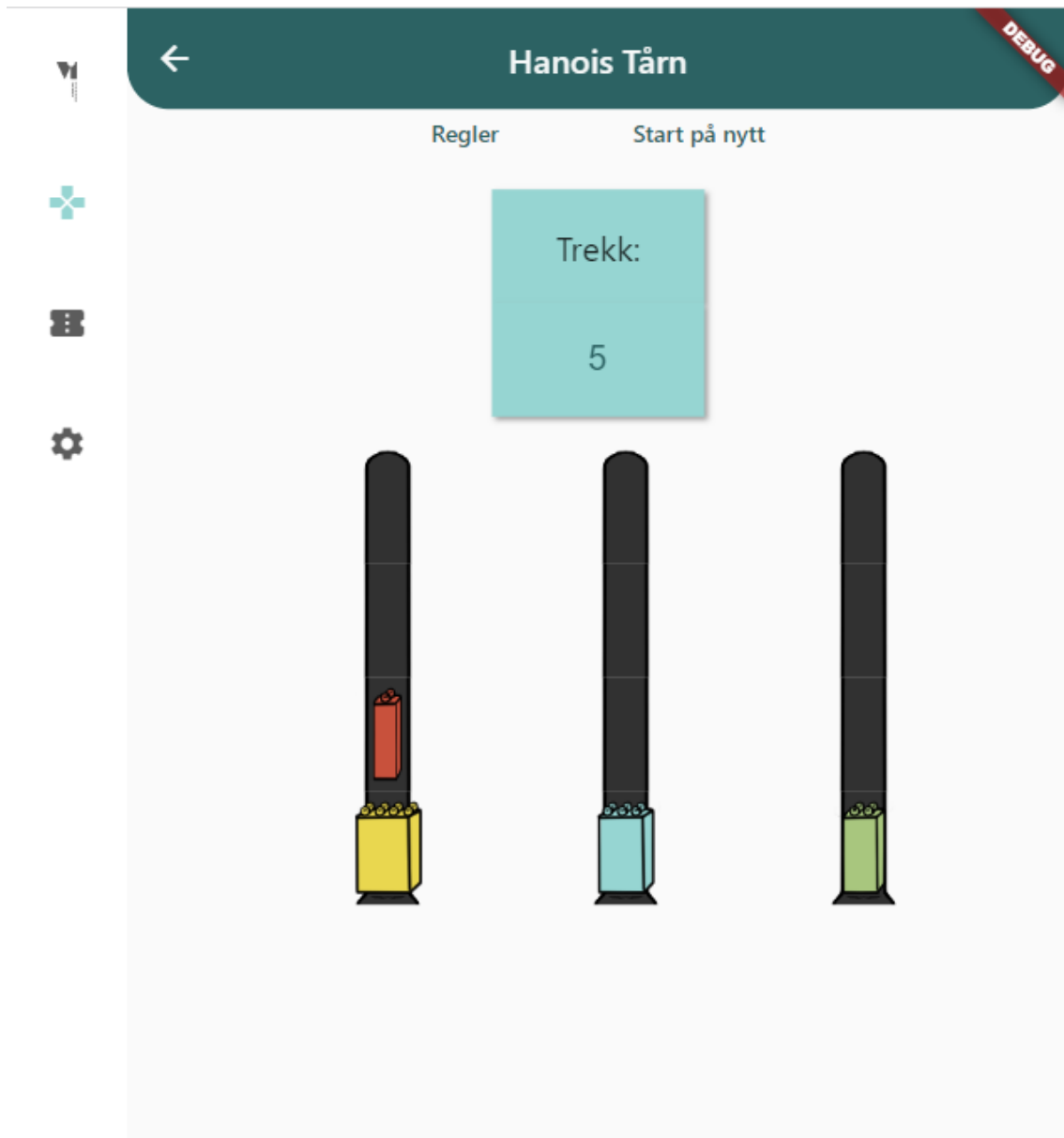


Figure 11: Screenshot of the game "Hanoi's Tower".

5.2.3 Viten Kode

This game takes inspiration from the Lego-Mindstorms coding block ideas[15]. Where you can string different coding blocks together as a sort of primitive way to code. However, developing a proper game like this would be outside the scope, and time set aside for these games, and a smaller version was created, where a task would be presented, and the user could slide the block so that the code produced the relevant result. The tasks are fetched from Firebase, however by the time of delivery we had not received any data for this game, and thus only the example task can be played. More on this in the chapter "Discussion and Further Development".

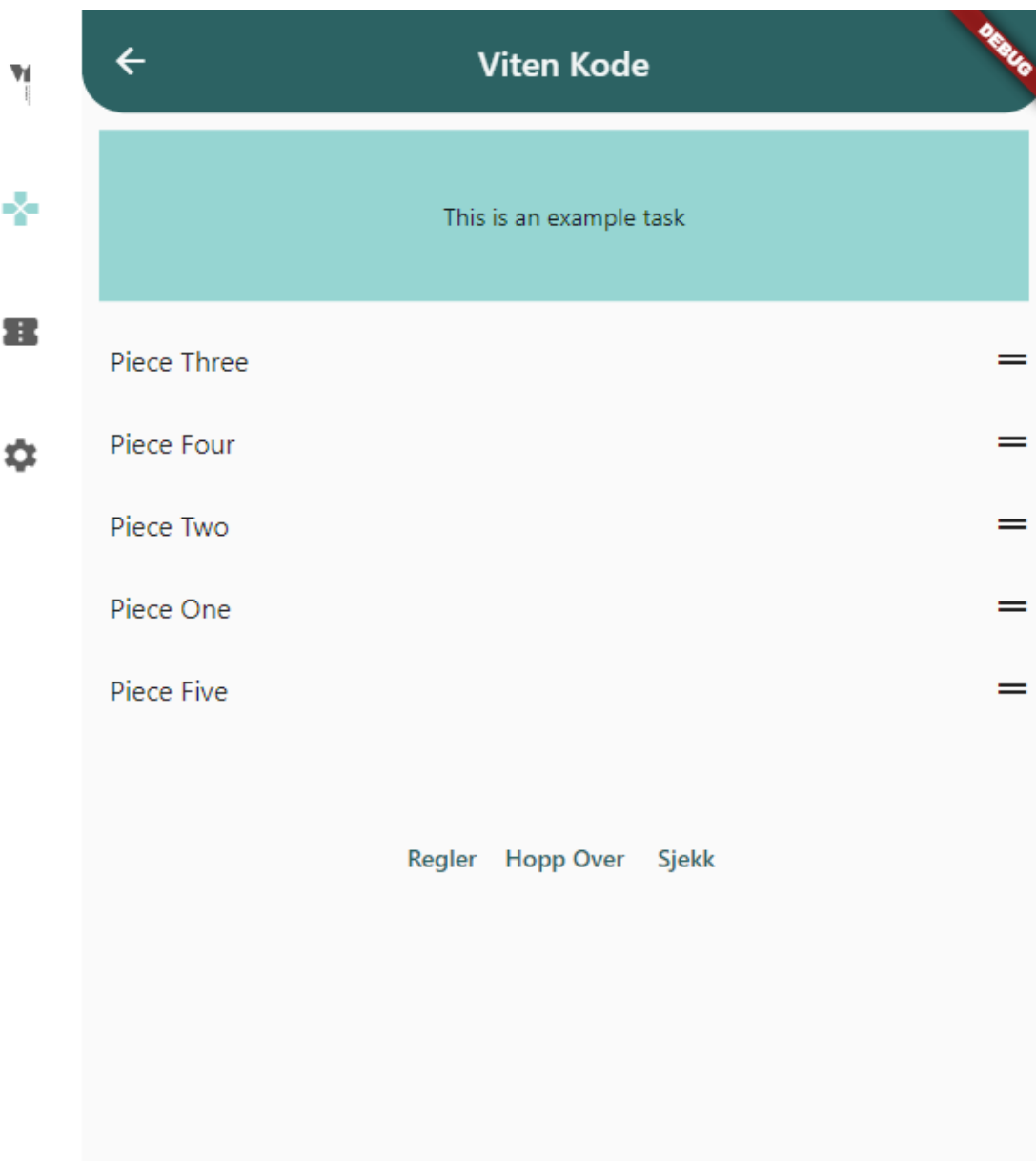


Figure 12: Screenshot of the game "Viten Kode".

5.3 Authentication Page

When a unauthenticated user tries to open the ticket or settings page they will be met with a login page, as both of these pages require user information to function. The login page allows the user to input a username and password to login. If the user enters valid information they will be brought to the page they were trying to access. If the user doesn't have an account they can click "Registrer bruker" to be brought to the registration page. In the registration page the user enters their first name, last name, email, phone number, date of birth and password. Password has to be repeated to ensure the user didn't mistype it. The user also has to add a picture from their device to help with identification.

After registering a new account the user will be brought to the login page again where the user can use their newly created account to log in.

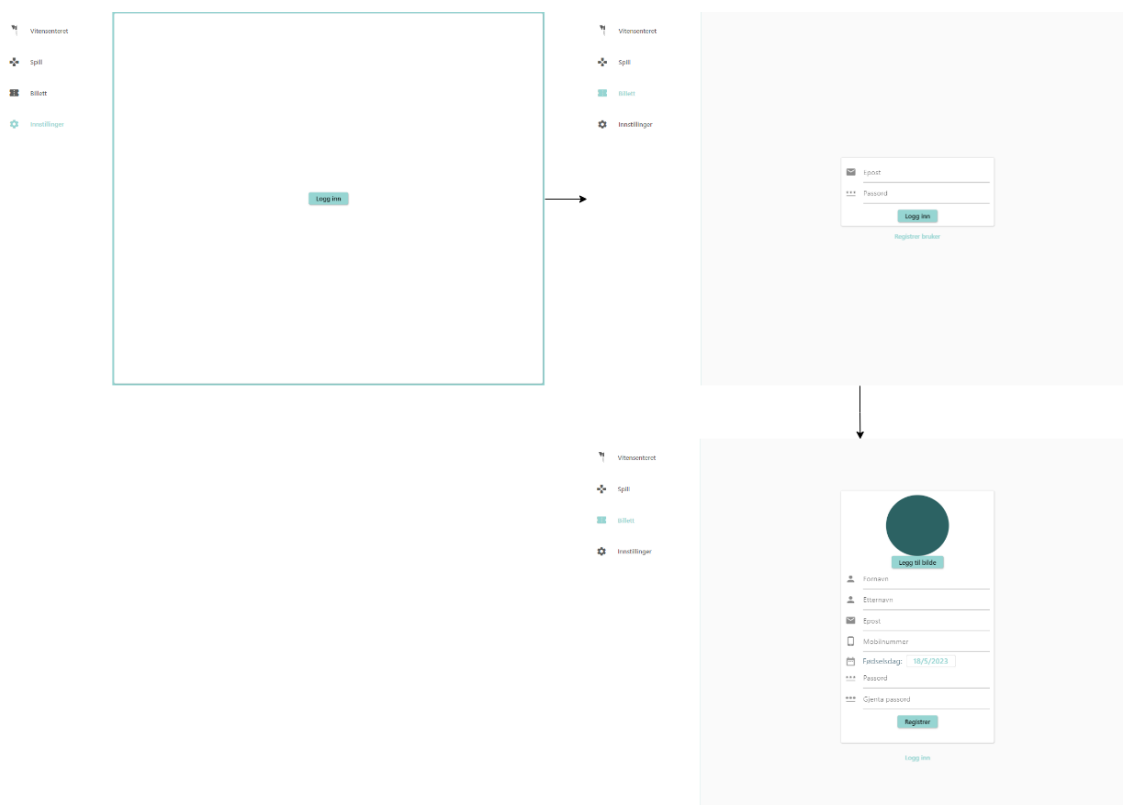


Figure 13: Final design of web version of login and registration pages

5.4 Ticket Page

The ticket page is divided into two parts, the customer part and the employee part. As the phone application is meant for visitors and the web application is meant for employees. Both parts have only been designed with their respective devices in mind. The user will automatically be brought to the correct part based on their device type.

5.4.1 Customer Ticket Page

The customer ticket page gives the user an overview of their purchased tickets, both valid and expired and the remaining duration of them. This page is created with the purpose of being something you can show to the employee working at Vitensenteret when you arrive to prove you have bought a ticket.

A customer can get more tickets by pressing the large plus sign in the bottom right corner, they will then be brought to a page where they can choose between buying tickets or activating a gift card code. Choosing code will give a popup that prompts the user for a

code.

Choosing to buy a ticket will open a different page where the user chooses the ticket type, amount and ticket activation date. The total price will be displayed in the bottom. Payment is initialized by pressing the "Kjøp" button, this will cause the app give a Stripe popup that prompts the user for all necessary payment information.

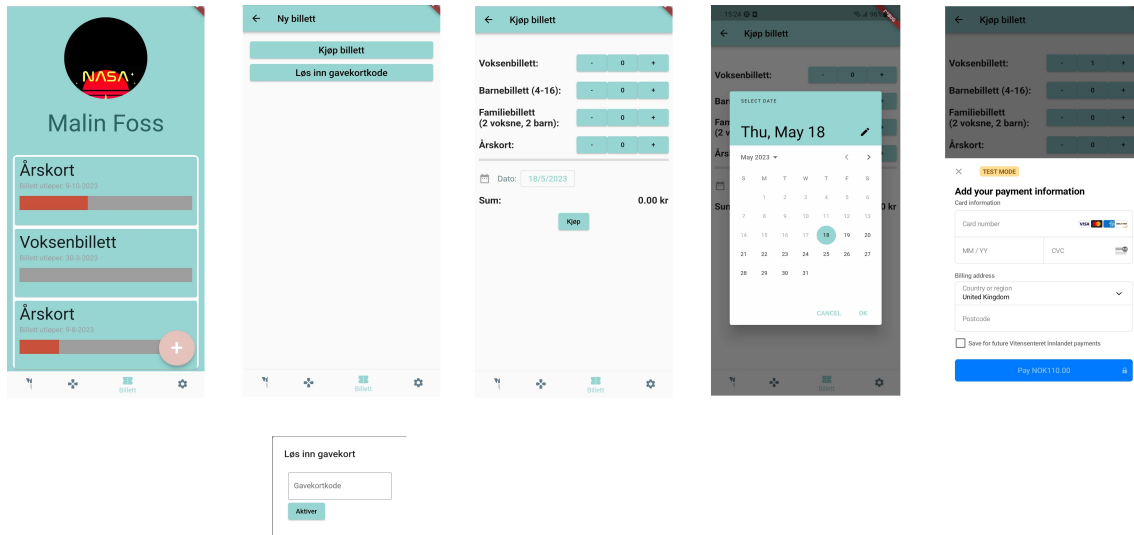


Figure 14: Final design of phone ticket pages

5.4.2 Employee Ticket Page

When employees open the ticket page they will get a list over all the accounts in the system. Every account shows information like name, phone number, email, date of birth and profile picture which can be sorted either alphabetically or by date of birth. From this page the employee can create a new account for people and add tickets to their accounts if they buy them in person. The employee can search for users by name, phone number or email.

There is also a tab with a list of all tickets. The list gives an overview of all the necessary information like: ticket type, date the ticket was bought, date of activation and expiration date, ticket buyer, who activated the ticket and in the cases where the ticket has yet to be activated, its gift card code. In this tab the employee can also search for a customer's tickets to validate their entry using buyer name, gift card code, activation date.

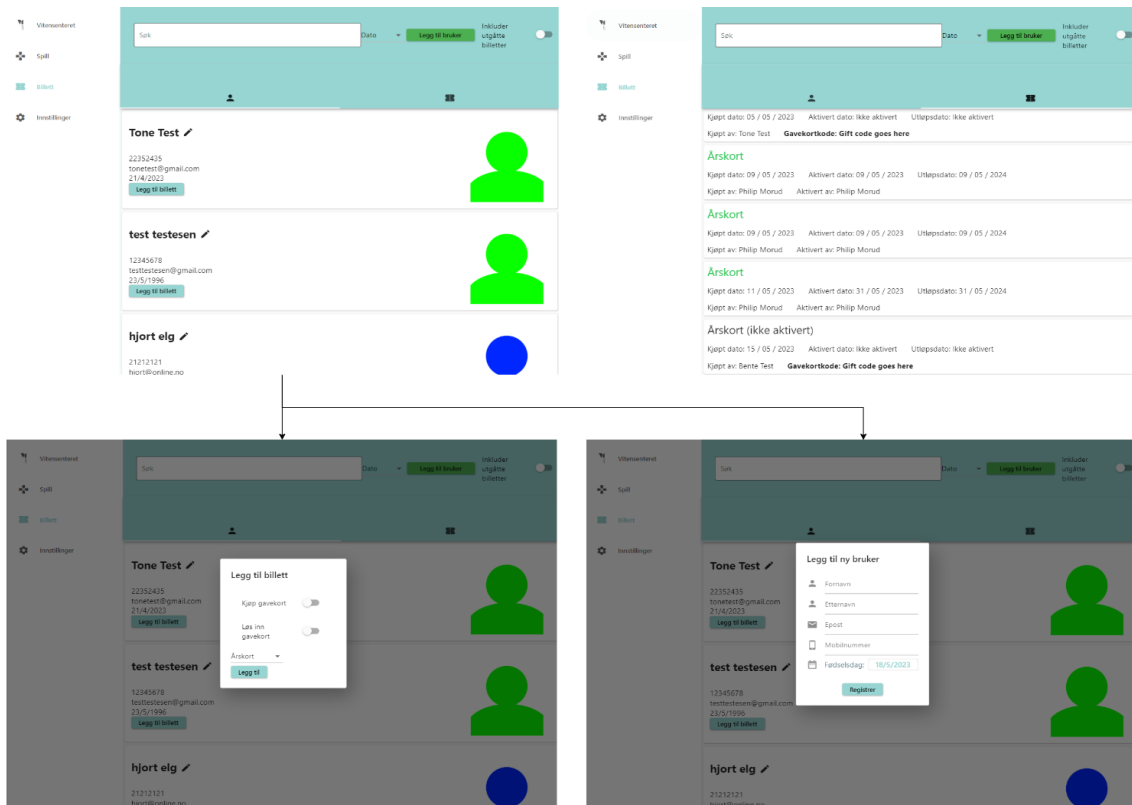


Figure 15: Final design of web version of admin ticket pages

5.5 Settings Page

The settings page gives an easy access for a user to modify personal information when needed. The user can change their profile picture, email, phone number and password. The user can also log out and delete their account.

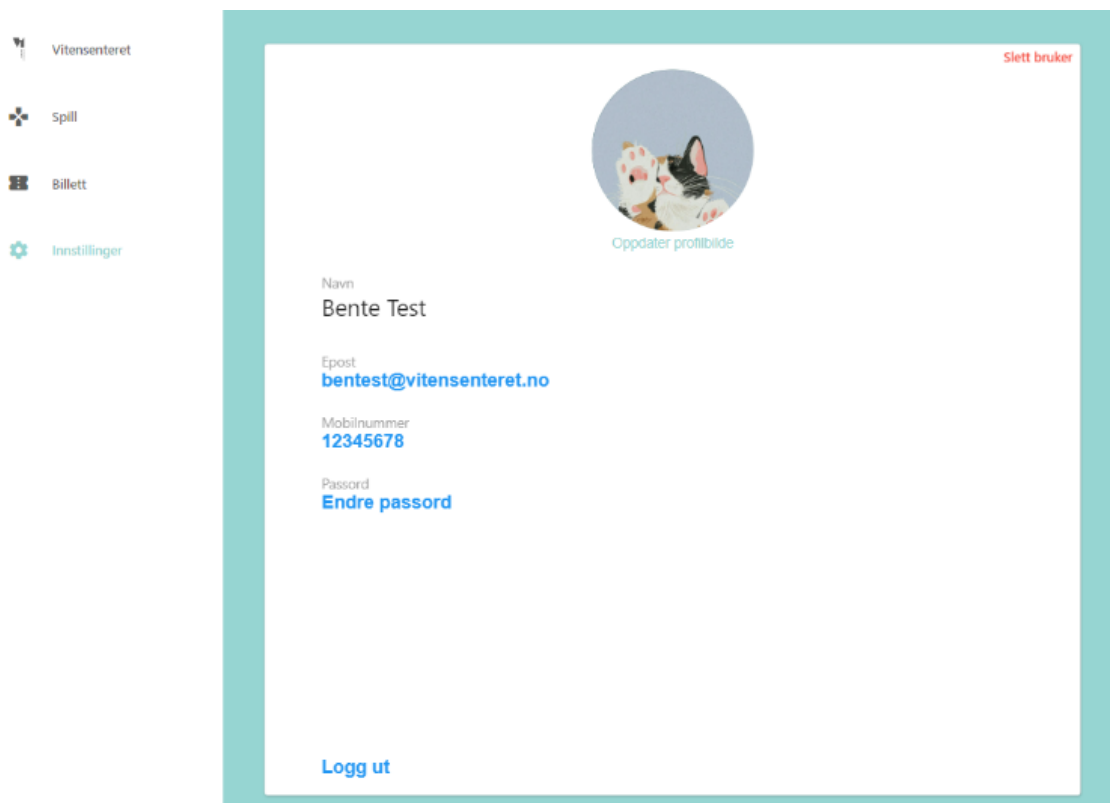


Figure 16: Final design of web version of settings page

As can be seen, the original paper model has been expanded on and some minor changes have happened. When adding a new ticket you get moved to a new page where you get the option to purchase or add a code to activate a ticket. We also made it so the settings are found under a gear icon instead of the profile image of the user, and clicking it leads to a new page instead of an overlay.

6 Technical Design

6.1 Technology

To make the flow of work easy we have a list of technology we use in addition to the software we used to create the application. The following chart illustrates this technology and the general gist of what it was used for.

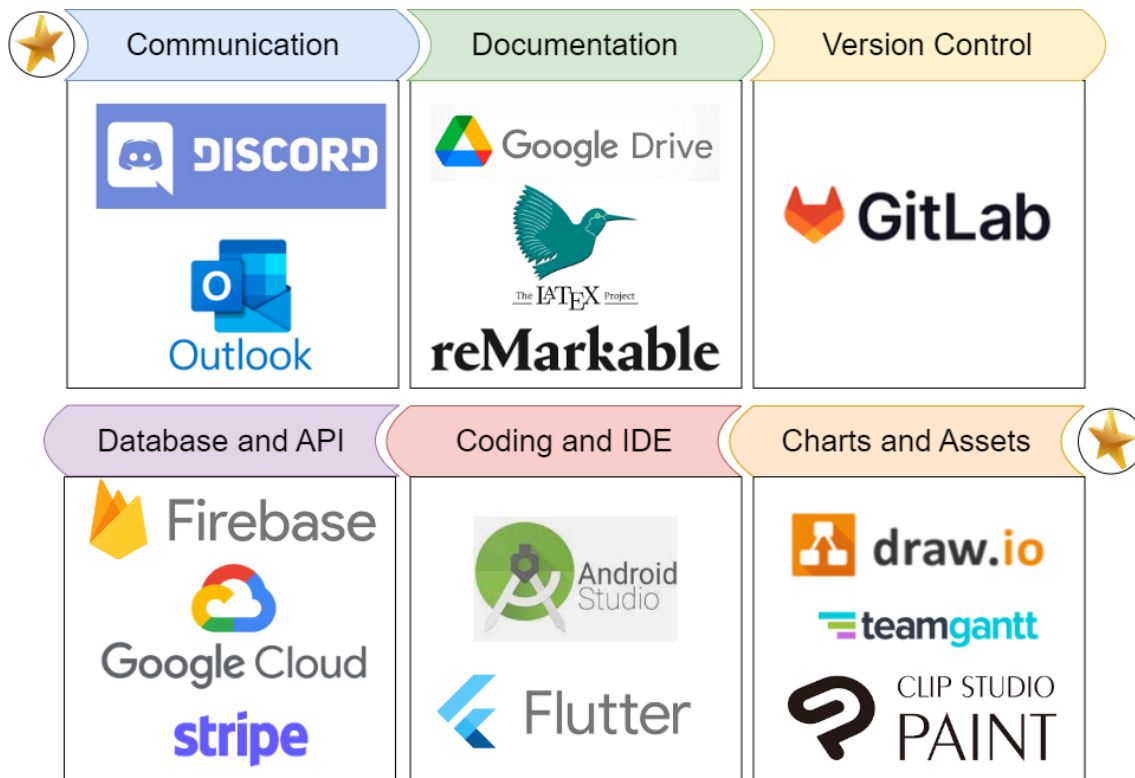


Figure 17: All technologies used in this project.

We used Discord for internal communication within the group, and outlook, email, for any communication with any outside actors, like Vitensenteret or mentor. For documentation we used Google drive to organize all the smaller documents and charts, while we used L^AT_EX for the final report as its much larger with many appendixes. We used reMarkable to keep track of notes during meetings, and it also hosted our backlog. For version control we used GitLab, with its branches it was easy to keep track of any versions and which functionality was added where. For any chart made, or asset created, we used draw.io which is easy to create diagrams. For the Gantt chart, we specifically used a webpage called teamgantt, and for creating assets we used Clip Studio Paint. For creating the software, we used a cross platform coding framework called flutter, and we used the IDE android studio to code. We used Firebase as the database for this project, as well as Firestore functions through google cloud. Stripe was used as the API for the payment solution.

6.2 API

The API consists of two connected parts, Google Cloud Functions in Firebase and Stripe's payment API. The Firebase Functions create a payment intent, which is a collection of information a payment requires to be initialized: amount, client secret, customer id, and currency in addition to storing the status of the current payment [16]. If the user has bought something before, the function will securely fetch payment information from the previous transactions so you don't have to input your card again. After the customer has given a valid payment method in the app, the rest of the necessary API calls to the Stripe API are handled by the Stripe Payment Sheet UI extension, so if your card requires additional authorization it will automatically prompt the customer and return the result. Authentication between the UI and backend is done using a shared secret [17].

6.3 System Architecture

The core of the application is the software we developed in Flutter. This runs many functionalities that are not dependent on any other source. In addition to this, we also connect to the Firebase database and use the Stripe API. Our structure then looks like this.

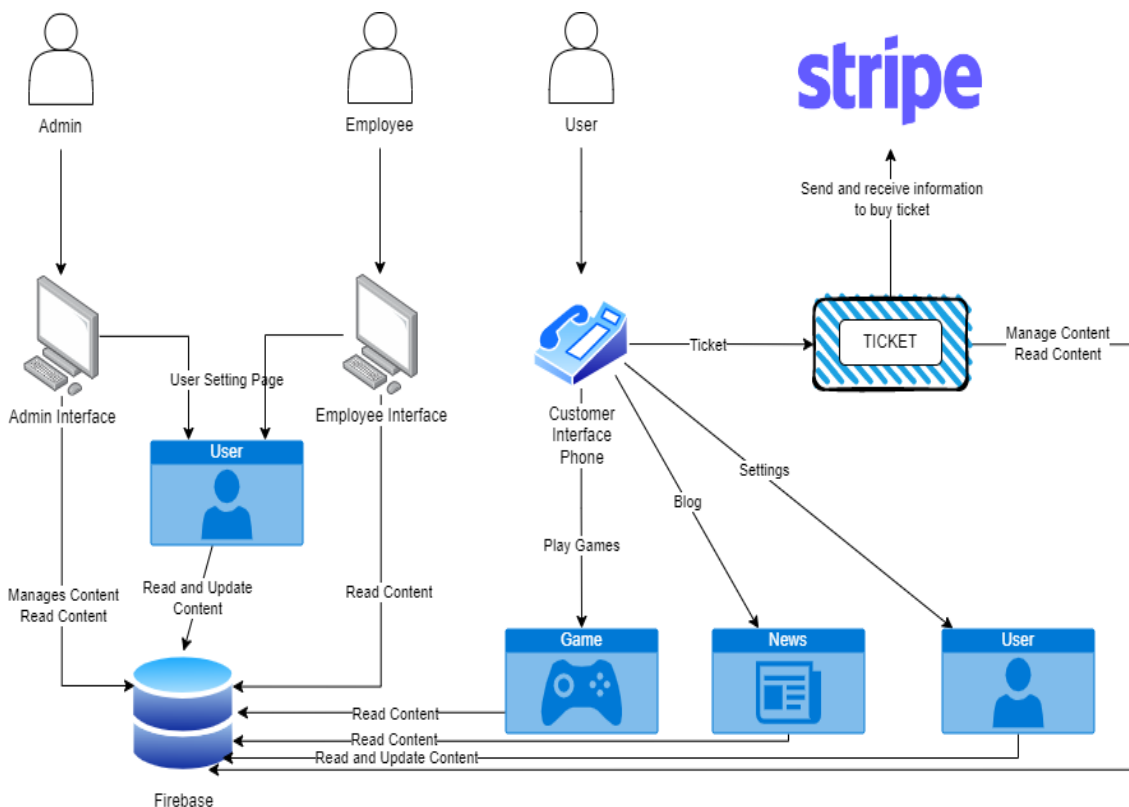


Figure 18: Model of the Database.

This structure shows how everything is connected. The admin is connected to the database through the computer interface, and so is the employee, however, the admin has the option to change the content in the database. In addition to this, they both have access to the database through the settings page where they can change and edit their own user information.

The user is using the interface created for the phone, and this application is also connected to the database through the games, the blog, and the ticket page, all fetching information from the database. The ticket page for the users is also connected to the API stripe which handles the payment. This page then receives the callback from the API and updated the database with the relevant information. The user, like the admin, also has access to the setting page where they can update their own user data.

6.4 Data Storage

For the storage of data, we used Firebase. This is the diagram for the structure of the data, on how they connect and interact. Firebase Realtime Database uses nodes and json-formatting for data [18]. This means that you have to define some top level nodes, that then in turn holds other nodes. Firebase also suggests that the best way to design a database in Realtime Database is by keeping the number of nested nodes to a minimum, and to keep data as close to top level as possible [19]. As the image below shows, we chose to section the top level of into four main categories: posts, tickets, users, and vitenkode.

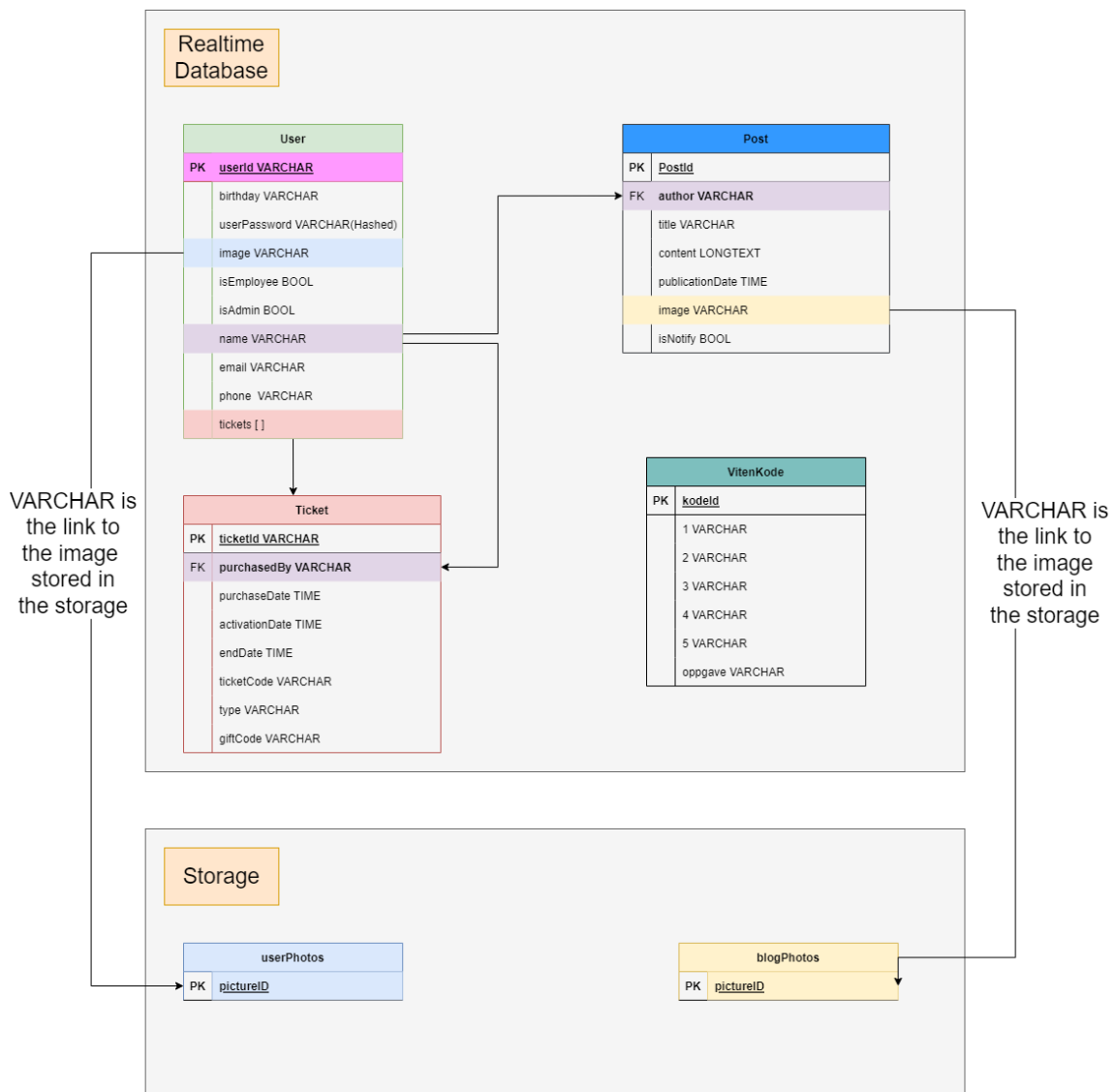


Figure 19: Model of the Database.

Posts keep track of all blog posts, they are assigned a random ID by Firebase and contain all data needed for a blog posts. For each blog post we save the author, title, image, content, publication date, and whether or not to notify users of the app. The content is saved as a string directly in the database, instead of using a separate file and storing it in Firebase Storage. This was done for simplicity since the blog posts are not supposed to be longer than what the strings are capable of holding.

Tickets holds all tickets. Each ticket has a randomly assigned ID from Firebase. The tickets have three different dates, the end date tells when the ticket will expire, the purchase date is for when the ticket was purchased, and the activation date is for gift cards and tickets that has to be activated for a later date. In the case that the ticket is bought and to be used the same day, all these dates will be the same. Each tickets also holds the name of who bought the ticket, and the name of who activated it, as well as potential gift codes and what type of ticket it is.

For users the method for finding unique IDs is a bit different. If the user is created through the app using email and password they get assigned a unique ID in Firebase Authentication, this ID is then used in the Realtime Database as that users ID. If the user is created via the admin page, and therefore does not have a Authentication profile, a randomly assigned ID by Realtime Database is used. Some data about the user is also stored here, mostly for the purpose of making sure that tickets are not being used by others than the owners. There are also flags for whether or not the user is an employee and/or admin. Lastly a list of ticket IDs, this makes the process of getting a single users tickets easier.

The vitenkode portion of the database is dedicated to a game within the application, with a task description and the five options that have to be arranged in the correct order in the game. In the database these are correctly ordered and acts as the solution.

6.4.1 Access and Security

One of the main ways we keep the data in the database available only to those that should see it, is by using rules. Both Firebase Storage and Firebase Realtime Database use rules, albeit in different formats, to regulate who has access to certain data, and what can be done with the data. For example, a user should have access to their own data, but no one else's. At the same time, employees need to be able to add tickets to users, and in some cases be able to edit certain user data. Admins should be able to appoint users to be employees or admins in the system.

```
{
  "rules" : {
    "tickets": {
      ".read" : "auth != null",
      ".write" : "auth != null"
    },
    "users": {
      "$uid": {
        ".read": "$uid == auth.uid || auth.token.email.endsWith('@vitensenteret.no')",
        ".write": "$uid == auth.uid || auth.token.email.endsWith('@vitensenteret.no')",
      },
      ".read": "auth.token.email.endsWith('@vitensenteret.no')",
      ".write": "auth.token.email.endsWith('@vitensenteret.no')",
    },
    "posts": {
      ".read": "true",
      ".write": "auth.token.email.endsWith('@vitensenteret.no')",
    },
    "vitenkode": {
      ".read": "true",
      ".write": "true"
    }
  }
}
```

Figure 20: Rules for Firebase Realtime Database.

Anyone who has logged in can see and edit any of the tickets or vitenkode, the code

further limits who can change what tickets and only admins can change vitenkode. To read or write to a node of user data the user has to be logged in with a user having a matching ID, or be authenticated with an email address from Vitensenteret. To access all users at once the user has to be authenticated with an email address from Vitensenteret. Blog posts can be read by anyone, even non-authenticated users, but can only be edited by users authenticated with an email address from Vitensenteret.

```
1 rules_version = '2';
2 service firebase.storage {
3   match /b/{bucket}/o {
4     //Users can only acces their own image, employees using the @vitensenteret.no mail can access all
5     match /userPhotos/{userId} {
6       allow read, write: if userId == request.auth.uid ||
7         request.auth.token.email.matches("[a-zA-z0-9.!#$%&'*-~/=?^_`{|}~]+@+(vitensenteret.no)");
8     }
9     //Anyone can read blog posts, only @vitensenteret.no mail addresses can change them
10    match /blogPosts/{blogPostId} {
11      allow read: if true
12      allow write: if
13        request.auth.token.email.matches("[a-zA-z0-9.!#$%&'*-~/=?^_`{|}~]+@+(vitensenteret.no)") &&
14        blogPostId.matches(".*\\.txt")
15    }
16    //Anyone can see blog images, only @vitensenteret.no mail addresses can change them
17    match /blogPhotos/{blogPhotoId} {
18      allow read: if true
19      allow write: if
20        request.auth.token.email.matches("[a-zA-z0-9.!#$%&'*-~/=?^_`{|}~]+@+(vitensenteret.no)")
21    }
22  }
23 }
```

Figure 21: Rules for Firebase Storage

Rules for Firebase Storage follows a different pattern. As the image above shows, the rules follow the same logic as with the Realtime Database. Users can access their own images, or any blog content whenever, but only users authenticated with a mail address from Vitensenteret can access all. Unlike Realtime Database that compare strings, Storage uses regular expressions to match email addresses. These rules are broad and are only meant as an assurance that no one has accidental access to data they shouldn't, the application code covers more specialized cases.

As mentioned we use Firebase Authentication to authenticate our users via email and password. By using a service already provided by Firebase we don't have to secure passwords ourselves, and can rest assured that they remain safe. This also allows us to use certain functionalities within the codes itself, for example to check if a user is authenticated, or to re-authenticate easier when that is needed.

6.4.2 Limitations

When fetching data from Firebase Realtime Database we can only get all data under a specific path, but there is no way of sorting this data. This means that there is no way to

get all users born on in a year, without getting all users and then picking out the correct ones in the code. Some choices made in the database design, like the duplication of data, and in the code are made to limit how much data we download to the devices.

Another limitation that changes how the database is set up and how the code has to be structured, is the fact that all fields have to be present when fetching data, or the program crashes. Due to this some fields in some nodes exist, but contain just an empty string.

7 Implementation

7.1 Database

All calls to the database are done from the front end, and because of this we try to keep the number of calls down.

```
Future<int> firebaseRegister (var pass, var user) async {  
  
  //Attempt to create a new user in firebase using Firebase Authenticate  
  try {  
    final credential = await FirebaseAuth.instance.createUserWithEmailAndPassword(  
      email: user,  
      password: pass,  
    );  
    if(credential != null) {  
      return 3;  
    }  
  } on FirebaseAuthException catch (e) {  
    if (e.code == 'weak-password') {  
      log('The password provided is too weak.');      return 1;  
    } else if (e.code == 'email-already-in-use') {  
      log('The account already exists for that email.');      return 2;  
    }  
  } catch (e) {  
    log(e.toString());  
  }  
  
  return 0;  
}
```

Figure 22: Screenshot of registering a user in Firebase Authenticate.

```
Future<int> firebaseSendRegisterInfo (var name, var mail, var image, String uid, String mobileNum, String birthday, File profilePic) async {  
  
  //Location of user profile image  
  final storageRef = FirebaseStorage.instance.ref("userPhotos/$uid");  
  
  String profilePicString = "";  
  
  //Upload profile picture and get link  
  try {  
    await storageRef.putFile(profilePic);  
    profilePicString = await storageRef.getDownloadURL();  
  } catch (e) {  
    log("Could not upload profile image, Error $e");  
    return 1;  
  }  
  
  //Set user data in Firebase  
  try {  
    FirebaseDatabase.instance.ref().child("/users/$uid").set({  
      'name': name,  
      'email': mail,  
      'phone': mobileNum,  
      'birthday': birthday,  
      'isAdmin': false,  
      'isEmployee': false,  
      'image': profilePicString,  
      'tickets': ""  
    });  
  } catch (e) {  
    log("Could not set user data, Error $e");  
    return 2;  
  }  
  
  FirebaseAuth.instance.currentUser!.updatePhotoURL(profilePicString);  
  FirebaseAuth.instance.currentUser!.updateDisplayName(name);  
  FirebaseAuth.instance.currentUser!.updateEmail(mail);  
  
  return 0;  
}
```

Figure 23: Screenshot of saving user data to Firebase Realtime Database.

When registering a user a lot has to happen, the order of these operations are important. First the username (email address) and password is registered with Firebase Authentication. Figure 23 shows the code that is executed right after this registering is done. Here the returned user ID from the first function is used to upload user data to Firebase Realtime Database. The first thing it does is upload the user image to Firebase Storage, and retrieving the download URL. Then all user data is pushed to a new node under users in Firebase Realtime Database, using the user ID as identifier.

```
class TicketData {
    String activatedBy = "";
    String activationDate = "";
    String endDate = "";
    String purchaseDate = "";
    String purchasedBy = "";
    String giftCode = "";
    String type = "";
    String ticketID = "";

    TicketData({
        required this.activatedBy,
        required this.activationDate,
        required this.endDate,
        required this.purchaseDate,
        required this.purchasedBy,
        required this.giftCode,
        required this.type,
    });

    TicketData.fromJson(Map<dynamic, dynamic> json) {
        activatedBy = json['activatedBy'];
        activationDate = json['activationDate'];
        endDate = json['endDate'];
        purchaseDate = json['purchaseDate'];
        purchasedBy = json['purchasedBy'];
        giftCode = json['giftCode'];
        type = json['type'];
    }

    Map<String, Object?> toJson() {
        final Map<String, Object?> data = new Map<String, Object?>();
        data['activatedBy'] = this.activatedBy;
        data['activationDate'] = this.activationDate;
        data['endDate'] = this.endDate;
        data['purchaseDate'] = this.purchaseDate;
        data['purchasedBy'] = this.purchasedBy;
        data['giftCode'] = this.giftCode;
        data['type'] = this.type;
        return data;
    }
}
```

Figure 24: Screenshot of ticket data class.

Figure 24 is an example of how data that frequently move between front end and database is stored. Some data is required, this is the data that comes from the node in Firebase Realtime Database. In the case of the users, only the ID is not required, as this is the name of the node, and not technically part of the user data. It is therefore never pushed or changed in the database, and fetched differently from other user data. There is also functions for marshalling and unmarshalling to and from json, as this is how the data is saved in the database.

```
Future<Int> firebaseGetTickets() async {  
    //Check if user exist in database  
    bool userExists = false;  
    var tickets;  
  
    DatabaseReference ref = FirebaseDatabase.instance.ref("/users/${FirebaseAuth.instance.currentUser!.uid}");  
    final snapshotUser = await ref.get();  
    if (snapshotUser.exists) {  
        userExists = true;  
    }  
  
    //If user does not exist return  
    if (!userExists) {  
        log("User does not exist");  
        return 1;  
    }  
  
    //If yes: fetch users tickets  
    ref = FirebaseDatabase.instance.ref("/users/${FirebaseAuth.instance.currentUser!.uid}/tickets");  
    final snapshotTickets = await ref.get();  
    if (snapshotTickets.exists) {  
        tickets = snapshotTickets.value;  
    }  
  
    List<String> splitTickets = tickets.split(',');  
    var ticketData;  
  
    userTickets = [];  
    //Retrieve tickets  
    for (var t in splitTickets) {  
        ref = FirebaseDatabase.instance.ref("tickets/$t");  
        final snapshotTicket = await ref.get();  
        if (snapshotTicket.exists) {  
            final map = snapshotTicket.value as Map<Object?, Object?>;  
            map.forEach((key, value) {  
                ticketData = TicketData.fromJson(map);  
            });  
        }  
        userTickets.add(ticketData);  
    }  
}
```

Figure 25: Screenshot of function for getting all tickets from a user.

Here we fetch all tickets that belong to a specific user. First we ensure that the user exists in the database, then we fetch that users list of tickets. The list is saved as a comma separated list in a single string, so it needs to be split up before we retrieve the tickets. The tickets are fetched one by one from the database and saved to a local list, there is no way to our knowledge to get several specific nodes in one call.


```

Future<int> firebaseUpdateMail(String newMail, String password) async{
  //Re-authenticate user
  //This is required by firebase before updating user authentication data
  try {
    final credential = await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: currentUser.email,
      password: password
    );
    if (credential.credential != null) {
      await FirebaseAuth.instance.currentUser?.reauthenticateWithCredential(credential.credential!);
    }
  } on FirebaseAuthException catch (e) {
    if (e.code == 'user-not-found') {
      log('No user found for that email.');
```

```

      return 2;
    } else if (e.code == 'wrong-password') {
      log('Wrong password provided for that user.');
```

```

      return 2;
    }
  }
}

//Update user authentication data
try {
  await FirebaseAuth.instance.currentUser!.updateEmail(newMail);
} catch (e) {
  log("Could not update user authentication, Error $e");
  return 3;
}

//Updates to be made
var updates = {
  'email': newMail
};

//Update users data in Firebase
try {
  FirebaseDatabase.instance.ref().child("/users/${FirebaseAuth.instance.currentUser?.uid}").update(
    updates
  );
} catch (e) {
  log("Could not update database, Error $e");
  return 4;
}

//Update local variable
currentUser.email = newMail;

return 0;
}

```

Figure 26: Screenshot of function updating user email.

Here we have the code needed for updating a users email address, this is the most complicated function when it comes to updating data as several changes in several places needs to be done. Before user data can be updated, the user has to be logged in again and re-authenticated. Then the log in information, stored in Firebase Authentication, has to be updated with the new email. After that the data stored in Realtime Database has to be updated. And lastly the locally saved user data, this step is not strictly necessary, but is a nice addition temporarily before all user data is fetched from the database again.

```
Future <int> firebaseDeleteUser(String userID, bool deleteImage, bool deleteAuth) async {  
  
  //Delete profile picture  
  if (deleteImage) {  
    final ref = FirebaseStorage.instance.ref("/userPhotos/$userID");  
    try {  
      await ref.delete();  
    } catch (e) {  
      log("Could no delete profile image, Error: $e");  
      return 1;  
    }  
  }  
  
  //Delete user data saved in Firebase Realtime Database  
  final ref = FirebaseDatabase.instance.ref("/users/$userID");  
  try {  
    await ref.remove();  
  } catch (e) {  
    log("Could no delete user data, Error: $e");  
    return 2;  
  }  
  
  //Delete the user from Firebase Authentication  
  if (deleteAuth) {  
    try {  
      await FirebaseAuth.instance.currentUser?.delete();  
    } catch (e) {  
      log("Could no delete Firebase Authentication, Error: $e");  
      return 3;  
    }  
  }  
  
  return 0;  
}
```

Figure 27: Screenshot of function deleting a user.

Deleting a user is quite easy, but still requires three calls to the database. The first call is to delete the profile image stored in Firebase Storage, the second to delete user data from Firebase Realtime Database, the third deletes authentication data. When authentication data is deleted, the user is sent back out to the log in screen automatically.

7.2 API

7.2.1 Firebase Functions

```
11 // Takes init payment request from a customer in app and returns the payment intent
12 // Payment code copied from https://www.youtube.com/watch?v=BR4sF_VzV0w
13 // github: https://github.com/hadikachmar3/grocery_app_course/blob/master/functions/index.js
14 exports.stripePaymentIntentRequest = functions
15   .region("europe-west1")
16   .https.onRequest(async (req, res) => {
17     try {
18       let customerId;
19
20       // Gets the customer who's email id matches the one sent by the client
21       const customerList = await stripe.customers.list({
22         email: req.body.email,
23         limit: 1,
24       });
25
26       // Checks the if the customer exists, if not creates a new customer
27       if (customerList.data.length !== 0) {
28         customerId = customerList.data[0].id;
29       } else {
30         const customer = await stripe.customers.create({
31           email: req.body.email,
32         });
33         customerId = customer.data.id;
34       }
35
36       // Creates a temporary secret key linked with the customer
37       const ephemeralKey = await stripe.ephemeralKeys.create(
38         {customer: customerId},
39         {apiVersion: "2020-08-27"},
40       );
41
42       // Creates a new payment intent with amount passed in from the client
43       const paymentIntent = await stripe.paymentIntents.create({
44         amount: parseInt(req.body.amount),
45         currency: "nok",
46         customer: customerId,
47       });
48
49       res.status(200).send({
50         paymentIntent: paymentIntent.client_secret,
51         ephemeralKey: ephemeralKey.secret,
52         customer: customerId,
53         success: true,
54       });
55     } catch (error) {
56       res.status(404).send({success: false, error: error.message});
57     }
58   });
```

Figure 28: Screenshot of Firebase function

The Firebase function takes a post request with a body containing the customer email and transaction cost. First the function gets the email from the body and checks if the email is in the list of previous customers. If it's a new customer the function will add them to the list instead. Next, the function creates an ephemeral key for the transaction. The function then creates the payment intent using the amount from the body of the post request. Lastly the function returns status 200 with the secret keys, customerId and the success of the initialization. In the case where the function fails it will instead return status 404, that the initialization didn't succeed and the error message.

7.2.2 Stripe API

```
248     try{
249         //Update sum
250         calculateSum();
251
252         // Create payment sheet
253         final response = await http.post(
254             Uri.parse(
255                 "https://europe-west1-mockaarskort.cloudfunctions.net/stripePaymentIntentRequest"),
256             body:{
257                 'email': currentUser.email,
258                 'amount': sum.toString(),
259             });
260         // Decode and read response
261         final jsonResponse = jsonDecode(response.body);
262         log(jsonResponse.toString());
263
264         // Initialize the payment sheet
265         await Stripe.instance.initPaymentSheet(
266             paymentSheetParameters: SetupPaymentSheetParameters(
267                 paymentIntentClientSecret: jsonResponse['paymentIntent'],
268                 merchantDisplayName: 'Vitensenteret Innlandet',
269                 customerId: jsonResponse['customer'],
270                 customerEphemeralKeySecret: jsonResponse['ephemeralKey'],
271             ));
272         await Stripe.instance.presentPaymentSheet();
```

Figure 29: Screenshot of Stripe API call

The stripe API is mostly handled by the Stripe payment sheet UI extension. The first lines of code are simply gathering information to initiate the payment through the Firebase function. Then using the generated intent it creates a payment sheet and presents it. Calling the API is done in a try-catch block which means that as long as the sheet doesn't return an error the payment has succeeded.

7.3 Mobile Application

7.3.1 Log in and Register Page

Both logging in and registering uses forms to validate the input fields. For logging in it simply checks if both fields, email and password, are filled in before calling a function that attempts to log the user in. If the user is successfully authenticated, they get redirected to the appropriate page. Registering is a lengthier process, and requires a good portion more information, but uses the same basic steps. The fields needed were a requirement placed by Vitensenteret. A field in the form looks like this is the code:

```

TextFormField(
  keyboardType: TextInputType.emailAddress,
  decoration: const InputDecoration(
    hintText: 'Epost',
    icon: Icon(Icons.email)
  ), // InputDecoration
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'Skriv inn epost';
    }
    if (!validateEmail(value)) {
      return 'Feil format på epost';
    }
    emailAddress = value;
    return null;
  },
), // TextFormField

```

Figure 30: Screenshot of email register field.

Where the validator gets triggered by pressing the 'Registrer' button. For the email field there are two things that can be wrong, either the field is empty or the email does not follow the email format. The email format is checked with a function that uses regular expressions for this purpose

```

bool validateEmail(String mail) {
  if (!mail.contains(RegExp("^[a-zA-Z0-9. a-zA-Z0-9.!#$%&'*+~/=?^_{}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+"))) {
    return false;
  }
  return true;
}

```

Figure 31: Screenshot of function for validating email formats.

When the register button is pushed it validates all fields before calling the two functions needed to authenticate and save all user data. More on that in the section about the database.

7.3.2 Ticket Page

This page shows the profile picture and name of the logged-in user, as well as all their tickets. The page is designed to function as a users ticket to Vitensenteret Innlandet, and the tickets, therefore, feature expiration date, type, and bar showing a count down to the ticket expires. There is also a button that redirects the user to the payment page, which is where they can purchase new tickets.

7.3.3 Settings

The settings page allows a user to edit user information, delete their user or log out. The user can change their profile picture, here we use the same image picker as we do for registering a user. For all other fields that a user can change, the values act as buttons and when clicked opens a pop-up dialog. For example will the field for changing email contain two fields, and a button for saving. The fields contain the new email and the current password. The password is needed for re-authentication. Once the button is pressed, a function is called that saves this data to the database.

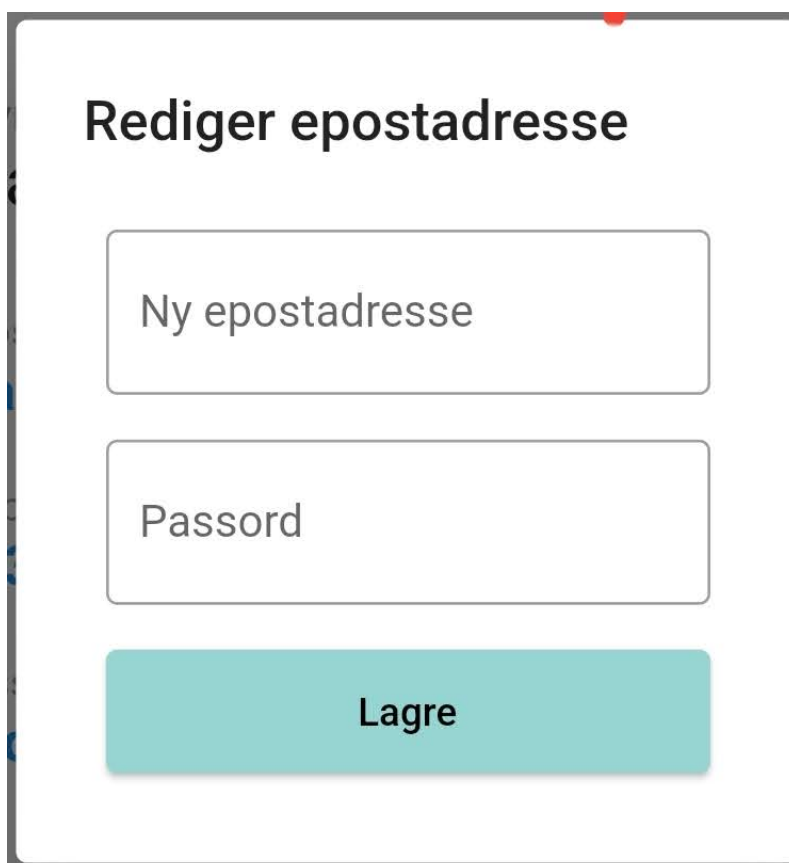


Figure 32: Screenshot of page for editing email address.

7.3.4 Blog

The blog is a very simple scrollable list containing unique members. The blog pulls down data from Firebase, and organizes it by date to display the most recent blog post on the top of the page. Each blog post is summarized with a title and the beginning of the blog contents, while clicking on this summary will expand the widget, showing all the contents of the post. If the user is logged in as an admin, an app bar will appear at the top, with a button to take them to the admin page for the blog.

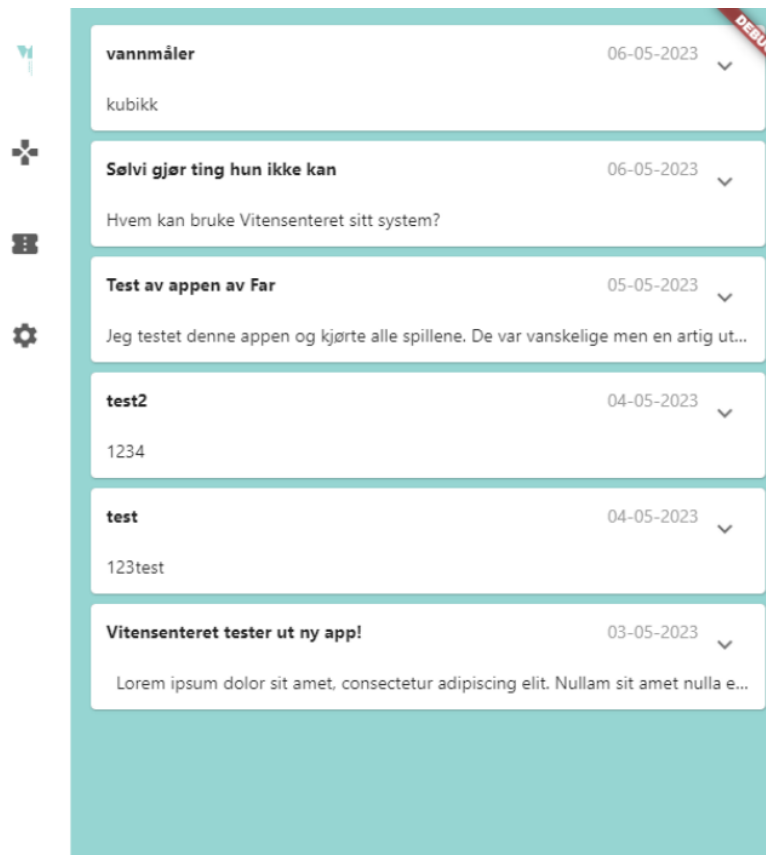


Figure 33: Screenshot of the blog.

7.3.5 Payment Page

The payment page consists of a simple list of the different types of tickets where the customer can freely choose how many they want of each ticket. Then the customer picks the date they want the tickets to activate on with the default being the current day. The total sum is updated as the customer adds the tickets. Once the customer is happy with their choice they press the "Kjøp" button which will cause the app to initialize the payment.

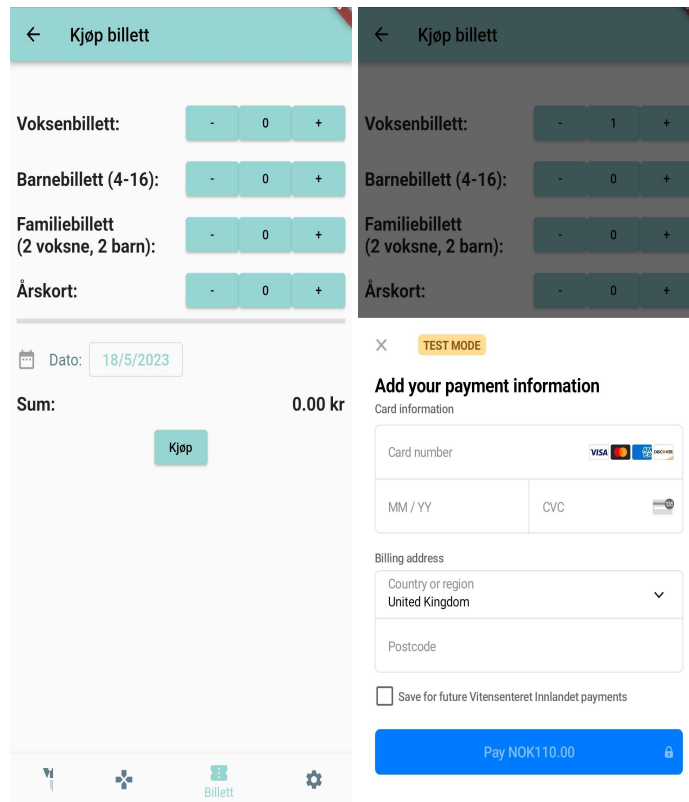


Figure 34: Screenshot of the payment page with and without the payment sheet

With the payment initialized the customer gets a Stripe payment sheet popup. In this sheet they enter the required information for the payment and if necessary, authenticates the payment. If the customer wishes they can save their card for future purchases. If the customer chooses to do so, the next time they initiate a purchase they will first be prompted if they want to use one of their stored cards.

7.4 Web

The web interface is solely for employees at Vitensenteret Innlandet, and the focus is therefore on functionality over looks. We make use of a tool that Flutter makes available with the project, called *kIsWeb*, to figure out whether someone is using the web interface or the mobile interface. There are primarily two pages that are only available on the web interface, to access them the user also has to have been authenticated with a `@vitensenteret.no` mail address, as a secondary precaution.

7.4.1 Ticket Page for Admins

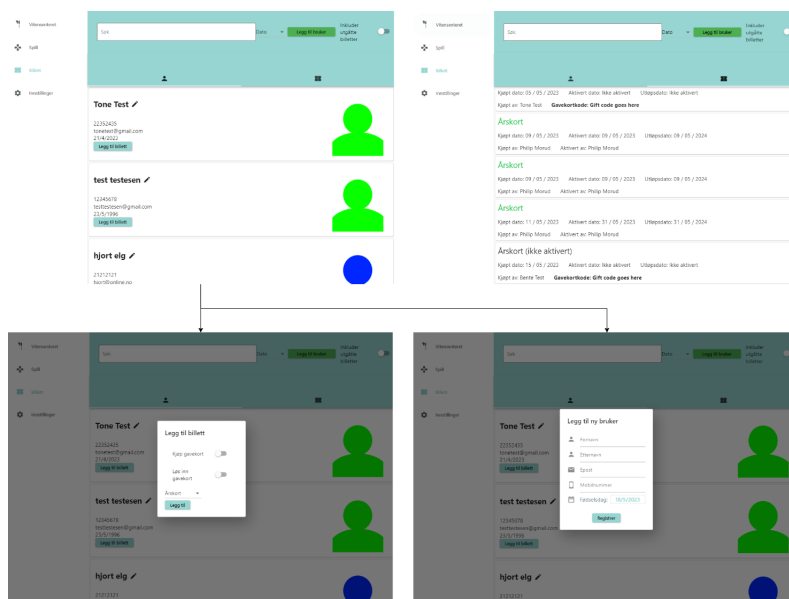


Figure 35: Screenshot of the ticket page for Admin.

This page is first and foremost meant as a tool for employees during visiting hours at Vitensenteret. The page shows all users and all tickets in the system, and lets the user search, sort, include/exclude expired tickets, add users, add tickets and edit basic information about a user. When fetching code from the database it is saved in two lists, one for users and one for tickets. In the admin ticket page there is two additional lists, both used to keep temporary lists of users and tickets when the user searches or sorts the lists.

```

searchFor(String search) {
  if (search == "") {
    setState() {
      adminUserData = allUserData;
      adminUserTickets = allUserTickets;
      isSearching = false;
    });
  } else {
    adminUserData = [];
    adminUserTickets = [];
    isSearching = true;

    setState() {
      for (var element in allUserTickets) {
        if (element.activatedBy.toLowerCase().contains(search.toLowerCase())) {
          adminUserTickets.add(element);
        } else if (element.purchasedBy.toLowerCase().contains(search.toLowerCase())) {
          adminUserTickets.add(element);
        } else if (element.giftCode.contains(search) && element.endDate == "") {
          adminUserTickets.add(element);
        }
      }
      for (var element in allUserData) {
        if (element.name.toLowerCase().contains(search.toLowerCase())) {
          adminUserData.add(element);
        } else if (element.email.toLowerCase().contains(search.toLowerCase())) {
          adminUserData.add(element);
        } else if (element.phone.contains(search)) {
          adminUserData.add(element);
        }
      }
    });
  }
}

```

Figure 36: Screenshot of the search bar code.

The search bar allows employees to find specific users or tickets. It is possible to search for names both on users and who activated or purchased a ticket, emails, phone numbers and gift codes. Above is the code that runs every time the search bar is changed. When the search bar is emptied, the temporary lists holding users and tickets are set back to hold all users and tickets. When something is typed in the search bar it empties the temporary lists, before looking through all users and tickets and adding them to the lists if they match the criteria. Here we make sure to use lower case for everything, so the search is not case sensitive. Doing it this way also renders the search result in real time as the user is typing.

```

sortBy(String type) {
  if (type == 'Date') {
    adminUserTickets.sort((a, b) => int.parse(a.purchaseDate).compareTo(int.parse(b.purchaseDate)));
    adminUserData.sort((a, b) => a.birthday.compareTo(b.birthday));
  }
  if (type == 'Alfabetisk') {
    adminUserTickets.sort((a, b) => a.activatedBy.toLowerCase().compareTo(b.activatedBy.toLowerCase()));
    adminUserData.sort((a, b) => a.name.toLowerCase().compareTo(b.name.toLowerCase()));
  }
}

```

Figure 37: Screenshot of the code that sorts users and tickets.

This part makes use of existing sort functions in Dart, and sorts the temporary lists by

either date or alphabetically.

```
excludeIncludeExpired() {
    DateTime endDate = DateTime.now();
    if(!isSearching) {
        if (!showAllTickets) {
            adminUserTickets = [];
            for (var element in allUserTickets) {
                if (element.endDate == "") {
                    adminUserTickets.add(element);
                } else {
                    endDate = DateTime(int.parse(element.endDate.substring(0, 4)),
                        int.parse(element.endDate.substring(4, 6)),
                        int.parse(element.endDate.substring(6, 8)), 23, 59);
                    if (endDate.compareTo(DateTime.now()) > 0) {
                        adminUserTickets.add(element);
                    }
                }
            }
        } else {
            adminUserTickets = allUserTickets;
        }
    }
    else if (!showAllTickets) {
        List<TicketData> temp = [];
        for (var element in adminUserTickets) {
            if (element.endDate == "") {
                temp.add(element);
            } else {
                endDate = DateTime(int.parse(element.endDate.substring(0, 4)),
                    int.parse(element.endDate.substring(4, 6)),
                    int.parse(element.endDate.substring(6, 8)), 23, 59);
                if (endDate.compareTo(DateTime.now()) > 0) {
                    temp.add(element);
                }
            }
        }
        adminUserTickets = temp;
    }
}
```

Figure 38: Screenshot of the code that includes or excludes expired tickets.

To include or exclude expired tickets from the ticket overview page we use a switch. The code accounts for three different situations, and excludes a fourth. If the user is not searching for something at the moment, and turns the switch of, then the temporary tickets lists becomes identical to the list from the database. If the user is again not searching for anything, but turns the switch on, all tickets that have an older date than today is excluded. Tickets without an end date is included here as well. If a users in searching the switch only works one way, turning it on. When searching and excluding expired tickets, the code will do the same as when not searching and excluding expired. There is no way no then include all tickets again if the user is searching for something, in that case they have

to end the search and start again.

7.4.2 Blog Page for Admins

So that news can be uploaded to the app without manually adding it into the database, we created an admin page for handling blogs. This page is reachable with a button on the blog page, and this button is only visible if the user logged in is an admin.

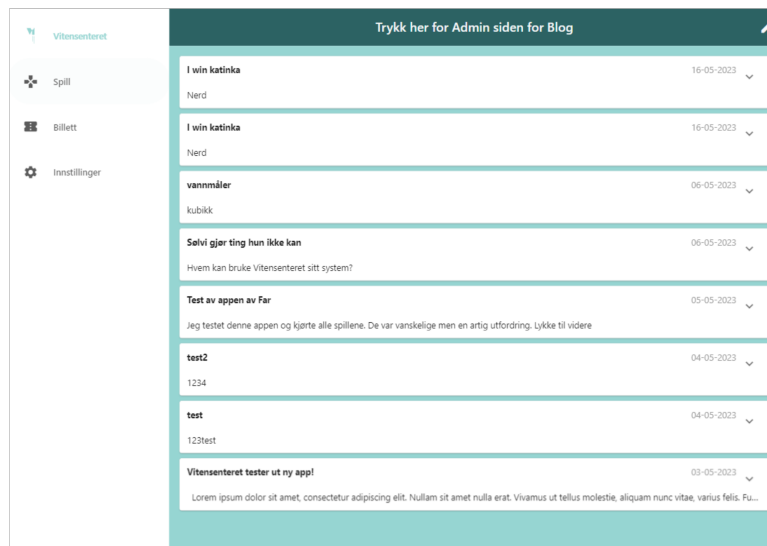


Figure 39: Screenshot of how the blog page looks for an admin.

Clicking this leads to the admin blog page, where the admin can manage blogs. There are three functionalities here, upload a blog, edit an existing blog, and delete a blog. Uploading a blog takes the user to a form where they fill in the contents of the post. Here they create a title and content, and choose if they want to upload an image. Information about date and author gets added automatically. There is also an option to create a notification when the blog is uploaded, but the functionality for the notification system was not completed for this thesis. More on this in "Further Development".

Editing and deleting blogs both function similarly, as when the admin clicks either of these options they are taken to a page where they will have to search for the blog post they want to affect. They search by title name, and in the case of edit is taken to editing page for that post. In the case of deletion there will be a popup asking if the admin really wants to delete this blog post, to make sure no posts are accidentally deleted.

7.5 Game Implementations

One of the ways to retain users in the app was to create a collection of brain teaser games. Vitensenteret specifically requested the three games described in the Requirement Specification. As a way to learn the language, we elected to follow a tutorial[20] to both learn Flutter and get some help to create the game as we were all new to Flutter at this time. Later on, we developed all the games without any assistance.

7.5.1 Viten Ord

While much of the code for this game was written following the tutorial, the game developed in that tutorial had a lacking algorithm to check the solution and compare the inputted letters. That is why we improved it. How it works is that the code essentially has a list with the solution, and when it checks to see what the user has inputted it, compares that list to the solution, one letter at a time. The first thing the algorithm does is to check if a letter is both correct as well as in the correct position Figure: 40.

```
// Update word with colors
setState() {
  if(currentWordLetter == currentSolutionLetter) {
    letters.add(currentWordLetter);
    _currentWord!.letters[i] =
      currentWordLetter.copyWith(status: LetterStatus.correct);
  }
}
```

Figure 40: Screenshot of the code handling correct letter guesses.

It then flips it, changing the color to green on both the game screen, as well as the keyboard. The next part of the algorithm checks if the letter is part of the solution, which will mark it as yellow. However, it is more complicated than that. Because while the solutions never contain a double instance of a letter, the user may guess repetitive lettering, to figure out where the letter will be. To handle this, we created an algorithm that will only make the letter that is close to the location of the right placement yellow.

```

} else if(!_solution.letters.contains(currentWordLetter)) {
    letters.add(currentWordLetter);

    // Check if guess has placed this correctly, before flipping it to yellow
    bool guessedRightAnotherTime = false;
    int latestIndex = 0;
    for (int j = 0; j < _currentWord!.letters.length; j++) {
        if (_currentWord!.letters[j].val == _solution.letters[j].val &&
            _currentWord!.letters[j].val == currentWordLetter.val) {
            guessedRightAnotherTime = true;
        }
        if (_currentWord!.letters[j].val == currentWordLetter.val) {
            latestIndex = j;
        }
    }

    // If its guessed correctly later, no need to mark it
    // However if it is guessed later, flip the latest "closest" to the right guess
    if (!guessedRightAnotherTime && latestIndex == i) {
        _currentWord!.letters[i] =
            currentWordLetter.copyWith(status: LetterStatus.inWord);
    } else {
        _currentWord!.letters[i] =
            currentWordLetter.copyWith(status: LetterStatus.notInWord);
    }
}
}

```

Figure 41: Screenshot of the code handling yellow cases and wrong letters.

The code loops through both the current list of letters that the user guessed, and the solution list, it then makes sure that the user has not guessed this letter before, and that it is not a green letter. For instance, if the word is "viten" and the user has guessed "eplet", then the first "e" should not be marked as yellow, because the user guessed correctly with the last "e". Then it checks the user-made list if it matched the current letter, and if it does, then it marks the index as the latest index.

After this, if it has not been guessed right at another time, and the index matches the last seen instance of this letter, then it flips the letter to yellow.

The code under this checks if the letter is in the word, and if it isn't, it flips it to dark blue.

7.5.2 Tower of Hanoi

Now this game is the most complex out of them all. This game is based on three rods where you move bricks from the leftmost rod to the rightmost one. To pull this off, the entire playable area needs to consist of a "grid" where you can place down game pieces. The game board is also stored in a list of strings, keeping track of the placement of the game pieces. The grid consists of rods, which are placed at the bottom of a stack. On top of this rod is a game object. This game object will only be shown when there is a game piece resting on the rod. To know that, the game checks the list containing the game board and activates the game pieces according to the list. The pieces store information like color, image, and last location.

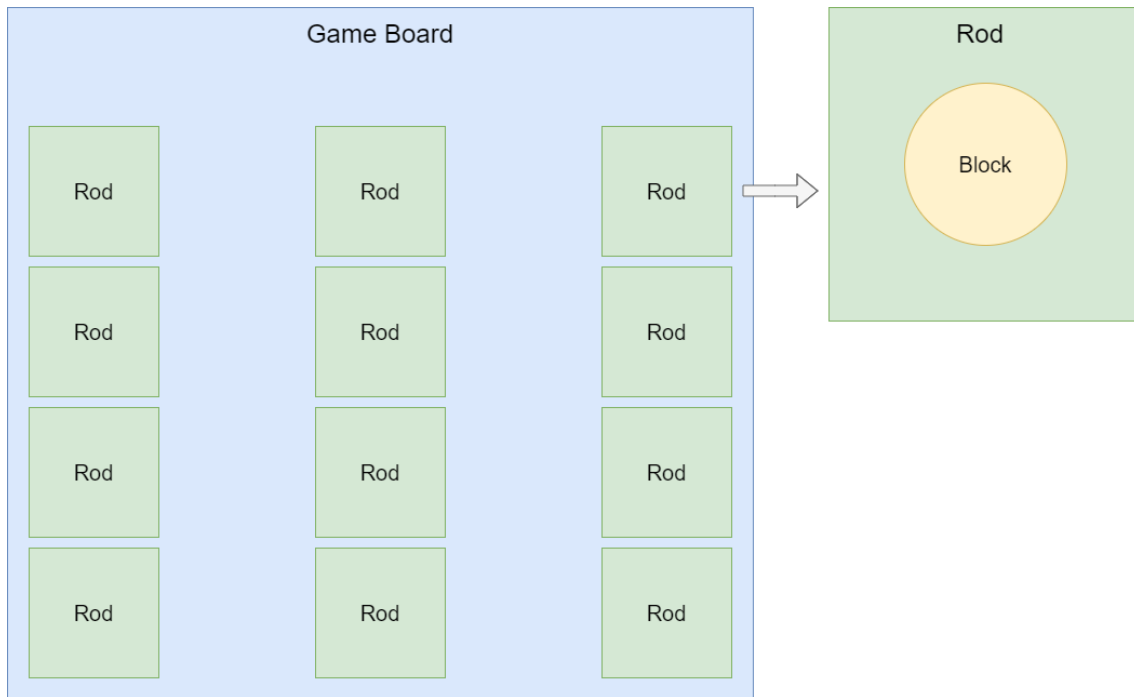


Figure 42: Illustration of how the game board is set up in the Tower of Hanoi.

The different objects have different conditions. The rods are a container which is called a "dragTarget". This is a target where you can drop objects. Before an object is dropped there are some rules to allow object to be dropped:

```

onWillAccept: (gameObject) {
  return checkLegalMove(
    _gameObjects[lastPiece], gameBoard, 0, lastIndex);
},
onAccept: (gameObject) {
  setState(() {
    //update new place
    gameBoard[_gameObjects[lastPiece].lastIndex] =
      _gameObjects[lastPiece].color;
    //remove the piece from where it was
    gameBoard[lastIndex] = 'empty';
  });
},

```

Figure 43: Code of DragTarget for Tower of Hanoi.

The dragTarget will only accept a piece if it can be considered a legal move. To check this the function "checkLegalMove" needs to return true. More on this function later. After the piece is allowed to be placed, OnAccept will run, which updates the gameboard to remove the game piece where it came from and activate it on top of the current dragTarget.

```

Draggable<GameObject>(  

    data: _gameObjects[getId(gameBoard, 0)],  

    feedback: Image.asset(  

        _gameObjects[getId(gameBoard, 0)]  

        .imageId), // Image.asset  

    childWhenDragging: Container(),  

    onDragStarted: () {  

        lastIndex = 0;  

        lastPiece = getId(gameBoard, 0);  

    },  

    onDragCompleted: () {  

        setState(() {  

            moves++;  

            win = checkWinCondition(gameBoard);  

        });  

    },  

    child: Image.asset(  

        _gameObjects[getId(gameBoard, 0)]  

        .imageId), // Image.asset  

), // Draggable

```

Figure 44: Code of Draggable with index 0 for Tower of Hanoi.

The game piece has rules similar to this. It is called a Draggable in the code, and it holds the information specified earlier. It displays as an image of the game piece. When the game piece is being picked up by the user, `onDragStarted` gets called, which sets the `lastIndex` to be the index of the current rod, which in the example is 0, it also gets the `lastPiece` from this location, to know which piece is being moved. These variables are stored locally so that the game knows where the piece came from, and which piece it is. When the user drops the draggable onto the `dragTarget`, and if the `dragTarget` allows the piece to be dropped, then `onDragCompleted` gets run, which adds to the number of moves used so far, and also checks to see if the game has been won. The win condition is easily checked by comparing the list of the game board to a list that contains the solution to the game board, and if they match, the user has won the game.


```

28
29 /// Checks if the location the player is trying to place a [gameObject] on has anything under it on the [gameBoard]
30 /// Returns true if it can be placed
31 bool checkLegalMove(GameObject gameObject, List<String> gameBoard, int index, int lastIndex) {
32     // If the piece can be moved
33     if (!checkIfPieceCanBeMoved(gameObject, gameBoard, index)) {
34         return false;
35     }
36
37     // Edge-case where its the bottom of the rod
38     if (index == 3 || index == 7 || index == 11) {
39         if (gameBoard[index] == 'empty') {
40             gameObject.LastIndex = index;
41             return true;
42         }
43     }
44
45     // If the place you are trying to put the piece on is empty
46     if (gameBoard[index] == 'empty') {
47         // Check whats underneath
48         int indexOfNextAvailableSpace = indexesToPieceUnderneath(gameBoard, index);
49         // Check which piece is there, to determine if the current piece can be placed there
50         if (canPieceBePlaced(gameObject, gameBoard, indexOfNextAvailableSpace)){
51             gameObject.LastIndex = indexOfNextAvailableSpace;
52             return true;
53         }
54     }
55     return false;
56 }
57

```

Figure 45: Code of function "checkLegalMove" Tower of Hanoi.

Now back to the function checkLegalMove. The first thing happening here is that it checks if the piece can be moved at all. This is done with the function "checkIfPieceCanBeMoved". This function takes the game board, the current game piece, and the relevant index. It then makes sure that the game piece is moved to a new index because if it is dropped onto the place it came from, no change is needed to make. Now that we know the location we are moving to is different from the one we originated from, we have to check if the piece is allowed to move from its current location. We check the game board if there is another game piece resting on top of the relevant game piece, and if that is not the case, the game piece is allowed to move.

After we know that we are allowed to move, we now check to see if where we want to move is allowed. There are some edge cases, as you can always play a game piece at the bottom of a rod. This is illustrated in the code from lines 37 to 43. This also ends the function, and updates the game piece.

Now if it is not an edge case, we need to run some more checks, first on line 46 we check if the relevant placement is empty, meaning there is currently nothing there. If it's empty we continue on. Now to simulate gravity and the feel of the pieces dropping down, it was important that users could simply drag a game piece over to a rod, and it would "drop" to the first available spot. To pull this off the function "indexOfNextAvailableSpace" counts the number of dragTargets under the current index. It returns the number of spaces that the piece will drop. Now finally, we check if the piece can be placed with the function "canPieceBePlaced". This function checks what's "under" the location where you want

to drop the piece. As one of the rules is that you can only place a smaller piece on top of a bigger one, and not the other way around, this function checks if the piece under is bigger than the active game piece. It returns true if the game piece can be placed. And after all these checks the game piece gets its last index updated, and the function returns true.

7.5.3 Viten Kode

Added on by Vitensenteret after the other two games were finished was the game called Viten Kode. This game was supposed to be a way for younger people to learn coding without having any experience with it. As a way to create a game Vitensenteret suggested a game where you place blocks and get different outcomes out of this, however, due to time limitations we elected to create a game where you put the blocks in the right order to produce the expected result.

The game has a list in which you can reorder the blocks to be in the right order. The code for this is relatively simpler, as it is by far the smallest game implemented. First, since this game is connected to the database, it checks if the information is downloaded from there, and if it isn't, it will fetch it. Once it has fetched it, it will display the game. The main aspect of the game is the reorderable list:

```
// Start game when data has been fetched
if(fetchedData) {
  return Scaffold(
    body: SafeArea(
      child: Column(
        children: <Widget>[
          // Display a task
          Container(
            padding: const EdgeInsets.all(20),
            margin: const EdgeInsets.all(10),
            height: 100,
            color: blaa,
            child: Center(child: Text(getTask(taskId))),
          ), // Container
          // Reorderable list containing options (main part of game)
          Flexible(
            child: ListView(
              shrinkWrap: true,
              children: [
                Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: <Widget>[
                    SizedBox(
                      height: 300,
                      child: _reorder), // SizedBox
```

Figure 46: Screenshot of the main widget for Viten Kode

At the bottom of this code, the child calls reorder which is a widget containing the logic behind the moving boxes. This simply deletes the old instance of the box, and inserts it where the user dropped it.

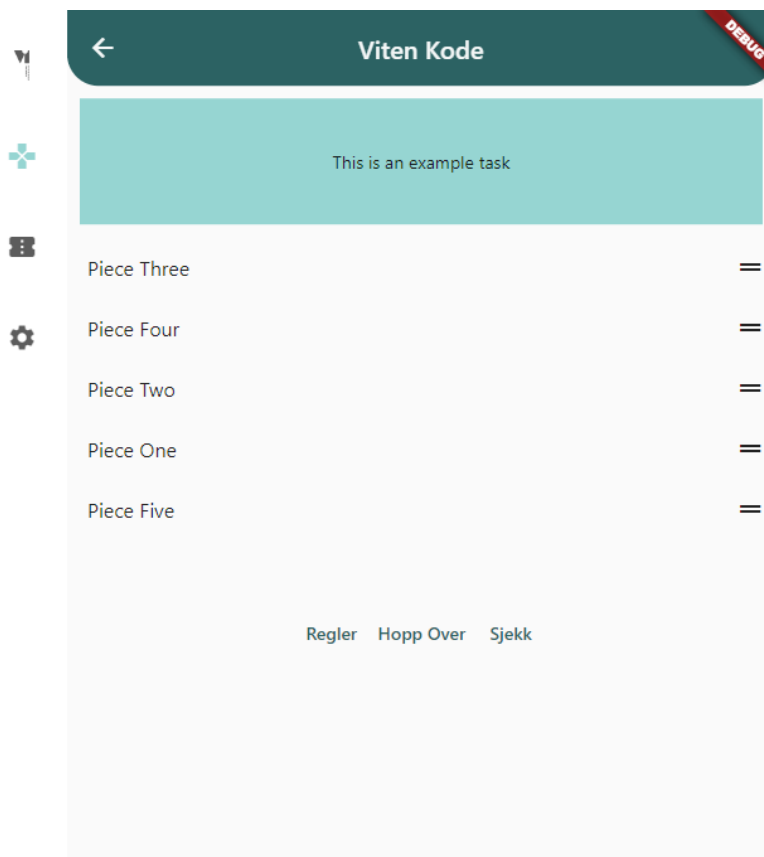


Figure 47: The game Viten Kode

The rest of the logic happens when the user presses the available buttons. "Regler" displays the rules, "Hopp Over" skips the question and pulls a new random one from the database, and "Sjekk" compares the current list that the user has worked on, to the solution list that it fetched from the database. If they match, then a victory message is displayed to the screen.

8 Testing

8.1 Unit Testing

As mentioned earlier we created unit tests for the functionality of the games. These tests were checking any function that does not revolve around I/O, that is connected to the games. These tests all succeeded. These tests checked functions that calculated outputs, that their outputs were correct, and that the format it get and returns are as expected.

```
Running with sound null safety
Debug service listening on ws://127.0.0.1:59888/Pb9ZL7ga9U4-/ws
00:00 +0: Programming Game (Viten Kode) compare two lists, should check solution with current guess from the user, they match
00:00 +1: Programming Game (Viten Kode) compare two lists, should check solution with current guess from the user, they dont match
00:00 +2: Programming Game (Viten Kode) compare two lists, should check solution with current guess from the user, user wrote random values
00:00 +3: Tower of Hanoi Game Function checks rules to see if the piece is allowed to move, move to unoccupied space
00:00 +4: Tower of Hanoi Game Function checks rules to see if the piece is allowed to move, move to space where a piece already is
00:00 +5: Tower of Hanoi Game Function checks rules to see if the piece is allowed to move, try to put bigger piece ontop of smaller piece
00:00 +6: Tower of Hanoi Game Function checks if the location the player is trying to place a piece on has anything under it, if it can be placed, it will return true, move to unoccupied space
00:00 +7: Tower of Hanoi Game Function checks if the location the player is trying to place a piece on has anything under it, if it can be placed, it will return true, move ontop of bigger piece
00:00 +8: Tower of Hanoi Game Function checks if the location the player is trying to place a piece on has anything under it, if it can be placed, it will return true, move ontop of smaller piece
00:00 +9: Tower of Hanoi Game Function checks if the board meets the win condition, sending winning board
00:00 +10: Tower of Hanoi Game Function checks if the board meets the win condition, sending wrong board
00:00 +11: Tower of Hanoi Game Function checks if the board meets the win condition, sending jumbled short board
00:00 +12: Tower of Hanoi Game Function that returns true if the current index being checked is occupied by a gamepiece, checking empty slot
00:00 +13: Tower of Hanoi Game Function that returns true if the current index being checked is occupied by a gamepiece, checking non-empty slot
00:00 +14: Tower of Hanoi Game Function that returns true if the current index being checked is occupied by a gamepiece, checking outside array
00:00 +15: Tower of Hanoi Game Checks the given index if a the gamepiece can be placed there based on the color of the piece, red can always be placed
00:00 +16: Tower of Hanoi Game Checks the given index if a the gamepiece can be placed there based on the color of the piece, red placed ontop of yellow
00:00 +17: Tower of Hanoi Game Checks the given index if a the gamepiece can be placed there based on the color of the piece, blue on top of red is not legal
00:00 +18: Tower of Hanoi Game Function counts the number of empty places under the chosen index, returning next available index, counting from 4, should return 7
00:00 +19: Tower of Hanoi Game Function counts the number of empty places under the chosen index, returning next available index, counting from 4, should return 5
00:00 +20: Tower of Hanoi Game Function counts the number of empty places under the chosen index, returning next available index, counting from 11, should return 11
00:00 +21: Tower of Hanoi Game Function returns the id of the gamepiece at the requested tile, requesting at 0, which has red
00:00 +22: Tower of Hanoi Game Function returns the id of the gamepiece at the requested tile, requesting at 2, which has blue
00:00 +23: Tower of Hanoi Game Function returns the id of the gamepiece at the requested tile, requesting at 7, which has empty
00:00 +24: Tower of Hanoi Game Function takes the tileId and returns which image should be displayed, getting top of a rod
00:00 +25: Tower of Hanoi Game Function takes the tileId and returns which image should be displayed, getting middle of a rod
00:00 +26: Tower of Hanoi Game Function takes the tileId and returns which image should be displayed, getting outside array
00:00 +27: Tower of Hanoi Game Function displays victory text based on number of moves used, sending best result
00:00 +28: Tower of Hanoi Game Function displays victory text based on number of moves used, sending decent result
00:00 +29: Tower of Hanoi Game Function displays victory text based on number of moves used, sending high number
00:00 +30: Word Game (Viten Ord) Function returns the index where the instance of the letter was located at, checking letter in word
00:00 +31: Word Game (Viten Ord) Function returns the index where the instance of the letter was located at, checking letter with multiple instances in word
00:00 +32: Word Game (Viten Ord) Function returns the index where the instance of the letter was located at, checking something that isnt in word
00:00 +33: Word Game (Viten Ord) Function counts the number of instances of an item in the list, and if the number exceeds 1, then returns true. Check word with multiple instances
00:00 +34: Word Game (Viten Ord) Function counts the number of instances of an item in the list, and if the number exceeds 1, then returns true. Check word without multiple instances
00:00 +35: Word Game (Viten Ord) Function counts the number of instances of an item in the list, and if the number exceeds 1, then returns true. Check word without letter
00:00 +36: All tests passed!
```

Figure 48: Unit Tests, performed on all games.

8.2 User Testing

In addition to the unit tests we also performed user tests. The full documentation for this can be found in [J]. To conduct these tests we asked different people with different backgrounds to perform the tests. Overall we have five testers, and while two have a similar profession, we made sure to also ask those well versed with IT, and those not so much.

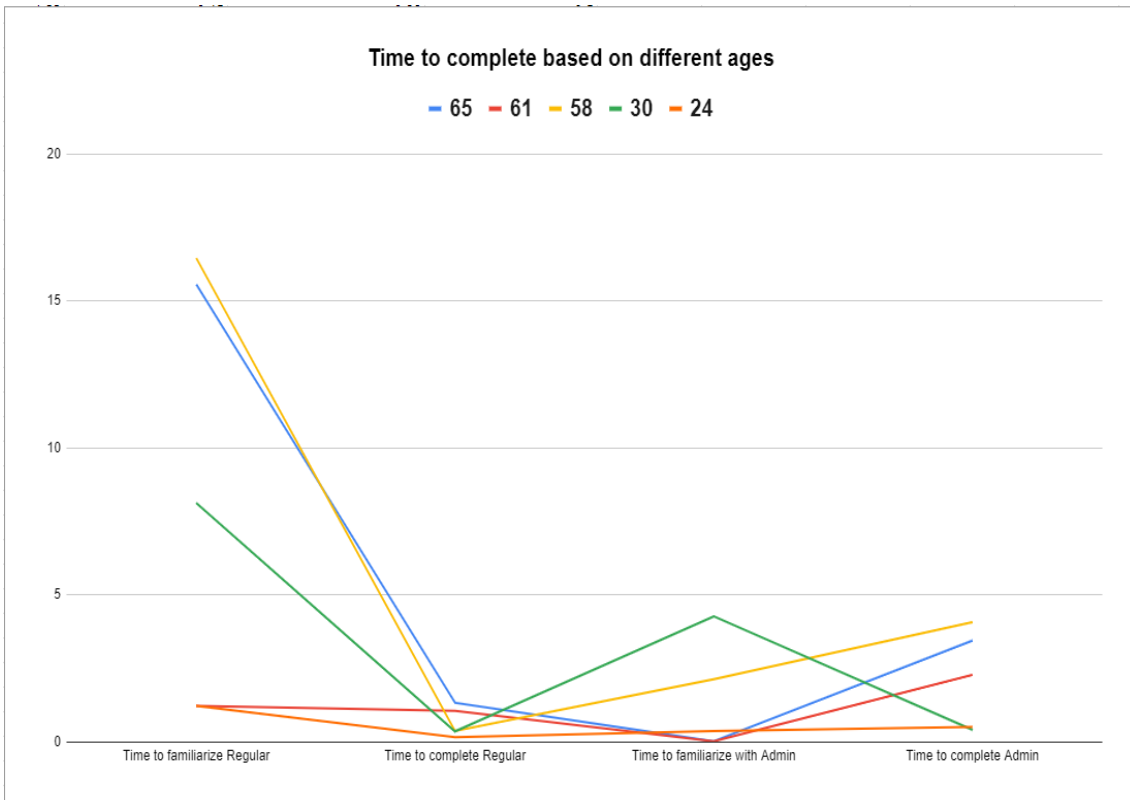


Figure 49: User Test Results.

This chart shows the time in minutes it took the users to familiarize themselves with the app and the time it took for them to solve the tasks given during the testing. As can be seen, there are some pretty high numbers in the first category: Time to familiarize Regular. This is because those three, yellow, blue, and green all found the games included in the app and spent time playing these games. We can also notice that the younger testers did incredibly well on the tasks, both using under a minute to perform all the tasks. The older generation was slower, albeit everyone managed to solve the tasks. In addition to making them perform some tasks, we asked them some questions:

- Was the application easy to navigate? Was there at any point a time where you did not know where to go?
- Did you find something in the application that you did not expect to find? Did you notice or try any of the games?
- Do you feel the admin pages flow naturally and have any functionality that may be needed?
- Would you download this app if you were a frequent user of Vitensenteret? And if you would only download it for the ticket system, do you feel that the games included in the app makes it more likely for you to keep the app?

-
- Anything else you would like to add?

Generally, everyone found that the app was easy to learn and intuitive to use. We got some constructive feedback towards the admin pages, specifically the admin blog page, as this one requires some more polish to be able to use more intuitively. The most notable feedback in this regard was to make it easier to access the admin blog page, with a bigger button or something akin to that. In addition, some felt that the return button should be in the top left corner, similar to the games.

In regards to unexpected things, the users all mentioned the games. They were all pleasantly surprised to find them, some even to the degree that they forgot they were performing a user test, and that affected their time to familiarize themselves with the app as they were simply playing games and enjoying themselves. Everyone liked the games and they felt it was a nice addition, some mentioned that simply having ad-free games on the app could mean more people download the app. They did however remark that it was difficult to understand how to solve them, so to make this better we should add more rules and information in the info box that they can bring up when playing.

Most of the testers would download the app, most seemed excited about having it, and feel it adds to the experience at Vitensenteret. However, the youngest tester was incredibly skeptical, stating that they don't download apps to their phone that they don't need. As all the functions of the app can be accessed at the physical location, save the games, this app is not a "forced" download to gain access to Vitensenteret, and therefore this person would not download it.

Most of our testers really liked the app and stated that they think it will benefit Vitensenteret greatly to have such an app. It allows the user to have their tickets, maybe even for the entire family, and with the news it's easy to see if something is happening at Vitensenteret.

9 Code Quality

9.1 Database Best Practices

When designing the database it was important to follow best practices as laid out by Firebase [19]. These best practices include keeping a flat data structure and designing the database to scale well. The first practice we hold by dividing our data in categories, and ensuring that we don't go deeper than two levels. The second practice is visible in tickets and users, where the name on the ticket is the name of a user, and each user has a list of tickets. This means that whenever this data is changed, it has to be changed in two places, but this is how Firebase encourages the data to be stored, and it simplifies data fetching.

Additionally when we fetch from the database, or push to the database we try to keep the number of calls to a minimum. This is the lower the chances of something going wrong.

9.2 Commenting Standards

As we are new to flutter we decided to take inspiration from the official dart documentation standards [21]. When it comes to commenting functions, dart suggests something quite different than most languages which we are familiar with. To our recollection, all languages we have used until now have a standard format for commenting where you first write a summary of what a function does, then orderly list all input variables and their meaning/purpose and lastly the return values. Dart, however, suggests to write all documentation as sentences where you refer to in-scope identifiers in square brackets (Figure: 50).

For general comments in the text we resorted to comment as much as we could for each visible part of the application, meaning every visible unique item should be commented to make it easier to see a connection between items in the app and what you are looking at in the code. It should be specified that by item we are not referring to a widget but rather a group of widgets that together perform one task i.e. a text button widget and it's text widget. This means individual widgets are not commented but rather groups and their purpose.

```
13  /// Validates that a string [num] could be a potential phone-number
14  bool validatePhone(String num) {
15    if (!num.contains(RegExp(r"^[0-9+]*$"))) {
16      return false;
17    }
18    return true;
19  }
```

Figure 50: Example of a function comment

9.3 Naming Conventions

We followed the naming conventions that Dart suggests on variable names, class names, function names and file names [22]. Dart suggests using upper camelcase for classes, this means that each word in the name starts with an upper case letter, no spaces, underscores, or dashes. For filenames it is suggested to use snakecase, which is full lowercase with underscores for spaces. For other names it is suggested to use lower camelcase, all words in the name starts with a capital letter, except the first word, no spaces, underscores, or dashes.

9.4 Using Flutter's Built-in Linter to Minimize Redundant Code

When coding the built in linter would alert us about redundant code, places where something should be a constant, and other smaller and larger issues. We made ample use of this feature, and the code has come out cleaner because of it.

We also took a round towards the end of the project where we went over all the code, using the linter actively. The purpose of this round was to make the code faster, cleaner and better for future review and development.

10 Deployment

While the app won't be deployed to the market, it is still in a state where it's very close to being able to be uploaded. What remains are mostly the legal terms, to be able to upload the application to the Internet, as well as some tweaks to the design. The app is fully functional despite its security risks, but will simply be given as it is to Vitensenteret, and they will be responsible for any further development or upload to any app store they may see fit.

The code is ready to be uploaded and is configured to work on all platforms, most computers with a connection to the Internet can run it through their chosen browser, and it runs on most phones as well.

10.1 App Store for iPhone

To release an app to App Store, a device capable of running Xcode is needed, and a developer account on the App Store. To see more about the process for releasing an app on App Store, see this site: <https://docs.flutter.dev/deployment/ios> [23].

10.2 Google Play Store for Android

To deploy on Google Play Store a developer account is needed, the process of getting a developer account also costs money. The app has to be signed before it can be uploaded. For more information on how to deploy an app to Google Play Store, see this site: <https://docs.flutter.dev/deployment/android> [24].

10.3 Source Code

The source code is available online for any who has access to the link to the repository. The code is located in a GitLab repo with a proper structure. Navigating the file tree to look into relevant files, and following the README.md to be able to download and run it locally. The source of this repository is <https://gitlab.com/hirkasa/billettapp-for-vitensenteret/> [25].

10.4 Terms and Conditions

On all platforms where the app will be downloadable, there will be a section including the terms and conditions. The user has to agree to these in order to download the app.

These terms will encompass the GDPR rules mentioned in "3.1.5 Storing Personal Data in Accordance with GDPR."

11 Further Development

11.1 Deployment

Unfortunately, we were not able to upload this app anywhere, except for leaving the code online for those with the link. This was a difficult decision because everyone on the group wanted to produce a finished result, however, with the time constraints and final polish not being completed we elected to leave the deployment for Vitensenteret, which means they may decide to upload the app as it is or hire someone to work out the last few things, which for the most part are not integral to the app itself.

11.2 Further Polish of Interface

As our user tests uncovered, there is still some polish left to do on the admin pages. This involves creating a better way to access the admin page for the blog and making the design easier to use, perhaps drawing some more inspiration from the admin ticket page by displaying all blogs, or perhaps not having a button leading to the admin page, but having a similar solution to the ticket page, where if you navigate to that tab you are simply in the admin page.

In addition to this, we would make more feedback for the user, for instance, one tester commented that if you tried to log in with the wrong credentials, there was no message explaining why it didn't work. And when you upload or work with the blogs, there is no message that the work was successful, save for the page going back to default.

11.3 Database

The most important thing that has to be done next is transferring the real data that Vitensenteret has in their database, into the same format as our database use. Then their database can be set as the database of the app.

There are not many things that can be improved in the Firebase Realtime Database as of today, not because it is ideal, but because it is not meant for complicated data storage and transfer. For future development, it is recommended to switch to Firestore, as this database has more functionality and more customization.

Currently, all blog posts are saved as a single string in Realtime Database, it would be better if they could be stored as text files in Firebase Storage, we did however not find the time to do that in this project.

Currently a lot of functionality between the database and app is dependent on continuous

connection, as several operations require more than one call to the database. Using Firebase Functions to do these operations would improve the security of the app significantly. Then a single call to the server, with an independent answer would be all that is needed.

With Firebase Authentication user data is very safe. However, there is currently no way to check if someone tries to make a user with a fake email. Implementing email verification is therefore highly recommended, as well as a few other functionalities that Firebase allows for, like password change verification and warnings.

11.4 Games

11.4.1 Game Engines

When we started out creating the games, we did not imagine they would be as big and complicated as they turned out to be. Especially towers of Hanoi and Viten Ord which both have complicated structures and algorithms. As a way to optimize and further improve these games, one could instead of using vanilla Flutter, use a Flutter-based game engine called Flame[26]. This one would create the engine for you as well as make handling movable objects much easier. While Viten Ord is a game that flows neatly, the Tower of Hanoi would greatly benefit from using this system.

It may also be possible to use other game engines like Unity or Unreal, or perhaps even Godot to create a better framework for the games. This would likely enable the games to run smoother and have higher complexity, but it would also make the app much larger to install.

11.4.2 Further development of "Viten Kode"

As seen in the implementation, the game Viten Kode is much smaller than the other two implemented games. This was mostly a time limitation, and therefore given more time, this game could be revamped to be more in line with Vitensenteret's original idea. A way to solve this could be to look at some of the code of the other game, Tower of Hanoi, as this utilizes movable pieces. One could use these pieces and assign them functionality, and create a game where players could organize the blocks to create unique functions. However, the scope of creating a game like this, if it is done thoroughly would be a massive undertaking.

Say there are 5 blocks that you can put in any order, and all of them work together. Say you can put any repeating numbers as well, That allows for a massive amount of possible ways the code can work. Creating this game could be in its entirety a whole bachelor thesis.

A simple way to easily improve the current implementation could be to just add some more tasks into the database. Originally the game was supposed to pull tasks from our storage in Firebase, however, due to some misunderstandings between the group and Vitensenteret, we did not receive any tasks to put in there, and thus the only playable task is a placeholder.

11.4.3 Adding a Scoreboard and Daily Rewards for Logged-in Users

Another way to create more interest in the games, and perhaps even a community could be to create a user board with a score amongst the best players. Perhaps even sponsored with prizes from Vitensenteret to give an incentive. There could be a daily word inside Viten Ord, or perhaps keeping a daily streak increases the number of points you get.

This could be linked to games or attractions at Vitensenteret. There could be QR codes you scan to get more information about a display, and doing so rewards some points. There are many ways to further the development of the games and make them even more enticing.

11.5 Notification System

One of the things Vitensenteret originally requested was a notification system. That way they could create news about events or discounted tickets and so on. We have laid some of the framework down for this, as when you create a blog post, you can select for it to create a notification. To make this system we would have to create webhooks as well as add to the existing database.

To create a notification system we would have had to prep each version of the application, and the process is different on iPhone, Android, and Web apps. Furthermore, we would have needed a plugin called FCM, this plugin is what allows us to send messages to clients. Firebase has detailed instructions on how to set this up [27]. Though the process is simple enough, by the time we had blog functionality we were already at the end of the project period and decided there was not sufficient time to develop it.

11.6 Other Additions

When we had our first meeting with Vitensenteret, they gave us a list of anything they could think of, that could be added into the app. Some of it was not as relevant as others. The full list can be found in Appendix [I] source. However, we picked out some points that the group thinks have potential.

- QR-Code to scan and get more information about the displays.
- A game which connects to the astronomy room which displays stars with Augmented Reality.
- A way to find information about food and parking. Perhaps even a way to order in the app.

In addition to this, we should also factor in the user test, and what can be improved around the application from those. These are mostly comments surrounding the admin pages, where they occasionally struggled the first time to locate some buttons. While this may be circumvented by having the admins read the user manual, the design should still be improved. In addition to this, there should be more rules and information on how to play the games.

12 Discussion

12.1 Implementation and Follow Through

12.1.1 Worked Hours

The following figure is a chart of all the hours tracked in relation to this bachelor thesis. The figure is color coded with general topics, however, also links directly to issues or topics that that individual worked on. We had some categories with the general workload that day, and we color-coded accordingly, where we used the color of the activity we spent the majority of the time doing that day. These primary topics were as follows:








Color:	Explanation:
	Research
	Report Writing/Documentation
	Testing
	Coding
	Meetings
	Planning
	Sick or Unavailable

Figure 51: Time chart legend

Here are the worked hours for all group members. The name is top left, and each day has a number of hours followed by more specifics on what was done that day. Each member also has listed the hours worked on each specific topic.

Malin Foss								
Week:	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total:
2			2 (Seminar)		3 1.5 (Project plan)			6.5
3	4 (Project plan)	6.5 (Research and meeting)	4.5 (Meeting, report)		4 (meeting with VI)		1	17
4	1.5 (Project plan)	5 (Report writing and meeting)	6 (Flutter, writing req.spec)	4 (Flutter)	4 (Flutter, GitLab CI, prototype)			20.5
5	1 (req.spec)	4 (GitLab CI, flutter)	4 (Flutter, 2FA, gitlab CI)	4 (Group and VI)				13
6	1 (Firebase)	1 (Sprint meetings, firebase)	1.5 (firebase)	5 (issues 6 and 10, meeting)	3 (issue 10)			11.5
7	6 (Issue 10)	3 (Sprint and mentor meeting)	3 (issue 18)		2 (for issue 18)			14
8	6 (Issues 18 and 19)	5 (Meetings and Issue 21)	1 (Issue 21)	6 (issues 20, 21, 22)			1.5 (Issue 21)	19.5
9	2.5 (issue 21, meeting)	5 (Meetings, lecture, etc.)	2 (issue 21)	7 (issue 21, meeting with VI)				16.5
10	4 (Issues 21, and 22)	6 (21 and 22)	6 (Issue 20 and 21)	6 (Meeting with VI, testing, issue 20)	1 (Issue 25)			23
11	3 (Duell)	6 (Issue 20)	6 (Issue 20)	6 (Issue 20)	6 (Firebase)			27
12	2 (Issue 20)	4 (Issue 20)	6 (Issue 26)	4 (Issue 26, research database rules)	3 (Issue 28)			19
13	3 (Issue 29)	6 (Issue 30)	5 (Issue 29)	5.5 (issue 30)	4 (Issue 29)			23.5
14								0
15		6 (Issue 29)	6 (issue 30)	6 (issue 30)	4 (issue 30)			22
16		5 (Planning, Issue 29)		7 (deployment, issue 35)			3 (issue 30)	15
17	3 (issue 30)	4 (general fixes)	3 (issue 29)				1.5 (issue 29)	11.5
18	2 (issue 29)	5 (General fixes)	5 (General fixes)	5 (Vilensenteret, gruppetete)				17
19	4 (final report)	3 (final report)	4 (final report)	5 (final report, issue 41)	6 (final report)			22
20	4 (issue 46)	4 (final report)	6 (final report)	4.5 (final report, user manual)	5.5 (final report)	6 (final report)	5 (final report)	35
								333.5

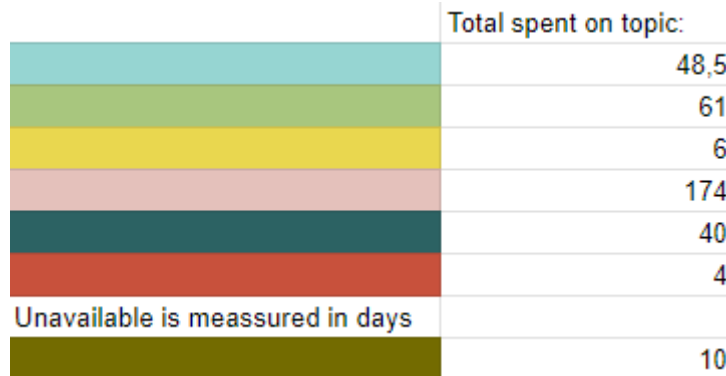


Figure 52: Time Chart Malin Foss

Malin was, in general, responsible for the flow of the project, being the project manager. She spent the majority of the time coding, but quite a bit on research as well to help find the technology. Some time was also spent on communications with the project owner and mentor. The majority of the time coding was spent on the authentications pages, the settings page, and the admin ticket page, and especially the admin ticket page ended up taking a lot of time and effort. While writing this thesis she also had two extra courses and a TA-job, both of which took considerably more time than anticipated. The unavailable days are for the Easter holiday.

Philip Morud								
Week:	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total:
2			2 (Seminar)	3 (Planning)				5
3		6 (Research and meeting)	5 (Research and meeting)	2 (Research)	1 (meeting with Employer)			13
4		3 (Research and meeting)	5 (Research)	5 (Flutter)	3 (Flutter)	2 (Flutter)		18
5	3 (Flutter)	5 (Research)	5 (Git and flutter setup)	4 (Group and VI)	3 (Flutter)			20
6		2 (Sprint)		3 (Meeting and research)	2 (Flutter)	3 (Issue 9)		10
7	2 (Issue 9)	5 (Vipps solutions)	4 (Vipps, Blog and Status)	3 (Issue 17)	3 (Issue 17)			17
8	3 (Issue 17)	5 (Issue 17)	3 (Vipps)	4 (Issue 17)	5 (Issue 17)			20
9	3 (Meeting about Vipps)	4 (Vipps research and meeting)		5 (Issue 17 and meeting)	4 (Issue 17)			16
10		4 (Issue 17)	4 (Issue 17)	6 (Meeting with VI, Error fixing)	3 (Issue 23)	3 (Issue 23)	2 (Issue 23)	22
11	2 (Issue 23)	2 (Issue 23)	4 (Issue 23)	7 (Issue 20)				15
12		4 (Issue 20)	6 (Issue 20)	6 (Issue 20)	2 (Vipps)		2 (Vipps)	20
13		5 (Issue 31 and bugfixing)	5 (Issue 23)	5 (Issue 23)				15
14								0
15		3 (Stripe payment system)	6 (Stripe payment)	5 (Stripe payment)				14
16		6 (Stripe)	4 (Stripe)	5 (Stripe)	2 (Issue 31)			17
17	4 (Issue 31)	6 (Issue 31)						10
18	4 (Issue 31)	7 (Issue 31)	11 (Issue 31)	5 (Improved blog page)	2 (Issue 31)	2 (Issue 31)	2 (Issue 31)	33
19	5 (Issue 31)	4 (Issue 31)	6 (Issue 31)	4 (Issue 31)	6 (Issue 31, 34 and 45)			25
20	5 (Issue 46)	4 (Issue 46)	4 (Final Report)	6 (Final Report)	6 (Final Report)	10 (Final Report)	5 (Final Report)	42
								332

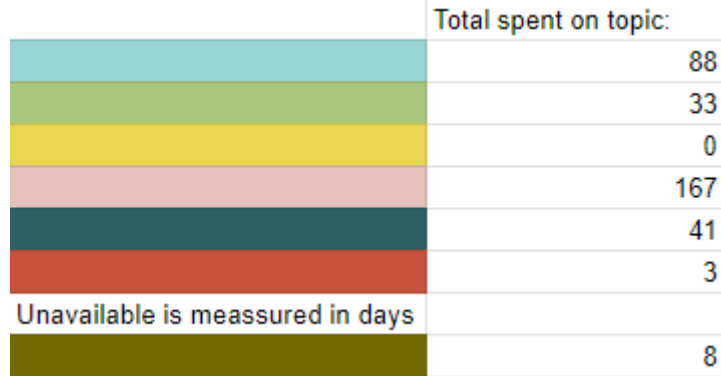


Figure 53: Time Chart Philip Morud

Philip was responsible for research. He spent around 25% of the time researching valid technology, and he was the one who did all the research around the payment systems which came out to be quite a bit of work due to the back and forth with Vitensenteret. He mostly focused on research and coding, but always helped out with any relevant documentation. While writing the thesis, he also had three other subjects; Advanced Programming, Cloud Technologies, and IØ2000. Philip was the one with the least amount of unavailable days despite this.

Susanne Skjold Edvardsen								
Week:	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total:
2				3 Planning				5
3	3 (Project plan)	5 (Research and meeting)	3 (Project plan)				3(Design)	14
4		5 (Report writing and meeting)	5 (Research)	4 (Flutter)	5 (Flutter)			19
5		4 (req spec)	4 (Flutter, project plan)	4 (Group and VI)				12
6		3 (Issues: 7,11)	4 (Issue: 8)	4 (Issues: 2, 11) Meeting with VI				11
7	5 (Issue: 8)	4 (Issues: 13,16)	4 (Issues: 13,16)		4 (Issue: 14)		6 (Issues: 12, 15)	23
8	3 (Issues 12,14,15)	4 (Issue: 13) Meeting with Mentor		2 (Issue: 13)		2 (Issue: 13)		11
9	(Group)			2 (Issue: 14)	3 (Issue: 14)	3 (Issue: 14)	4 (Issue: 14)	13
10	2 (Issue: 14)	4 (Issue: 14) Meeting with Mentor	2 (Issue: 14)	3 (Issue: 14, 24)	4 (Issue: 14)			15
11	5 (Issue: 14)	4 (Issue: 14)	2 (Game Design)	4 (Issue: 14)	4 (Issue: 14)			19
12	5 (Issue: 14)	4 (Issue: 14) Meeting with Mentor	4 (Issue: 14)	4 (Issue: 14)	4 (Issue: 14)			21
13	5 (Issue: 14)	4 (Issue: 14, 24) Meeting with Mentor		4 (Issue: 24) Meeting with VI	4 (Issue: 24)			17
14								0
15							5 (Issue: 32)	5
16	5 (Issue: 32)	5 (Issue: 32) Meeting with Mentor	5 (Issue: 32)	5 (Issue: 32, 38)	5 (Issue: 38, 39)			25
17	5 (Issue: 39)	4 (Issue: 39) Meeting with Mentor	9 (Issue: 39)	8 (Meeting with VI Writing documentation)		4 (Issue: 39)		25
18		4 (Issue: 36, 39) Meeting with Mentor		8 (Issue: 40)	8 (Final Report)	5 (Final Report)	5 (Final Report)	30
19	5 (Final Report)		8 (Final Report)	5 (Final Report)	8 (Final Report)			26
20	5 (Final Report)	12 (Final Report)	5 (Final Report)	3 (Final Report)	5 (Final Report)		5 (Final Report)	35
								326

Total spent on topic:	
	29
	128
	14
	127
	23
	5
Unavailable is measured in days	
	21

Figure 54: Time Chart Susanne Skjold Edvardsen

Susanne was responsible for the documentation. This is reflected by the time being spent on documentation totaling around 40% of the total hours for this project. A fraction more than what was spent coding. She created the games as well as the blog and admin blog. She also took the main responsibility for the testing. She had two part-time jobs while undertaking her bachelor, in addition, had a close family death during the project. Despite this was able to dedicate a lot of time to this project.

12.1.2 Following the Gantt Chart

It was the original Gantt chart that we created at the start of the project. This was divided into four sections, planning, production, testing, and report. Additionally, we marked the dates where our mentor requested we send in statuses so that we had a nice organized view of all deadlines.

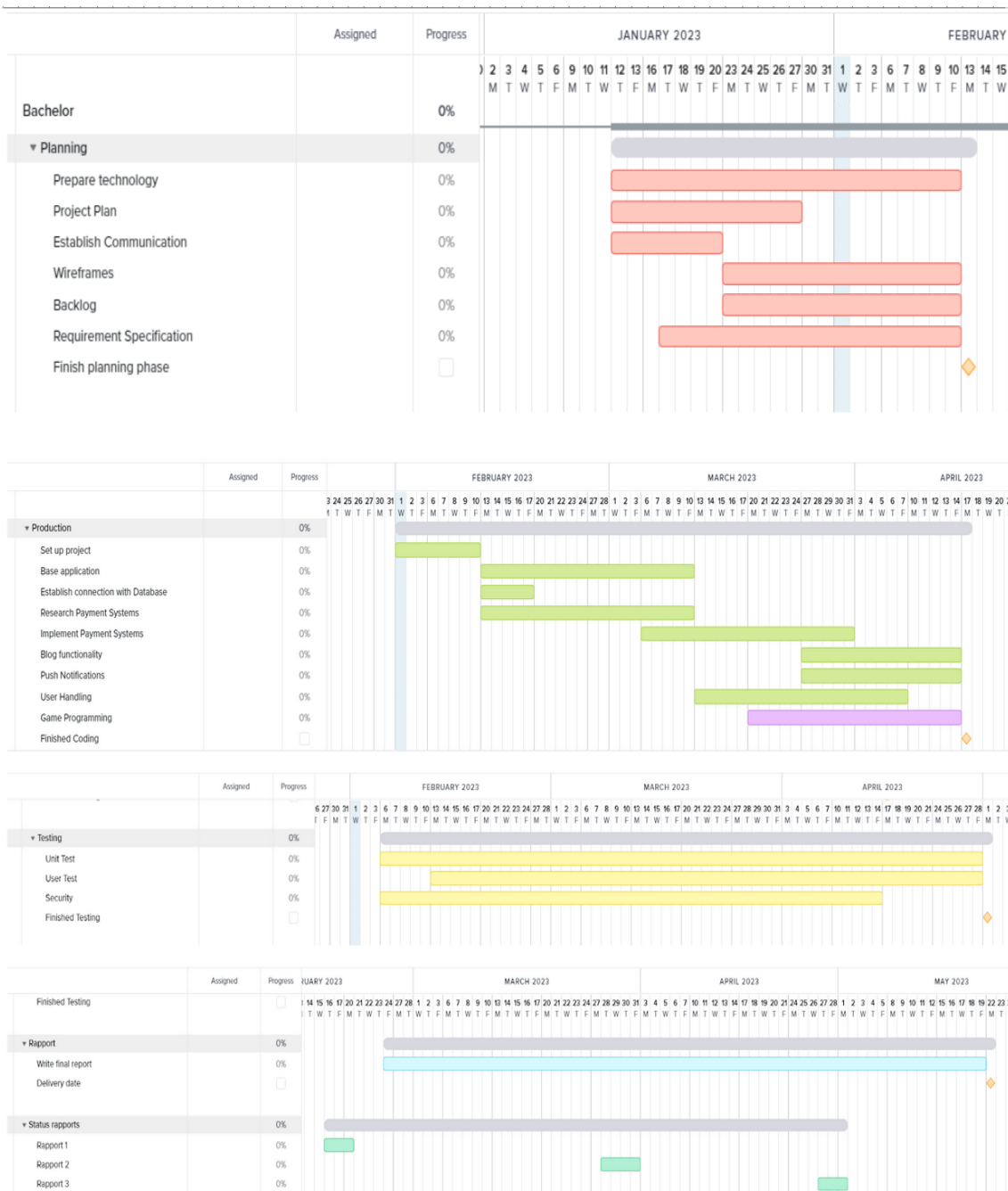


Figure 55: Gantt Chart

The planning phase went how we expected it to go. We planned pretty much everything in this phase, only changing and adding minor things at later stages.

Our production part was not followed as closely. While we managed to set up the project within the allotted time frame and connect with the database, we quickly ran into a problem with the payment systems. Because Vitensenteret has an existing payment solution, we had to confer with them about how to solve this. This resulted in a long back and forth where Vitensenteret was in between systems and had yet to decide which to actually use. As a way to still manage our time effectively, we started to work on the games, which were

supposed to be added on if we saw that we had time. We also started work on the blog implementation. With the database came a lot of functionality that was added throughout the app, and we continually worked on the user experience. Eventually, we managed to get the payment systems up and running, however, due to all the issues around that topic, we did not have time to implement push notifications.

For testing, we continually tested everything we did and worked on different branches inside of git. We created unit tests, however, we were not able to perform the user tests until the last month of the project due to some missing features. Security has been in focus since we started production, and it was something we always considered when working.

The report was also in line, continually being worked on and documentation was added throughout the course of the allotted time. And with the status reports also being written, it always felt like we had a good grasp of our project.

While we have not spent as much time as we wished on the project, we are still happy with the outcome and know that we did not have more to give.

12.2 Alternative Technology and The Choices We Made

12.2.1 Firebase Realtime Database vs. Firebase Firestore vs. SQL

When deciding what database to use we looked at a few different things that were important to the project.

- Our previous knowledge
- Ease of use
- Familiarity for Vitensenteret
- Future use

We have used MySQL as a Database in previous courses, and that meant all team members were familiar with the process behind design and use. We had no way of hosting such a database, other than locally on our own machines, which would be quite impractical for this project. This was also quite far from the database Vitensenteret already used, and would need additional setup were they to use it after we were done developing the application.

Firebase Realtime Database was new to us, and we were therefore a little unsure at first. The documentation for this database is pretty comprehensive, which made the familiarization process easier. Because both the database and the framework is made by Google, there were no huge barriers to connecting the two. This was also the database

Vitensenteret used for their current solution for yearly memberships and would be easier for them to take control of after the development process.

Firebase Firestore is like Firebase Realtime Database a no-sql database and was therefore also new to us. From the documentation we could see that this had a different, and more comprehensive structure, which would probably benefit us. Since this is still part of Firebase, it would not have been difficult for Vitensenteret to get to know it. It would also have been easy to hand the database to them after development.

	MySQL	Realtime Database	Firestore
Previous knowledge	High	Low	Low
Ease of use	Low	Medium	Medium
Familiarity for Vitensenteret	Low	High	Medium
Future use	Low	High	High

Table 1: Table of database choices

This all meant that for us Firebase Realtime Database became the best choice for this project.

12.2.2 Kotlin vs. Flutter vs .Net vs React Native

When it came to choosing a framework and language for the project, we had a few options. It was important to us that:

- Cross-platform
- Documentation
- Future development
- Whether we like the language
- Size of the finished application

Kotlin was something the whole group had experience in. However, it is solely for development for Android Phones. This meant that Kotlin was not a viable option for us.

Flutter is a framework developed by Google, and uses Dart as its programming language [28]. It allows for development for Android, iOS, macOS, Windows, Linux, and the Web. When looking around, we found a good amount of documentation, and there was also quite a lot of user-submitted documentation. The framework is fairly new, being released in 2017, and is in continuous development.

Xamarin.Forms and .NET MAUI are the platforms offered by the Microsoft .NET framework, both of which use C# as their programming language. Xamarin is the older platform and was deprecated by Microsoft in 2021 in favor of MAUI, the newer platform[29]. As the switch was quite recent it meant Xamarin had a lot of documentation online while MAUI had very little. Both Xamarin and MAUI allow for development in Android, iOS, macOS, Windows, and the Web. However, MAUI doesn't support Linux which was a major issue.

React Native is another popular framework for developing cross-platform apps, it uses JavaScript [30]. React has the largest following of all the frameworks and thus a lot of documentation, released in 2015 [31], it is among others used by Facebook, Outlook, and Discord. React Native allows for development in Android, iOS, macOS, windows, and the web. But just like MAUI, doesn't support Linux without additional plugins. In addition, we also wanted to avoid JavaScript as none of the members of the group were fans of the language.

In the end, we decided to consult one of our programming teachers, Mariusz Nowostawski whom we knew had experience with a lot of different programming languages and frameworks. We asked him if he had any tips or preferences when it came to the Cross-Platform frameworks .NET MAUI and Flutter. He told us that he had used Flutter and liked it but not MAUI which he was more skeptical of. He gave us the tip of writing some simple "Hello World" programs in both frameworks and comparing the process and results. We followed his advice and found that Flutter was very satisfying to program in and decided to use it for the project.

	Kotlin	Flutter	Xamarin.Forms	.NET MAUI	React Native
Cross-platform	Low	High	High	Medium	Medium
Documentation	High	High	High	Low	High
Future development	Low	High	Deprecated	High	High
Likeability of language	Medium	High	High	High	Low
Application size	Low	High	High	High	Low

Table 2: Table of frameworks

12.2.3 Payment system

The payment system was one of the biggest dilemmas we had during the project duration. The choice was up to Vitensenteret with us only giving tips at solutions. Initially, we had recommended and agreed with our contact at Vitensenteret to use Vipps. Vipps is a natural choice of payment system in Norway as approximately 75% of Norwegians use the service [32]. We started research into what we needed and how to implement Vipps payment into our application.

A few weeks later we got the message from Vitensenteret that they no longer wished to have a payment system in the app, as Vipps required manual input into their current accounting system. We responded that a payment system was an integral part of the bachelor as it was presented, and asked if they had some other payment solution they would rather have us look into and eventually disable for release. We were then told they were considering a payment system called Duell [33] for their checkout system and we started looking into how to implement that into our application to match. Not long after we were told that Duell was off the table and returned to looking into Vipps. Then after another few weeks, we were given another solution to look into, Stripe. Easter had just ended and at this point, we told them we wouldn't have time to look into other payment solutions if they decided to change their minds again.

Although Stripe was a sort of last-minute solution we quickly got it to work with its large amount of support and documentation. Especially helpful were the premade Stripe widgets for Flutter.

12.3 What Would We Have Done Different Today?

12.3.1 Database choice

While choosing to work with Firebase was a good choice, if we were to choose again we would go with Firestore. Firestore uses a different way of organizing data, in documents and collections rather than nodes and levels. It also allows for a different way to fetch data, including querying for specific data, which is something the product needs. Per today fetching data is inefficient and tedious, much of this could be solved by switching to Firestore.

12.3.2 Payment system

One of the main things that we wished turned out differently was the payment system. We spent way too much time looking at different payment systems that we ended up not using. If we could have agreed on a payment system and stuck with it, we could have it implemented a lot earlier in the development process. In the end, this hurt the final product, while the payment system is functional in the app, it's far from ready to be deployed in its current state. While payment works it lacks the security and polish you expect from something so important. The largest issue is the fact that major parts of buying tickets are handled client-side which means there's a decent chance for something to go wrong. As it stands the user's device has to check if a payment succeeds and then add the tickets itself, but ideally Stripe should tell the Firebase functions that the payment succeeded and have the function add the tickets. This would avoid a niche situation where

you pay for a ticket but the ticket is never added to Firebase because the phone can't communicate with it.

12.4 Evaluation of the Group Effort

The group members have different experiences and interests, and this has made the group dynamic really good. When it came to dividing up responsibilities, there were never any disagreements on who should do what. Each of us has wanted to work on different things within the project, but this has not taken away from the end goal. The group has worked as well as we could, and made the best of the circumstances we were under. We would have loved to dedicate even more time to this project, but due to other commitments, we were unable to do that. Apart from a couple of misunderstandings and back-and-forths from the project owner, there have been no real issues.

12.5 Conclusion

For this app, we developed a multi-platform interface that can be used on almost any device. We connected to a database and an API, and we used both technologies we were familiar with, as well as learned new technology for this project.

We have learned a lot from this bachelor project, and are thankful for the opportunity that Vitensenteret Innlandet has given us.

13 Sources

Bibliography

1. Vitensenterforeningen. Vitensenterforeningen. Available from: <https://www.vitensenter.no/om-foreningen/> [Accessed on: 2023 May 21]
2. GlueUp. 5 Benefits to Using Digital Membership Cards. Available from: <https://www.glueup.com/blog/digital-membership-cards> [Accessed on: 2023 Jan 17]
3. NTNU. PROG2900 - Bacheloroppgave. Available from: <https://www.ntnu.no/studier/emner/PROG2900/#tab=omEmnet> [Accessed on: 2023 Jan 16]
4. Google. Target API level requirements for Google Play apps. Available from: <https://support.google.com/googleplay/android-developer/answer/11926878?hl=en> [Accessed on: 2023 Jan 24]
5. Apple. App Store. Available from: <https://developer.apple.com/support/app-store/> [Accessed on: 2023 Jan 24]
6. NTNU. Studieplan. Available from: <https://www.ntnu.no/studier/studieplan#programmeCode=BPROG&year=2019> [Accessed on: 2023 Apr 16]
7. FlutterFire. Flutterfire. Available from: <https://firebase.flutter.dev/> [Accessed on: 2023 May 21]
8. Consulting I. General Data Protection Regulation GDPR. Available from: <https://developer.apple.com/support/app-store/> [Accessed on: 2023 Feb 7]
9. Google. Firebase Authentication. Available from: <https://firebase.google.com/docs/auth> [Accessed on: 2023 Feb 9]
10. Firebase. Privacy and Security. Available from: https://firebase.google.com/support/privacy%5C#data%5C_encryption [Accessed on: 2023 May 16]
11. Babich N. Z-Shaped Pattern For Reading Web Content. Available from: <https://uxplanet.org/z-shaped-pattern-for-reading-web-content-ce1135f92f1c> [Accessed on: 2023 Jan 31]
12. Babich N. F-Shaped Pattern for Reading Content. Available from: <https://uxplanet.org/f-shaped-pattern-for-reading-content-80af79cd3394> [Accessed on: 2023 Jan 31]
13. Wikipedia. Mastermind (board game). Available from: [https://en.wikipedia.org/wiki/Mastermind%5C_\(board%5C_game\)](https://en.wikipedia.org/wiki/Mastermind%5C_(board%5C_game)) [Accessed on: 2023 May 16]
14. Wikipedia. Tower of Hanoi. Available from: https://en.wikipedia.org/wiki/Tower%5C_of%5C_Hanoi [Accessed on: 2023 Apr 16]

-
15. Lego. Learn to Program. Available from: <https://www.lego.com/en-us/themes/mindstorms/learntoprogram> [Accessed on: 2023 May 16]
 16. Stripe. PaymentIntents. Available from: <https://stripe.com/docs/api/payment-intents> [Accessed on: 2023 May 20]
 17. Stripe. How Stripe Apps work. Available from: <https://stripe.com/docs/stripe-apps/how-stripe-apps-work> [Accessed on: 2023 May 20]
 18. Firebase. Firebase Realtime Database. Available from: <https://firebase.google.com/docs/database> [Accessed on: 2023 May 20]
 19. Firebase. Structure your database. Available from: <https://firebase.google.com/docs/database/web/structure-data> [Accessed on: 2023 May 18]
 20. Ng M. Flutter Wordle Clone Mobile/Web/Desktop Tutorial — Apps From Scratch. Available from: https://www.youtube.com/watch?v=%5C_W0RN%5C_Cqhpq%5C&ab%5C_channel=MarcusNg [Accessed on: 2023 Feb 25]
 21. Dart. Effective Dart: Documentation. Available from: <https://dart.dev/guides/language/effective-dart/documentation> [Accessed on: 2023 May 20]
 22. Dart. Effective Dart: Style. Available from: <https://dart.dev/guides/language/effective-dart/style> [Accessed on: 2023 May 20]
 23. Flutter. Build and release an iOS app. Available from: <https://docs.flutter.dev/deployment/ios> [Accessed on: 2023 May 19]
 24. Flutter. Build and release an Android App. Available from: <https://docs.flutter.dev/deployment/android> [Accessed on: 2023 May 19]
 25. Malin Foss Philip Morud SSE. GitLab Repository. Available from: <https://gitlab.com/hirkasa/billettapp-for-vitensenteret/> [Accessed on: 2023 Jan 23]
 26. team BF. flame 1.7.3. Available from: <https://pub.dev/packages/flame> [Accessed on: 2023 May 15]
 27. Firebase. Set up a Firebase Cloud Messaging client app on Flutter. Available from: <https://firebase.google.com/docs/cloud-messaging/flutter/client> [Accessed on: 2023 May 20]
 28. TheManInTheBlackHat. Flutter (software). Available from: <https://en.wikipedia.org/wiki/Flutter%5Ctextunderscore/software> [Accessed on: 2023 May 19]
 29. Ortinau D. Introducing .NET MAUI – One Codebase, Many Platforms. Available from: <https://devblogs.microsoft.com/dotnet/introducing-dotnet-maui-one-codebase-many-platforms/> [Accessed on: 2023 May 20]
 30. Native R. React Native. Available from: <https://reactnative.dev/> [Accessed on: 2023 May 19]
 31. Reamgoxer. React Native. Available from: https://en.wikipedia.org/wiki/React%5C_Native [Accessed on: 2023 May 19]

-
32. Vipps. Vipps i tall. Available from: <https://www.vipps.no/om-oss/vipps-i-tall/> [Accessed on: 2023 May 20]
 33. Duell. Duell. Available from: <https://www.kasseservice.no/> [Accessed on: 2023 May 20]
 34. Firebase. Main page. Available from: <https://firebase.google.com/> [Accessed on: 2023 May 17]
 35. AnomieBOT. .NET framework. Available from: https://en.wikipedia.org/wiki/.NET%5C_Framework [Accessed on: 2023 May 19]

Appendix

A Gantt

B Project plan



VITENSENTERET
INNLANDET

‘App and ticketing system for Vitensenteret Innlandet’

Project Plan

Malin Foss

Philip Morud

Susanne Skjold Edvardsen



NTNU

Kunnskap for en bedre verden

1. Scope	3
1.1. Background	3
1.2. Subject area	3
1.3. Delimitation	4
1.4. Task description	4
2. Goals and constraints	5
2.1. Project goals	5
2.2. Constraints	6
3. Project organization	6
3.1. Responsibility and roles	6
3.2. Routines and group rules	7
4. Development Process	8
4.1. Development Model	8
4.2. Meetings	8
5. Organizing and quality assurance	9
5.1. Documentation, standards, configuration management, tools ...	9
5.2. Plan for Inspections and Testing	10
5.3. Project level risk assessment	11
6. Plan for execution	13
6.1 Gantt	13
7. Sources	14
Attachment A	15

1. Scope

1.1. Background

Vitensenteret is a larger organization that has many locations around Norway. These locations are open to visitors in possession of a ticket, and contain many exhibitions in which the visitors can play around with science and educate themselves on fun topics. Vitensenteret Innlandet (VI) is one of these locations.

Entry to Vitensenteret requires a ticket that is validated through a database in Firebase (*Firestore*, 2012). However, in these times where more and more people carry around their phones first and foremost (*FinTech*, Unknown) and in most cases exchanging their wallet for it, having a person carry a membership card is getting more and more difficult. In this context, Vitensenteret contacted NTNU with a bachelor thesis topic, asking for a solution to this problem.

Vitensenteret has asked that we develop a mobile solution so that users have a way to purchase, and track tickets purchased for entering Vitensenteret Innlandet. Additionally they also want a way to alert users of the app of events, sales or other important things through notifications. Lastly they want an interactive part of the app, a way to retain use of it even after a visit has been done.

1.2. Subject area

A physical membership card is very common for members to gain entry to the locations. As mentioned above, the use of such cards are becoming less common. The reason for this may be the decrease in use of a wallet, or simply the costs in manufacturing the cards, cardscanners and equipment to support this system. (*GlueUp*, 2023).

Phones are already on a person at all times. This device holds many important apps, like social platforms, BankID, or games. Making an application for phones which will contain all that is needed during a visit to Vitensenteret, like tickets and an overview over events at the location.

The subject for this project is the development of an application for a mobile device. This application should be able to run on both android phones as well as iOS devices. The process of developing an app involves a list of areas. This list includes designing the app so that the application feels good to use. The use of databases and cloud technologies, gamedesign, developing software for the phone in general.

In addition to the above, this project will teach the group members how to work together on a project of this scope, and while the final product has an important role in the final evaluation, one of the first limitations we employ as a group is to focus on the developmental process and documentation around the project. What this means for the project is that the wishes of the employer comes second in regards to what we wish to accomplish in relation to the finished project plan.

1.3. Delimitation

Instead of using the outdated systems at Vitensenteret, our group will develop a mobile application which will allow users to purchase tickets to gain entry to the location. In addition to this, the users will get updates about Vitensenteret in the app, and be able to play a brain teaser game. This will all be developed by the group, with a focus on proper documentation and solid workflow.

1.4. Task Description

- The group will make an application for the mobile platform. This application will run on both Android and iOS.
- This application will communicate with a premade database in Firebase to administer the users who have a valid ticket to enter locations. This database was made by Vitensenteret.
- The application shall allow for users to pay for tickets through the app, for example via Vipps (*Vipps*, 2015) or some other method.

- The app will also use cloud technologies to have a notification system, where administrators can create alerts and send it out as push notifications to general users.
- The app will allow administrators to push blogs with content from Vitensenteret into a feed which all users have available. This feed should also contain notifications that users received through push notifications.
- The application will be developed using design principles and will need to be adequately secured to make sure that there are no security breaches.
- Should there be time to develop an additional function to the application, this will be an interactive game. This game will be in line with other functions of Vitensenteret, and will consist of some form of brainteasers.

2. Goals and constraints

2.1. Project goals

Impact Goals: (Effektmål)

The main goal of this project is to develop an application for users of Vitensenteret so that the users are not dependent on bringing anything other than their phones to the locations. The application will bring the presence of Vitensenteret to those who have the app, creating a familiarity with it. Vitensenteret has a goal to reach 50-60 000 visitors in 2023, our goal is that 40% of those use the app before, during or after their visit.

Project Goals: (Resultatmål)

The main aspects that we aim to develop is a way to showcase tickets purchased, like a yearly one, as well as allow for purchasing these tickets through the app. In addition to this we will develop a notification system where the administrators of the app can post notifications so that normal users of the app are alerted to events or

sales at Vitensenteret. Safety is an important factor in all of this. If the development of these functions reach a point where a person in the group is able to focus on another part of the project, this person will develop an interactive game for the application.

It is also imperative that we make the app a base for further development as Vitensenteret has many plans for further development.

Educational Goals: (Læringsmål)

The goal with this task is to learn how to develop a bigger project than what has been previously worked on throughout this education. With a focus on the developmental process from day one, where proper documentation is vital, as well as having a fluid grasp and accomplishment of the agile development method that the group chose for this task, which is Scrum. In addition to this, we will learn to work with an associate outside of NTNU. Up until this point, most, if not all, tasks that have been worked on have been internal, while now that the group is working with a team outside of the organization, we will get to experience and solve unique situations that the group members may not have encountered before.

Additionally we aim to take many of the subjects we have had through our education, and combine it into this task. This encompasses everything from design, coding to task writing and ethics. All shall be done together in a structured group where each member has shared and separate roles. For the educational plan on this subject from NTNU (*NTNU studies*, Unknown).

2.2. Constraints

- The app has to work on both Android and iOS. Target api 31 (*Google*, Unknown), and iOS 15 (*Apple*, Unknown).
- The app has to be user friendly, for kids and adults.
- We have to follow the Vitensenteret design handbook when designing certain elements of the app.
- The app must work with the existing database in Firebase.

- The code has to be well documented and made for future development.

3. Project organization

3.1. Responsibility and roles

- Project manager: Malin Foss
 - Keep an overview of project progress
 - Responsible for all outgoing communication
- Documentation manager: Susanne Skjold Edvardsen
 - Makes sure there are notes taken from all meetings
 - Has main responsibility for all documentation
- Scrum master: Philip Morud
 - Responsible for all sprint related meetings

3.2. Routines and group rules

See attachment A for a signed copy of the group rules.

- Each member of the group should aim to work for approximately 30 hours per week. In the case where the total is repetitively below 30 hours, a meeting will be held amongst the group members to discuss causes and solutions.
- The group will host one physical meeting each week where it is expected that all members are present. Should a member be unable to attend this meeting, they should notify the other members of the reason in advance. (If all the group members agree, the meetings can be scheduled less often, for instance biweekly).
- All the hours spent working towards this project should be written down. These time logs should be available in the final delivery.
- Each member is responsible for showing up to scheduled meetings with the group, the mentor and the employer.
- Group members are obligated to warn the group if they believe themselves, another group member, or the group as a whole is starting to fall behind so measures can be taken to avoid losing too much time overall.

- In the case of disagreements within the group there are three levels of measures to be taken. All positives and negatives are to be weighed together in a discussion, if no decision is reached the mentor can be involved to get some outside perspective. Last resort is putting it to a vote.
- If a group member breaks several rules or repeatedly breaks rules it has to be brought up and escalated in the following order:
 - Discussion with the entire group
 - Written warning with cause, measures to fix the issue(s) and potential consequences
 - Conversation with mentor and the entire group
 - Written exclusion from the group (no later than four weeks before the delivery date)

4. Development Process

4.1. Development Model

We are planning to use the agile method Scrum. We want to be able to adapt our product to the product owner as we go, and this is best achieved by using an agile method. Scrum allows us to keep a lot of structure to the development, but also allows for changes to long term plans easily.

The project has a few technological challenges and uncertainties that might require us to change our approach, with Scrum we remain able to make these decisions along the way.

We will use Scrum with week long sprints in the beginning, and then an evaluation later on might change the sprint length to two weeks. We will be using a backlog with all long term goals and future features. In addition we will be using an issue board where tasks for each week will be posted in the four categories open, in development, testing, and important. Open will act as the sprint backlog, in development is for started tasks, testing for finished tasks that need review.

Important is reserved for any tasks that need to be resolved quickly and/or is a bottleneck.

4.2. Meetings

Sprint planning meeting

Sprint start is every Tuesday 11.30-13.00 as we have week long sprints, if we increase the sprint length sprint start will be every other Tuesday. Here we will figure out what the focus of the coming sprint is going to be, and pull tasks from the backlog, modify them and put them in the sprint backlog. We will also identify if any issues should be classified as important, or if any issue requires several people to work on.

Sprint review meeting

Sprint review meetings will be held Tuesdays 10.00-11.30. The goal of these meetings are to review our work in the past sprint, especially to identify potential problems. We will also use what we learned to better plan ahead, and to create a better product. If anyone has learned something useful during the sprint, this is an arena to share that.

Sprint retrospective meeting (?)

Sprint retrospective meetings will be held every other sprint before sprint reviews. The focus of the meeting will be to review the sprints, to see if the length will need to be changed, or if issues are too big or small.

Mentor meetings

We will have weekly meetings with our mentor Frode Haug. These meetings will primarily be used as a forum to discuss methods, report writing and anything else we need someone else to spar with.

Project owner meetings

It is important to us to keep an open and continuous dialog with the project owner.

We will aim to have meetings with Vitensenteret every other week. This way we can show of progres, and get pointers as to whether we are heading in the right direction product wise. We will also be able to catch potential issues, problems or misunderstandings during these meetings, and be able to work on it in the coming sprint.

5. Organizing and quality assurance

5.1. Documentation, standards, configuration management, tools ...

We should discuss this.

Technology:

Name:	Purpose:
Discord	Communication in group
Email	Communication outside group
Google Docs	Writing and managing documents
Draw.IO	Creating diagrams
Overleaf LaTeX	Writing large documents
Gantt	Managing Gantt Chart
Firebase	Managing the Database and requests
Remarkable	Managing backlog
GitLab	Repository for code, as well as version control.
Flutter	Cross-platform coding framework

5.2. Plan for Inspections and Testing

All group members are responsible for making sure their code works as expected before pushing to the repository, and to test for edge cases before merging to the main branch.

For all functions that return something or change something should be tested with unit testing. We are not using test driven development, so tests can be written after the fact.

We plan to setup and use a CI/CD pipeline on GitLab, this pipeline will compile and run some basic testing. Due to the restrictions of a free account this will only be done when merging branches.

User testing will be done regularly during our meetings with Vitensenteret. We will start with a simple wireframe demo on paper, and evolve from there.

Since this app will deal with user data it is important that these data are safe, we should test with this in mind.

5.3. Project level risk assessment

Number	Risk	Probability	Impact	Measures
1	A group member gets sick	Very likely	High	Yes
2	The group disagrees	Very likely	Medium	Yes
3	Technology is/becomes unavailable	Likely	High	Yes
4	The project is too small	Unlikely	Medium	Yes
5	Unable to finish product	Likely	High	Yes
6	Product already exists	Very unlikely	High	No

Number	Risk	Probability	Impact	Measures
7	A competing product is created	Likely	Low	No
9	Product no longer wanted/useful	Very unlikely	High	No
9	Product owner goes bankrupt	Very unlikely	High	No
10	Product is not safe enough	Likely	High	Yes

Measures

Number	Measure	After measure
1	We will be working together on most parts of the project, and during sprint review meetings we will catch each other up on what we have worked on. This way we avoid that someone being absent from the group will have	Low impact
2	The issue will be brought up with the entire group and positives and negatives will be discussed. If an agreement is not reached the mentor can be involved for advice. The next measure is taking it to a vote.	Low impact
3	We will explore different technologies for each choice we make. If one becomes unavailable, there will likely be alternatives, even if they might be less suitable.	Medium impact
4	The project owner has presented us with a list of wanted/possible features of the app. If we run out	Low impact

Number	Measure	After measure
	of things to do, there is more on that list to pick from.	
5	During the development process we have placed milestones as a way of checking the progress, and to make sure we don't fall behind. Every member is responsible for telling the group if they believe the project is falling behind. All parts of the project are put in order of priority, if we are in danger of not being able to finish everything we know what to cut.	Unlikely with medium impact
10	We will be testing for safety along the way to ensure that the product we are developing is safe enough.	Low impact

6. Plan for execution

6.1 Gantt

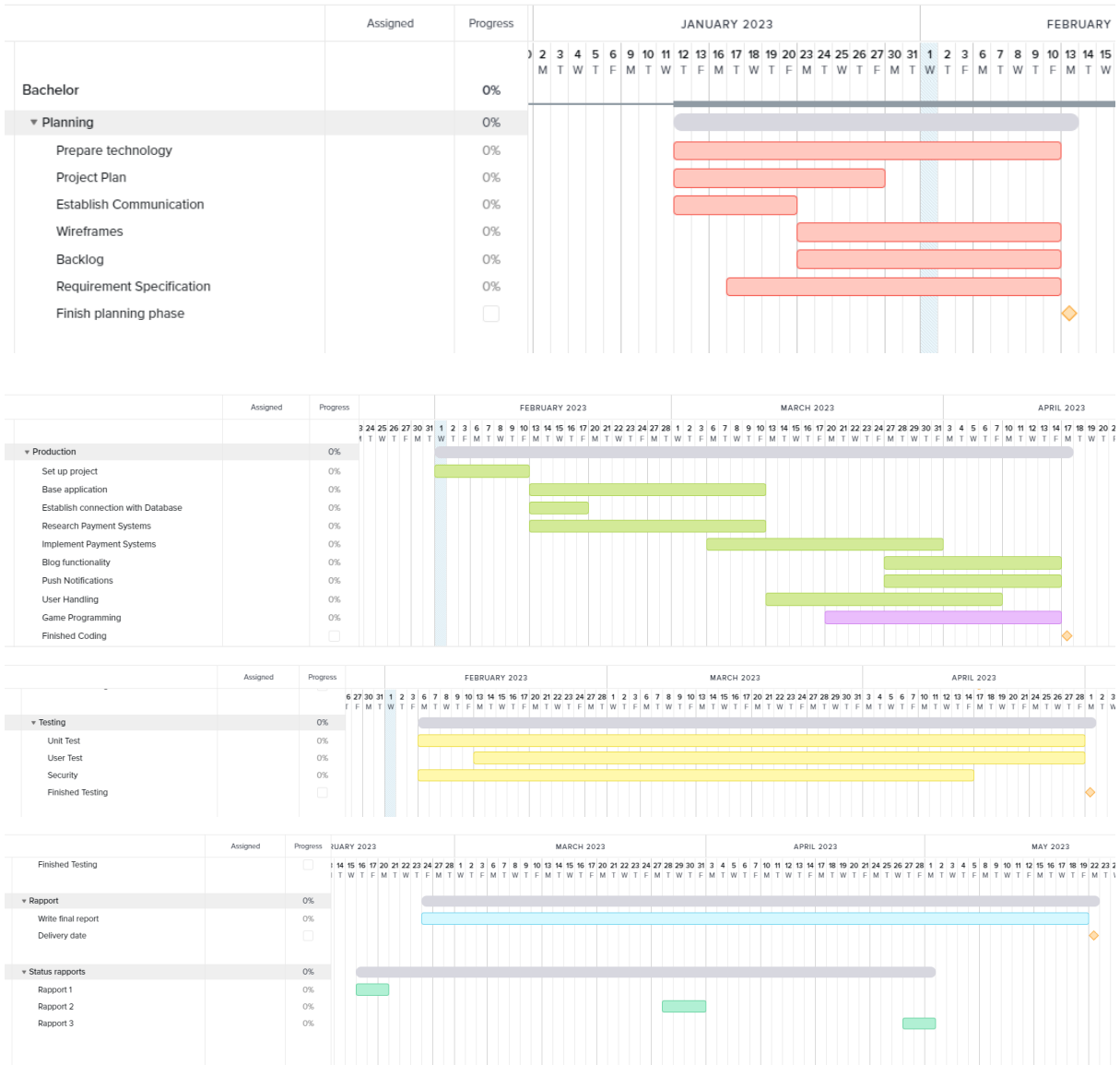


Chart made with (TeamGantt, Unknown)

This is our gantt diagram. It was constructed with a light start where everyone is focused on planning and organizing the project. This period will last until 13.

February, at this point all the planning should be finished, as well as some parts of the project started.

The production is set to start February 1st. This includes setting up the project and coding the contents, like communications with the database and making the blog. In addition to this we have the game programming in its own color to signify that it has a lower priority.

Testing starts a week after the production starts, although measures have been taken during the planning stages. In the testing phase, we have an emphasis on unit tests, and user tests.

The writing of the final report is initiated already in february, and it is meant to be on the backburner until more of the production is underway. The final report is set to be done the Friday before the delivery date, the following monday.

7. Sources

Vitensenteret (Unknown). Fetched from:

<https://vitensenteret.no/> (Date: 16.01.2023)

FireBase (2012). Fetched from:

<https://firebase.google.com/> (Date 17.01.2023)

FinTech (Unknown). Fetched from:

<https://fintechmagazine.com/digital-payments/75-of-consumers-now-using-mobile-wallets-survey> (Date 16.01.2023)

NTNU (Unknown). Fetched from:

<https://www.ntnu.no/> (Date 16.01.2023)

NTNU Studies (Unknown). Fetched from:

<https://www.ntnu.no/studier/emner/PROG2900/#tab=omEmnet> (Date 16.01.2023)

Vipps (2015). Fetched from:

<https://vipps.no/> (Date 16.01.2023)

Teamgantt (Unknown). Fetched from:

<https://app.teamgantt.com/> (Date. 18.01.2023)

Google (Unknown). Fetched from:

<https://support.google.com/googleplay/android-developer/answer/11926878?hl=en> (Date 24.01.23)

Apple (Unknown). Fetched from:

<https://developer.apple.com/support/app-store/> (Date 24.01.23)

GlueUp (2023) Fetched from:

<https://www.glueup.com/blog/digital-membership-cards> (Date 01.02.23)

Attachment A

Rules of the Group

- Each member of the group should aim to work for approximately 30 hours per week. In the case where the total is repetitively below 30 hours, a meeting will be held amongst the group members to discuss causes and solutions.
- The group will host one physical meeting each week where it is expected that all members are present. Should a member be unable to attend this meeting, they should notify the other members of the reason in advance. (If all the group members agree, the meetings can be scheduled less often, for instance biweekly).
- All the hours spent working towards this project should be written down. These time logs should be available in the final delivery.
- Each member is responsible for showing up to scheduled meetings with the group, the mentor and the employer.
- Group members are obligated to warn the group if they believe themselves, another group member, or the group as a whole is starting to fall behind so measures can be taken to avoid losing too much time overall.
- In the case of disagreements within the group there are three levels of measures to be taken. All positives and negatives are to be weighed together in a discussion, if no decision is reached the mentor can be involved to get some outside perspective. Last resort is putting it to a vote.
- If a group member breaks several rules or repeatedly breaks rules it has to be brought up and escalated in the following order:
 - Discussion with the entire group
 - Written warning with cause, measures to fix the issue(s) and potential consequences
 - Conversation with mentor and the entire group
 - Written exclusion from the group (no later than four weeks before the delivery date)

Date:

18. januar 2023

Signed:

Malin Foss

Malin Foss

Philip Morud

Philip Morud

Susanne S. Edvardsen

Susanne Skjold Edvardsen

C Contract

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett

Er den rett som den som skaper et åndsverk har til å fremstille eksemplarer av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater

Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater

Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn

Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring

Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt:	
Veileder ved NTNU: e-post og tlf.	FRODE HAUG FRODE.HAUG@NTNU.NO 95055636
Ekstern virksomhet: Ekstern virksomhet sin kontaktperson, e-post og tlf.:	Vitensenteret Innlandet 94217171 GAVIN ROBB, GAVIN.ROBB@VITENSENTERET.NO
Student:	Malin Foss
Fødselsdato:	09.02.2000
Ev. flere studenter ¹	Susanne Slijold Edvardsen 17.1.1999
	Philip Morud 13.12.1999

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato:	9.1.2023
Sluttdato:	22.5.2023

Oppgavens arbeidstittel er:

Vitensenter Innlandet - Appen

¹ Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven². Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

² Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

<input type="checkbox"/>	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven
--------------------------	--

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

<input checked="" type="checkbox"/>	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt
-------------------------------------	---

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

Bedrift skal ha mulighet til å videreutvikle applikasjon

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

<input checked="" type="checkbox"/>	Oppgaven skal være offentlig
-------------------------------------	------------------------------

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss	Sett dato
<input type="checkbox"/>	ett år
<input type="checkbox"/>	to år
<input type="checkbox"/>	tre år

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

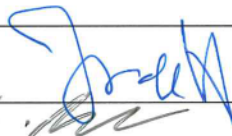

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder:	
Dato:	
Veileder ved NTNU:	
Dato:	
Ekstern virksomhet:	
Dato:	20/1/2023
Student: Susanne S. Edvardson,	
Dato:	18.1.2023
Ev. flere studenter	Melin Foss 20.1.2023
	Philip Mørud 20.1.2023

D Status report 1

Status 15. February 2023

What is finished, what is currently being worked on?

Finished	Working on
Project Plan	Requirement Specification
Setup Project	Implement Databases
Decide on (most) of the Technology	Implement Blog Solution
Designed (most) of the Infrastructure.	Implement Wordle and Hannoy's Tower
	Payment solutions

Anything that is taking longer to do?

Everything is going according to plan. Following the gantt chart created, every task that is supposed to be finished by now, has been finished. As well, any task that should have been started, has been started. In fact, we are ahead of schedule and have started other parts of the project ahead of time, namely implementing the blog functionality as well as the game functionality. This feels natural because it lends itself that everyone on the team then has a programming task, without overlapping with each other.

Planning:

Planning flows naturally, and work is implemented according to schedule. The only thing that is hindering progress a little is the implementation of payment systems. We want to use Vipps as it offers both mobile and debit solutions, but as different payment solutions/options have different commission fees we need to make sure our employer agrees with this. Therefore, before moving on with this implementation the group needs to meet with Vitensenteret to discuss if they still want Vipps (and which Vipps option), or if they want another solution.

Organizing the group work and responsibilities:

Roles fall naturally, and everyone is doing their part. Documentation, coding, research, everything is being worked on according to the responsibilities in the group.

Requirement specification

A version of this was sent in, and edits are being made to the document. After this, it will be sent in once again for reevaluation.

Report writing

The report has been started in Overleaf/LaTeX and content is being generated and updated. Among other things we have written summaries of all meetings, documentation for choices of technology, research documentation for instance in regards to GDPR.

Status in regard to the points above:

The work is ahead of the projected calculations for progression.

Opportunities or problems:

With being ahead of schedule we may be able to add some more incentive reasons for users to download the app, like more games and blog functionality, however nothing will be promised until the development reaches a point in which we can confidently add more content.

Motivation:

Teamwork, work environment and organization: Team works excellent together. No problems so far.

Relationship statuses: Everyone works together, no bad vibes or experiences in the work.

Contact with employer and mentor:

We have close contact with both, and do not have any problems around communications beyond the rescheduled meetings with Vitensenteret. When meetings were rescheduled, we were always able to meet with them, to discuss what we wanted to.

E Status report 2

Status 30.March 2023

What is finished, what is currently being worked on?

Finished	Working on
Project Plan	Payment solutions
Setup Project	Better ID Systems for tickets and blog posts
Decide on (most) of the Technology	Blog Solution
Designed (most) of the Infrastructure.	Settings Page
Requirement Specification	Ticket Page
Basic Application	Final Report
Basic Database with connection to App	
Implemented Wordle and Hanoi Tower	

Anything that is taking longer to do?

In regards to the gantt chart that was delivered in the project plan, we are taking the time we planned to do the planned tasks, but due to the problems surrounding the payment systems we have had to shuffle our tasks around. That means pushing game programming, user handling, blogs and ticket systems forward while we start with the heavy focus on payment systems from easter.

Planning:

Again, the planning and execution of the tasks are going fine. We are as mentioned a little behind on payment solutions and ahead in regards to other systems. We have decided and gotten the go ahead to use Vipps after much back and forth between the group and the management at Vitensenteret.

Organizing the group work and responsibilities:

The group work is still being equally divided, and all members work on their parts with communications to all members with each sprint start and review.

Report writing

We are continuing to update the documents and the report in overleaf. Documentation is going well, coding is well commented.

Status in regard to the points above:

The work is on track for the projection, however it is difficult to say if we are slightly ahead or slightly behind due to the change in our projected plan due to the payment systems being moved.

Opportunities or problems:

Since we have finished with the game implementations, Vitensenteret expressed a want for an additional game, and this will be added into the implementation.

Motivation:

Teamwork, work environment and organization: Team works excellent together. No problems so far.

Relationship statuses: Everyone works together, no bad vibes or experiences in the work.

Contact with employer and mentor:

Contact and communication with mentor is going great.

Contact with Vitensenteret is good as well, however there has been some miscommunications and delays in regards to the task description being altered by Vitensenteret. This issue is still being worked on being completely resolved, however we are on track to having it figured out and are currently looking into vipps solutions as the payment systems.

F Status report 3

Status 27.April 2023

What is finished, what is currently being worked on?

Finished	Working on
Project Plan	Payment solutions
Setup Project	Better ID Systems for tickets and blog posts
Decide on of the Technology	Admin Blog Page (Almost done)
Designed of the Infrastructure.	Ticket Page(Almost done, waiting for payment systems)
Requirement Specification	Final Report
Basic Application	Coloration/Themes
Basic Database with connection to App	Notifications
Implemented Wordle and Hanois Tower	
Blog Page	
Settings Page	

Anything that is taking longer to do?

Due to the issues around payment solutions, we are not following the gantt chart, and have moved things around. The blog solution is close to done, and is expected to be finished once the admin page is done. Notifications should be connected to the blog, but due to an oversight we are not sure we will be able to finish it. The ticket page, the page showing purchased tickets is working, and up and running, however due to the payment implementation being unfinished at this date, the page is not at a whole finished. The games are done, save for some minor tweaking, and the themes and colors we were given by Vitensenteret is slowly being applied to the application.

Planning:

The planning is once again fine, after even more back and forth with Vitensenteret we have settled on using Stripe for the payment systems. Due to all the issues around this the blog is not where we would want it, and we have delegated resources to get it caught up. The status when writing this report, is that the blog has the main functionality down, only missing the admin page and notification system.

Organizing the group work and responsibilities:

The group work is still being equally divided, and all members work on their parts with communications to all members with each sprint start and review.

Report writing

We are continuing to update the documents and the report in overleaf. Documentation is going well, coding is well commented. We have a base down with quite a bit of the earlier documentation like requirement specifications have been added.

Status in regard to the points above:

We think the effort in writing and making a base for the final report was a good idea, because now, while we are working to get all the code finished for our soft deadline the 1.may we are having doubts we will be entirely done with the coding. This means we will have to take some of the time estimated to work on the rapport, but since we are so well started with that at the time, the group is overall not concerned with time.

Opportunities or problems:

After finishing the last requested programming game and requesting tasks from Vitensenteret to use for the game, we were meet with some minor confusion, as they are unable to provide tasks in the wanted format for our firebase solution. Therefore the game will be left without any additional tasks to be solved, but have the functinaly to add this data at a later time when Vitensenteret has time to make these tasks fit the format in firebase.

Motivation:

Teamwork, work environment and organization: Team works excellent together. No problems so far.

Relationship statuses: Everyone works together, no bad vibes or experiences in the work.

Contact with employer and mentor:

Contact and communication with mentor is going great.

Contact with Vitensenteret is good as well, there have been some miscommunication and waiting on them gathering information, which has slowed the projects progress somewhat, but overall it has been good. And when communication first is established, it is good.

G Meeting Minutes

29.11.2022 Meeting with employer

Team Members present: All

Topic: Get a feel for the task

Summary: The group met with Gavin and discussed expectations around the task. The group was led around the area and got a private tour of the center.

12.01.2023 Internal planning meeting

Team Members present: All

Topic: Planning

Summary: The group agreement and rules were constructed as well as an overview of the schedule which the group will follow through the semester. The group decided to use the Scrum development method. In addition the timesheet was constructed and explained to all members.

17.01.2023 - 25.01.2023 Code Framework

Team Members present: All

Topic: Planning

Summary:

Flutter (Dart):

adv: Google, performance, Hot Reload, backwards and forwards compatibility, documentation

disadv: ui not as good as native apps, new (some functions are in early stages of development), apps are "large", learn a new programming language

Xamarin (.NET):

adv: microsoft supported, fast testing performance, good UI customization, near native, good documentation

disadv: "large" apps, overhead, deprecated

MAUI (.NET):

adv: evolution of Xamarin,

disadv: new (less documentation), unstable with larger applications, no Linux applications

React Native:

adv: largest community, small app sizes,

disadv: html

Consulted Mariusz about whether to use Flutter or MAUI. His response was that he had used Flutter, which he liked but not MAUI which he seemed more skeptical of. He recommended that we create simple hello world applications in both frameworks and compare them.

We created some simple applications using both frameworks and found that flutter was easier to use than we expected while MAUI was a lot like using kotlin to develop a native android app. Unfortunately due to the lack of Linux support meant that it would be a lot slower to test the app for those in the group who use Linux as their operating system meant they would have to emulate the app on an android device.

Malin liked flutter.

Sources relevant for this meeting:

<https://devathon.com/blog/flutter-vs-or-and-xamarin/>

<https://www.trio.dev/blog/xamarin-vs-react-native>

17.01.2023 Meeting with Mentor

Team Members present: All

Meeting with: Frode Haug

Topic: First meeting, and expectations for the project

Summary: An introductory meeting where Frode presented a couple documents for the group to fill, like the group rules, requirement specifications etc. In addition, expectations were discussed on the role Frode will occupy as our mentor and guide for the process first and foremost.

18.01.2023 Work Session

Team Members present: All

Topic: Work on project plan

Summary: Further discussions on which technology to use for the development of the mobile app. Gantt chart was constructed. Worked on a project plan.

20.01.2023 Meeting with Vitensenteret

Team members present: Philip and Malin

Meeting with: Gavin Robb (Vitensenteret)

Topic: Setting constraints and expectations, access to tools

Summary: Contract signed, no NDA needed. Discussed impact goals, and the constraints we had around the scope of the project. They would prefer the app to have interactive games. Establish a meeting schedule of every two weeks, precise dates and times will be discussed over mail. Access to the database and code from the previous ticket solution will be sent by mail. The general timeline of the development was talked about, as well as our choices of technologies.

24.01.2023 Meeting with Mentor

Team Members present: All

Meeting with: Frode Haug

Topic: Status Meeting

Summary: Got the signature from Frode on important documents.

24.01.2023 Work Session

Team Members present: All

Topic: Planning

Summary: Polished project plan. Made the first iteration of use case diagram as well as the domain model.

25.01.2023 Work Session

Team Members present: All

Topic: Planning

Summary: Delivered project plan. Worked on requirement documentation. Finished use cases as well as domain model. Decided on the use of Flutter over Net.Maui.

26.01.2023 Work Session

Team Members present: All

Topic: Planning & Setup

Summary: Received project plan from Mentor with some comments, discussed briefly in the group about them. Decided to improve the project plan after the next scheduled meeting with the mentor. Looked into the database that Vitensenteret has on Firebase, but was not able to gain access due to two factor authentication.

31.01.2023 Work Session

Team Members present: All

Topic: Planning & Setup

Summary: Setting up a work environment on all computers, however having some issues with one of the laptops. Worked on the contents for the requirement specification.

31.01.2023 Meeting with Mentor

Team Members present: All

Meeting with: Frode Haug

Topic: Receive feedback on project plan and wireframes

Summary: Received feedback on project plan, as well as wireframes. Feedback was generally good. Discussed use of flutter over other methods.

01.02.2023 Work Session

Team Members present: All

Topic: Planning & Setup

Summary: All temembers have flutter running on their computers. Git has been established as well as branches for testing. A physical wireframe is prepared for the meeting with the employer. Overleaf/Latex was set up for the final report, and a simple template was made. Rewrote project plan in accordance with the feedback given. Worked on requirement specification.

02.02.2023 Meeting with Vitensenteret

Team members present: All

Meeting with: Gavin Robb and associates (Vitensenteret)

Topic: Showcase wireframe and current application design. Gain access to the database.

Summary: In summary, Vitensenteret was mostly happy with the application. they had the following points to say:

- Want more languages than norwegian. Support should be developed for the English language.
- No need to develop the ticket system for anything other than a yearly card. Other options have a button which links to their webpage.
- The user needs to upload a picture of themselves when creating a profile. (Our original idea was to allow them some default avatars to choose from).
- The page where the user shows their valid tickets needs some form of animation so that it cannot simply be screenshot and edited to be used by another person.
- Instead of simply an admin, split it in two admin classes, an employee and an admin. Employees can see the database and generate codes for gift cards, but only admins can change the contents of the database.
- Instead of names, use icons. They want the blog to not be named blog. Suggestions: "Viten, Vitenbrev, Nyhetsbrev".
- Wish for more focus on games, although the group developing was adamant that the development of game functionality will not be prioritized over the ticket system. They suggested adding the following games: "Hanoys tårn, tangram, daglig sudoku".

02.02.2023 Work Session

Team Members present: All

Topic: Planning & Setup

Summary: After the meeting with Vitensenteret, and all members have gained access to the database, we discussed how to add the functionality needed to this database. Charts will be developed next week.

07.02.2023 Brief Update Meeting

Team Members present: All

Topic: Planning & Setup

Summary: Sprint start, issues added are making the charts for the database, doing some GDPR research, and generally use of flutter and attaching it to the database.

09.02.2023 Brief Update Meeting

Team Members present: All

Topic: Planning & Setup

Summary: Finished database design as well as requirement specifications. Connected the application to the database in firebase.

14.02.2023 Meeting with Mentor

Team Members present: All

Meeting with: Frode Haug

Topic: Receive feedback on requirement specification

Summary: Received feedback on requirement specification, generally good, but some reformatting was needed, as well as some additional content.

14.02.2023 Brief Update Meeting

Team Members present: All

Topic: Setup

Summary: Made connection to database for login for users as well as creating users. Research made on payment solutions. Wordle game basic functionality made.

15.02.2023 Work Session

Team Members present: All

Topic: Working

Summary: Delivered status rapport, started working on editing req spec.

21.02.2023 Work Session

Team Members present: All

Topic: Working

Summary: Working on editing req spec. Make a ticket for sessions after the user logs in. Continued work on blog functionality. Testing to get the app on a physical phone.

21.02.2023 Meeting with Mentor

Team Members present: All

Meeting with: Frode Haug

Topic: Status Meeting

Summary: The group got some feedback in regards to the statusrapport, and updated mentor on current standings.

27.02.2023 Emergency Group Meeting

Team Members present: All

Topic: Changing requirements from what was agreed upon with Vitensenteret

Summary: After receiving an email from our employer at Vitensenteret, detailing the situation, we were informed that they do not require payment systems in the application. However, removing this from the workload means that the workload will be substantially lessened, and the group is worried that the grade may suffer if we remove the plans to implement the payment systems. Additionally. Since the message from Vitensenteret was given so late, changing the design of the application to implement something with the same workload as the payment systems, would be too much work, since we would not only need to program it, but also design new systems, have a new research phase and so on. Our plan so far is to discuss this problem with our Mentor, and then have a meeting with Vitensenteret this coming Thursday.

28.02.2023 Mentor meeting

Team members present: Philip and Malin

Topic: Changing requirements from what was agreed upon with Vitensenteret

Summary: Continue status quo

Mentor suggests that moving forward with the original idea will be the best move. At this point in time removing this much will have a significant impact on the finished product, and it is too late to begin on something new. Frode will also speak with Tom since he has more to do with communication with task givers, we will get feedback on this conversation on Thursday. For now we resume work as usual.

02.03.2023 Meeting with Vitensenteret

Team members present: Philip, Malin and Gavin

Topic: Changing requirements from what was agreed upon with Vitensenteret

Summary: While the representative, Gavin, was understanding that removing the ticket purchasing system would lead to a much smaller bachelor thesis, he also stated that there are a couple problems with the proposed solution that we had gotten approved by him. During this meeting we were shown the ticket purchase system in action and came to learn that the accountant was working on gathering all these systems into one solution, and by adding our phone application solution to this, we would again be adding to the numbers of payment systems.

Additionally the use of Vipps in the solution was not wanted, as that would add a lot of work to keep track of all expenses and income. A followup with Vitensenteret was scheduled next week.

07.03.2023 Work Session

Team members present: All

Topic: Get everyone updated on each other's work

Summary: Briefly showed off work that has been done and the progress made. Furthermore the group discussed last week's issues around Vitensenteret changing the requirements for the application. After this work commenced on each of the assigned tasks for each member.

07.03.2023 Meeting with Mentor

Team members present: All

Topic: Followup up on last week's issues

Summary: We brought our mentor up to speed and discussed the future of the bachelor task. Furthermore we discussed that if more issues presented themselves, to contact and discuss the issues with Tom, but we also came to the conclusion that it seems we are working things out with Vitensenteret.

09.03.2023 Meeting with Vitensenteret

Team members present: All as well as Gavin and Hanne, Accountant, from Vitensenteret.

Topic: Discuss payment solutions

Summary: We discussed our grief with the loss of implementing a payment system, and were able to be updated on what platforms Vitensenteret is currently using, as well as how they are planning on using the payment systems in the future as they are currently in the act of gathering their various payment systems. They are primarily using something called duell, and when we did some brief research we

discovered that duell has an api that it's possible for us to use to connect to, therefore fulfilling the groups want to integrate a payment system, and also Vitensenteret wish to keep what they have.

16.03.2023 Work Session

Team members present: All

Topic: Work session

Summary: We continued work on our designated tasks. Progress was made on the game. pair programming was utilized for working on the ticket page for the app.

21.03.2023 Work Session

Team members present: All

Topic: Work session

Summary: We continued work on our designated tasks. The gamelogic is finished for hanoi's tower, continued pair programming on the ticket page for the app.

21.03.2023 Meeting with Mentor

Team members present: All and Frode

Topic: Followup up on last week's issues

Summary: We went over the discussion we had with Vitensenteret about the payment systems, and the following online communication we have had with them. We gave updates on our designated issues.

22.03.2023 Work Session

Team members present: All

Topic: Work session

Summary: We continued work on our designated tasks. Working on fetching the blog posts from the database and the UI for the ticket page as well as the hanoi's tower game.

28.03.2023 Work Session

Team members present: All

Topic: Work session

Summary: We continued work on our designated tasks. We pushed Hanoi's Tower game to the main branch and merged that in, as well as working on payment solutions and cleanup.

21.03.2023 Meeting with Mentor

Team members present: Maling, Philip and Frode

Topic: General update

Summary: Again this meeting was spent going over recent developments and showcasing further work.

30.03.2023 Meeting with VI

Team members present: All and Gavin.

Topic: Discuss payment systems and showcase progress

Summary: This meeting we showcased our progress to Gavin. At this stage we have the functionality down in many areas, like the profile, tickets and games, as well as rudimentary blogs. We asked for some resources, mostly in regards to gaining access to Vitensenteret account at Vipps, as well as some assets for the games, like questions and solutions they want present in the games.

30.03.2023 Work Session

Team members present: All

Topic: Work session

Summary: We sent the required information to Vitensenteret to gain access to their Vipps account. Malin continued working on the admin page for the tickets, Philip worked on Vipps, and Susanne is writing documentation.

12.04.2023 Work Session

Team members present: All

Topic: Work session

Summary: Reprioritized some tasks and discussed our change of developmental methods.

18.04.2023 Work Session

Team members present: All

Topic: Work session

Summary: The games on the app are nearly finished, and we worked on attaching them to firebase.

18.04.2023 Meeting with mentor

Team members present: All and Frode

Topic: Update

Summary: We updated him on the current development with payment systems, where Vitensenteret asked us to use Stripe. In addition to this we showcased our games, which at this point are functionally done and had him test them.

20.04.2023 Work Session

Team members present: All

Topic: Work session

Summary: Started deploying in code testing, and developed payment systems for the app with a minor breakthrough in being able to call it to the phone.

25.04.2023 Work Session

Team members present: All

Topic: Work session

Summary: Working on blog functionality and admin pages. Still continuing development on payment systems.

25.04.2023 Meeting with Mentor

Team members present: All & Frode

Topic: Status meeting

Summary: Presented current progress and told mentor that it is unlikely we will get to publish the application due to a couple reasons, such as the indicieness of vitensenteret.

27.04.2023 Meeting with VI

Team members present: All & Gavin

Topic: Update on current application

Summary: The group presented the application to Gavin as it is and gave some insight to development. We agreed to leave the programming game as it is, and discussed the development of a user manual for the application such that any users of the application, in particular those who work at Vitensenteret will have an easy time looking anything up in the scenario in which they would need to.

We also went over how we will not be publishing this application, but offered further development options for Vitensenteret and they seemed pleased at this. Furthermore this documentation on the further development will be added into the project as one of the final chapters, as per the report requirements.

02.05.2023 Work Session

Team members present: All

Topic: Status meeting

Summary: We picked up some of the remaining code, aiming to finish it off. We managed to finish the blog and are very close with the rest of the code. We also

decided not to develop the notification systems, as it will be too much to set ourselves into, after our soft deadline to finish coding.

02.05.2023 Meeting with Mentor

Team members present: All & Frode

Topic: Status meeting

Summary: We showed the process since the last meeting, the blog functionality is working, and discussed the fact that we dropped notification systems. In truth we have added more than originally planned so the group feels secure in the amount of work done.

04.05.2023 Meeting with VI

Team members present: Malin, Philip and Gavin

Topic: Showcase App

Summary: We showed the application as it is now. The application was done, and no issues were found. Gavin tried out different functionalities and the flow in which he navigated was smooth without much issues. The application proved intuitive to use, and we received much positive feedback from Gavin.

10.05.2023 Internal Status Meeting

Team members present: All

Topic: Status meeting

Summary: We went over what code is still not finished and made a plan to finish what is needed. In addition to this we went over the progress of the rapport at this moment. Most of the structure is done, with the earlier segments fleshed out. As it stands we are about halfway with content, however we have quite a bit of text written elsewhere, so a bit of the work is simply putting the pre-written texts and snippets where they fit into the final rapport. We are aiming to have a more substantial piece by Friday so that we can send it to our mentor to have him review it.

12.05.2023 Work Session

Team members present: All

Topic: Work Session

Summary: Continued development to get those final small things together, as well as catching everyone up on the report so far.

15.05.2023 Meeting with Mentor

Team members present: All & Frode

Topic: Feedback on report at this time

Summary: We delivered our report as it is at this time and got some feedback on how to improve it. Most of it was structure wise.

15.05.2023 Work Session

Team members present: All

Topic: Work on Report

Summary: We worked on finalizing the code, making it more readable. In addition to this we worked on the report.

19.05.2023 Work Session

Team members present: All

Topic: Work on Final Report

Summary: Today we worked through our to-do list and went over everything that is still unfinished, at this time we are on route to be done. Most of the content is done, although, some is still missing. We agreed to have a meeting on Sunday to go over the thesis together before we deliver.

21.05.2023 Work Session

Team members present: All

Topic: Delivery

Summary: We read over the report and made some final edits. After this we compiled everything together using the resources NTNU specified to fit the title screen etc and uploaded it to inspera.

H Time Chart

Color:	Explanation:
	Research
	Report Writing/Documentation
	Testing
	Coding
	Meetings
	Planning
	Sick or Unavailable

Malin Foss								
Week:	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total:
2								6.5
3	4 (Project plan)	6.5 (Research and meeting)	2 (Seminar)		3 1.5 (Project plan)			17
4	1.5 (Project plan)	5 (Report writing and meeting)	6 (Flutter, writing req spec)	4 (Flutter)	4 (Flutter, GitLab CI, prototype)		1	20.5
5	1 (req spec)	4 (GitLab CI, flutter)	4 (Flutter, 2FA, gitlab CI)	4 (Group and VI)				13
6	1 (Firebase)	1 (Sprint meetings, firebase)	1.5 (firebase)	5 (issues 6 and 10, meeting)	3 (issue 10)			11.5
7	6 (Issue 10)	3 (Sprint and mentor meeting)	3 (Issue 18)		2 (for issue 18)			14
8	5 (Issues 18 and 19)	5 (Meetings and Issue 21)	1 (Issue 21)	6 (Issues 20, 21, 22)			1.5 (Issue 21)	19.5
9	2.5 (Issue 21, meeting)	5 (Meetings, lecture, etc.)	2 (Issue 21)	7 (Issue 21, meeting with VI)				16.5
10	4 (Issues 21, and 22)	6 (21 and 22)	6 (Issue 20 and 21)	6 (Meeting with VI, testing, issue 20)	1 (Issue 25)			23
11	3 (Duel)	6 (Issue 20)	6 (Issue 20)	6 (Issue 20)	6 (Firebase)			27
12	2 (Issue 20)	4 (Issue 20)	6 (Issue 26)	4 (Issue 26, research database rules)	3 (Issue 26)			19
13	3 (Issue 29)	6 (Issue 30)	5 (Issue 29)	5.5 (Issue 30)	4 (Issue 29)			23.5
14								0
15		6 (Issue 29)	6 (Issue 30)	6 (Issue 30)	4 (Issue 30)			22
16		5 (Planning, Issue 29)		7 (deployment, issue 35)			3 (Issue 30)	15
17	3 (Issue 30)	4 (general fixes)	3 (Issue 29)				1.5 (Issue 29)	11.5
18	4 (Issue 29)	5 (General fixes)	5 (General fixes)	5 (Vitensenteret, gruppetote)				17
19	4 (final report)	3 (final report)	4 (final report)	5 (final report, Issue 41)	6 (final report)			22
20	4 (Issue 46)	4 (final report)	6 (final report)	4.5 (final report, user manual)	5.5 (final report)	6 (final report)	5 (final report)	35
								333.5

Total spent on topic:	
	48,5
	61
	6
	174
	40
	4
Unavailable is measured in days	
	10

Philip Morud								
Week:	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total:
2			2 (Seminar)	3 (Planning)				5
3		5 (Research and meeting)	5 (Research and meeting)	2 (Research)	1 (meeting with Employer)			13
4		3 (Research and meeting)	5 (Research)	5 (Flutter)	3 (Flutter)	2 (Flutter)		18
5	3 (Flutter)	5 (Research)	5 (Git and flutter setup)	4 (Group and VI)	3 (Flutter)			20
6		2 (Sprint)		3 (Meeting and research)	2 (Flutter)	3 (Issue 9)		10
7	2 (Issue 9)	5 (Vipps solutions)	4 (Vipps, Blog and Status)	3 (Issue 17)	3 (Issue 17)			17
8	3 (Issue 17)	5 (Issue 17)	3 (Vipps)	4 (Issue 17)	5 (Issue 17)			20
9	3 (Meeting about Vipps)	4 (Vipps research and meeting)		5 (Issue 17 and meeting)	4 (Issue 17)			16
10		4 (Issue 17)	4 (Issue 17)	6 (Meeting with VI, Error fixing)	3 (Issue 23)	3 (Issue 23)	2 (Issue 23)	22
11	2 (Issue 23)	2 (Issue 23)	4 (Issue 23)	7 (Issue 20)				15
12		4 (Issue 20)	6 (Issue 20)	6 (Issue 20)	2 (Vipps)		2 (Vipps)	20
13		5 (Issue 31 and bugfixing)	5 (Issue 23)	5 (Issue 23)				15
14								0
15		3 (Stripe payment system)	6 (Stripe payment)	5 (Stripe payment)				14
16		6 (Stripe)	4 (Stripe)	5 (Stripe)	2 (Issue 31)			17
17	4 (Issue 31)	6 (Issue 31)						10
18	4 (Issue 31)	7 (Issue 31)	11 (Issue 31)	5 (Improved blog page)	2 (Issue 31)	2 (Issue 31)	2 (Issue 31)	33
19	5 (Issue 31)	4 (Issue 31)	6 (Issue 31)	4 (Issue 31)	6 (Issue 31, 34 and 45)			25
20	5 (Issue 46)	4 (Issue 46)	4 (Final Report)	6 (Final Report)	8 (Final Report)	10 (Final Report)	5 (Final Report)	42
								332

Total spent on topic:	
	88
	33
	0
	167
	41
	3
Unavailable is measured in days	
	8

Susanne Skjold Edvardsen								
Week:	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total:
2			2 (Seminar)	3 (Planning)				5
3	3 (Project plan)	5 (Research and meeting)	3 (Project plan)				3 (Design)	14
4		5 (Report writing and meeting)	5 (Research)	4 (Flutter)	5 (Flutter)			19

I Wishes from Employer

Følgende er listen med ønsker som vi fikk fra bedriften i november da vi besøkte dem.

VI App

Innhold

-Info -Vitenskapelig prinsipper relatert til utstillingene -Spill - tankenøtter, kan makecode legges i? Lage egne spill? Gjør appen gøy å ha/bruke for voksne og små. -"Hva skjer"
- push varsling. Eventer og rabatterte dager osv... Vi må ha mulighet til å sende de ut.
-Årskort - mulighet for å ha det i appen. Bilde av personen. Må være mulig å vise det på en pc, f.eks. ved glemt telefon. Utgåtte årskort må være synliggjort. -Bluetooth beacons / QR koder på senteret?? Forklaring av installasjonene. Kan det brukes for å ha flere språk på appen for utenlandske besøkende? -Forhåndskjøp av billetter? Visit Innlandet... Kjøp av årskort? Kjøp av årskort som gave. Hvordan løses det? Kan vi produsere QR koder som kan gis i kort som kan trigge et ferdig betalt innlogging til årskort løsningen?
-Astronomi? Hvilke stjerner, planeter er synlige nå? -Bestilling av pizza(Pizzabakeren) Gjennom appen? -Parkering(Easypark) gjennom appen? -Sjekkliste? Har du fått med deg alle utstillingene ved å skanne QR koden? -Designhåndbok for Vitensenteret

Fremdrift/Sikkerhet

-Det må utarbeides en fremdrifts plan med tanke på å vedlikeholde appen. Hvem gjør det? Sikkerhet? Oppdateringer? -Sikre databasen. Bruke Firebase?

J User tests

User Tests

The following is a user test designed to test the application that has been developed by our team. The application was developed as such following a walkthrough of a paper model with the employer at Vitensenteret, and the feedback from there has been taken into account when creating the application that this user test will test.

Testers:

Firstly about our testers. All testers were contacted by the group and consist of a variety of people, including boomers and employed IT personel. They were all informed that answering and performing this tests will let us store their age and occupation. Made up names have been used to comply with the GDPR.

The tests and questions:

Start up the application and turn it to the user, to begin with, no user will be signed in, and they will have the basic knowledge that the application they are testing is developed for Vitensenteret, and you can get news as well as the ability to purchase tickets for entry.

Allow the tester to familiarize themselves with the application before giving them any questions. The user decides when they are ready to commence.

Ask the user to see what the most recent news are in relation to vitensenteret.

Ask the user to see what tickets they have (at this point they will discover they need to log in, in which the questionnaire will provide credentials).

At this point, we will make the user an administrator. Let them familiarize themselves if they wish before commencing.

Ask the user to make a blog post.

Ask the user to find a specific person by the name of Tone.

At this point, functionality has been tested, the following questions are in regards to the feel of the application and ease of its use.

1. Was the application easy to navigate? Was there at any point a time where you did not know where to go?
2. Did you find something in the application that you did not expect to find? Did you notice or try any of the games?
3. Do you feel the admin pages flow naturally and have any functionality that may be needed?
4. Would you download this app if you were a frequent user of Vitensenteret? And if you would only download it for the ticket system, do you feel that the games included in the app makes it more likely for you to keep the app?
5. Anything else you would like to add?

Tester: Askeladden

Name: Askeladden

Age: 58

Occupation: CEO of own company

Time to familiarize themselves: 16.45 minutes

Time to perform normal tasks: 36 seconds

Time to familiarize themselves with the admin page: 2.12 minutes

Time to perform admin tasks: 4.07 minutes

Answers to questions:

1. The app was easy to navigate. Although a manual or something akin to it would be nice for the admin tasks.
2. Nothing out of the ordinary. I played all the games, but they were very difficult, more hints would be appreciated.
3. Could not see if a ticket was added despite there being a very obvious ticket adding button.
4. The games added some flavor, but the tickets would have been enough to keep the app downloaded.
5. The games were fun, especially once you understood how to play them.

Any other comments about the performance of the tester:

The tester took a long while to familiarize themselves with the test to begin with, including playing all the games and trying multiple of the "tasks" that they were intended to perform later. This made their time faster than anticipated, however it shows how easy the app is to use, once it has been learned.

Tester: Gulli

Name: Gulli

Age: 61

Occupation: Senior advisor.

Time to familiarize themselves: 1.22 minutes

Time to perform normal tasks: 1,05 minutes

Time to familiarize themselves with the admin page: 1 second

Time to perform admin tasks: 2,27minutes

Answers to questions:

1. There was a bit of a disconnect between what the tester expected to do when undergoing the task, as they expected to purchase a ticket and not perform any of the admin tasks.
2. The wording of the tasks were a little confusing in regards to the word "blog" as the tester was confused to how they could make such a publication. The tester did enter one of the games, but swiftly left, and did not think much around it. But in reflections mentioned that the games are a nice add-on.
3. The admin pages were nice.
4. The tester would download the app.
5. It was very neat and the overview is very nice. If you are not sure if you want to go to vitensenteret, downloading the app might convince you to travel and experience vitensenteret. You could change your profile and such.

Any other comments about the performance of the tester:

The tester was swift at their tasks, however got a little stuck trying to make a blog post, could not find the button as it is partially hidden behind a debug banner.

Tester: Hjorten

Name: Hjorten

Age: 65

Occupation: Advicer

Time to familiarize themselves: 15,56 minutes

Time to perform normal tasks: 1,32 minutes

Time to familiarize themselves with the admin page: 1 second

Time to perform admin tasks: 3.44 minutes

Answers to questions:

1. The app was easy to navigate.
2. The games were there. They were fun.
3. The admin pages flowed naturally.
4. The tester would download the app.
5. The applikation is nice to have so that you get a connection to the location through the app. It is possible that you would visit the location more often because of the app. Maybe through new games or blog posts.

Any other comments about the performance of the tester:

The tester took their sweet time learning the application before venturing on with the tasks. This led to them discovering and playing the games, as well as getting a good feel for the application. Later on when they were performing the tasks, the tester got caught up with the specifics of what to write in the blog post and spent some time on that.

Tester: PkmnFan

Name: PkmnFan

Age: 30

Occupation: part time student, part time employed.

Time to familiarize themselves: 8.12 minutes

Time to perform normal tasks: 35 seconds

Time to familiarize themselves with the admin page: 4.26 minutes

Time to perform admin tasks: 40 seconds.

Answers to questions:

1. Overall the app was alright to navigate but the usage of red for the admin pages for the back button was not the right choice. Should instead have a back button top left. It should be more apparent how to get to the admin blog page.
2. Didn't expect to find games and were pleasantly surprised. It can be an incentive to download the app, especially since there won't be any ads ruining the game experience.
3. The buttons on the admin blog page were not nice to look at. Very easy to make a mistake when handling tickets.
4. Yes. Especially for buying tickets, so you don't have to interact with anyone.
5. Perhaps check what's in in regards to style, and update the app accordingly.

Any other comments about the performance of the tester: The tester took some time getting to know the app, and when doing the tasks did them very fast. They played through the games, and created tickets, users and blog posts.

Tester: FlyingEagle10000

Name: FlyingEagle10000

Age: 24

Occupation: part time IT student, part time employed as IT support.

Time to familiarize themselves: 1.22 minutes

Time to perform normal tasks: 15 seconds

Time to familiarize themselves with the admin page: 36 seconds

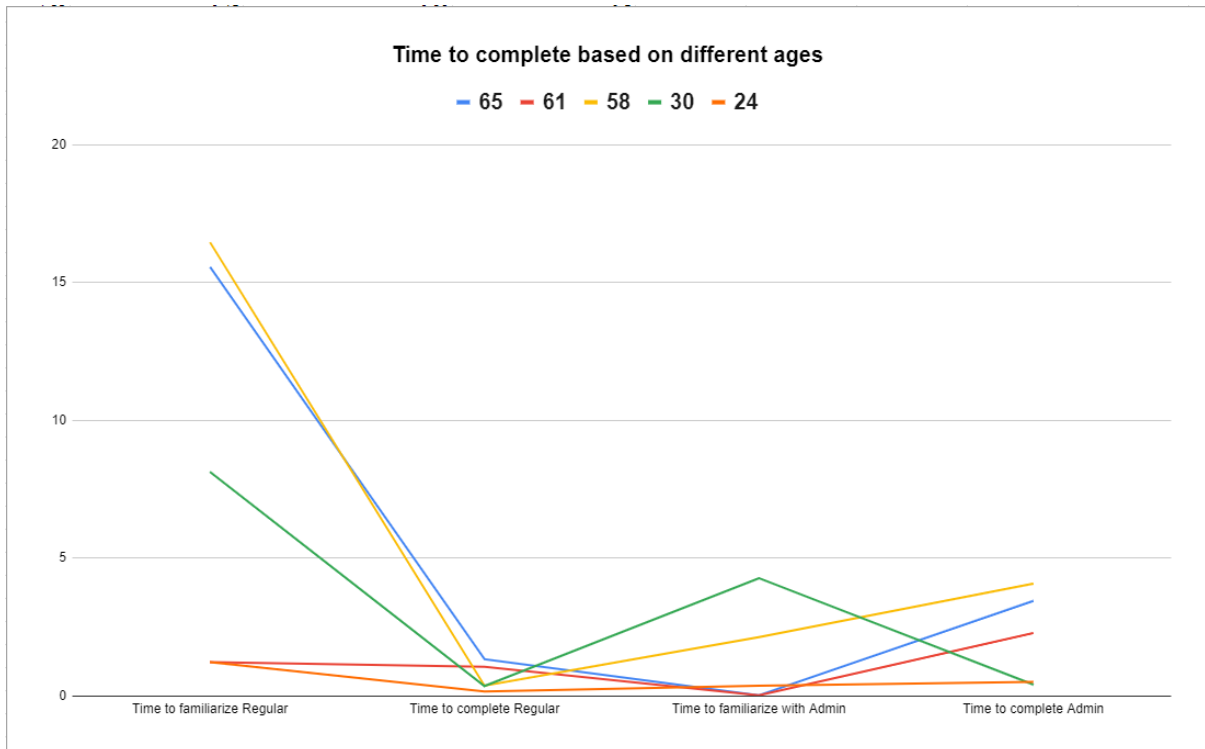
Time to perform admin tasks: 50 seconds

Answers to questions:

1. Due to the letters being small, it was hard to make out details in the blog posts. They remarked finding the button to click to get to the admin page was difficult.
2. No, error message when logging in, if credentials are wrong. Expresses devastation that they didn't try the games.
3. Intuitive to use, however the tester has a background in administrative IT.
4. No, they don't bother to download apps often. They don't want to be distracted by games on the phone.
5. Fine.

Any other comments about the performance of the tester: The tester focused on doing it as fast as possible, and missed out on some of the details, But their speed showed that the app was intuitive to navigate and learn for someone who has never used it before. In addition to this, this tester has poor eyesight and struggled to see some of the letters in the blog post, which they commented upon above.

Chart based on performance, with age as the defining feature.



K Group rules

Rules of the Group

- Each member of the group should aim to work for approximately 30 hours per week. In the case where the total is repetitively below 30 hours, a meeting will be held amongst the group members to discuss causes and solutions.
- The group will host one physical meeting each week where it is expected that all members are present. Should a member be unable to attend this meeting, they should notify the other members of the reason in advance. (If all the group members agree, the meetings can be scheduled less often, for instance biweekly).
- All the hours spent working towards this project should be written down. These time logs should be available in the final delivery.
- Each member is responsible for showing up to scheduled meetings with the group, the mentor and the employer.
- Group members are obligated to warn the group if they believe themselves, another group member, or the group as a whole is starting to fall behind so measures can be taken to avoid losing too much time overall.
- In the case of disagreements within the group there are three levels of measures to be taken. All positives and negatives are to be weighed together in a discussion, if no decision is reached the mentor can be involved to get some outside perspective. Last resort is putting it to a vote.
- If a group member breaks several rules or repeatedly breaks rules it has to be brought up and escalated in the following order:
 - Discussion with the entire group
 - Written warning with cause, measures to fix the issue(s) and potential consequences
 - Conversation with mentor and the entire group
 - Written exclusion from the group (no later than four weeks before the delivery date)

Date:

18. januar 2023

Signed:

Malin Foss

Malin Foss

Philip Morud

Philip Morud

Susanne S. Edvardsen

Susanne Skjold Edvardsen

L Requirements specification

Requirement Specifications

Functional requirements:

- Buy annual pass (Admins can manage these)
- Read info about Vitensenteret
- Register/log in
- Play a game

Summary:

The main purpose of the app is to serve as a ticket purchasing system, the user should be able to purchase yearly memberships. For this to be possible users should also be able to create a user and log in to view their tickets. With or without logging in users should be able to read news from Vitensenteret in a blog format, play brain teaser games, receive notifications from the app and change app settings. An admin user should be able to do all this, with the addition of being able to make blog posts and notifications. In addition to this, an admin shall be able to manage purchased tickets from other members, like removing, adding or assigning them.

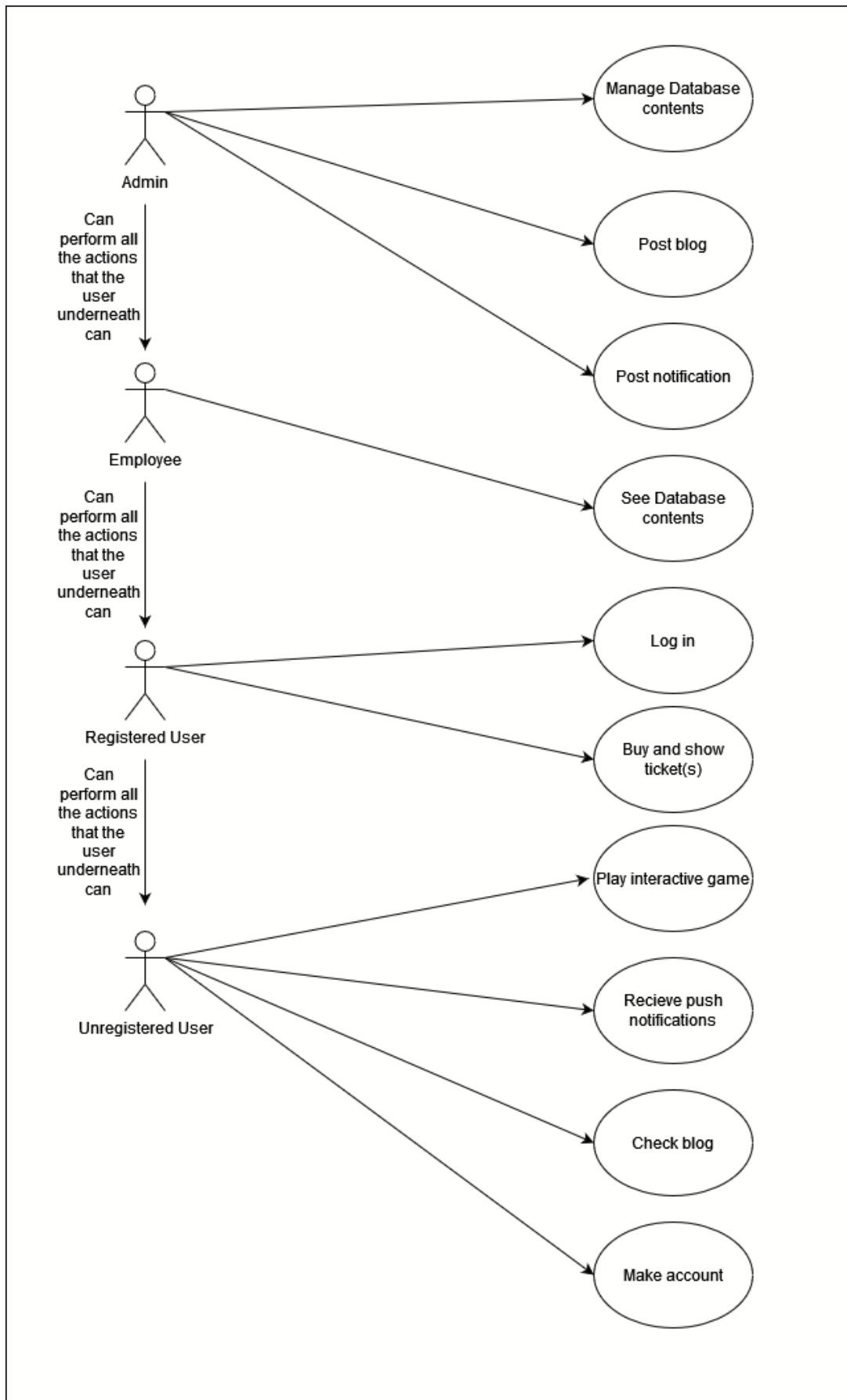
Frontend:

To use this application we are developing an application that can be downloaded for phones, both android and apple devices. Additionally we are making an interface that is compatible with a desktop so that those who are working at Vitensenteret can use their computers to gain access to the application, as well as the more administrative parts of the page.

Backend:

The application will be connected to a database in firebase, as well as having general updates and push notifications through this system. Flutterfier is the interface between flutter, the programming language that the group is utilizing to develop the app, and firebase.

Use Cases:



Use Case Descriptions:

Action:	Post blog
Actor:	Admin
Goal:	Make a publication in the form of a blog
Description:	The admin navigates to their profile icon, and taps it. This will open an additional menu, where options, tickets etc are available. On this menu there is an option to create a blog post, which the admin can press. They will then be presented with a simple blog creation page, where they input the title, content, and potentially image with the blog. Additionally there is an option to create a notification at the publication of the blog, which the admin can cross off on a yes or no based on the situation.

Action:	Manage Database Contents
Actor:	Admin
Goal:	Remove Post from database
Description:	The admin will be on the desktop version of the app. They will navigate to the button with posts "Innlegg". Here they will be presented with the option to search for the post they wish to remove. After searching they are presented with a list of posts that match the search criteria. Selecting the post that is the target, the admin is prompted to hide it from the public, or outright delete it. The admin deletes the post.

Action:	Make account
Actor:	Unregistered User
Goal:	Create a new account with personal identification
Prerequisite:	The user does not already have an account.
Aftermath:	The user has made an account.
Description, normal flow:	The user clicks on the “register user” button and is brought to a page where they are prompted to input a profile picture, name, email and password. If the input is invalid the user will be prompted again. When the user has entered valid data, a new user is created in the database and the user is automatically logged in.
Description, alternative flow:	In the case where the user is logged into an account they dont want to have, they will navigate to their usertab on the right. This will open the profile of the user, where there is a button to log out. After this is completed, the user can follow the normal flow description.
Error situations:	Errors that may happen, or stop the progression is if the user has entered credentials that is registered to another user, or if they mistype something. This includes the likes of: email, name, password, or picture is the wrong format. Another error that may stop this is if the user has a poor connection to the database.

Action:	Log in
Actor:	Registered User

Goal:	Log in on the registered account
Prerequisite:	The user has an account registered to them already.
Aftermath:	The user has logged into the app with their credentials.
Description, normal flow:	The user clicks a "Login" button and is brought to a page where they are prompted for email and password. If either email or password are invalid the user will be informed of such, and will have to reenter valid information. After they have logged in they are brought to the blog and the icon showcasing the profile will be updated with their profile image.
Description, alternative flow:	If the user does not have an account, they will have to create one, following the use case example: Make account.
Error situations:	<p>Errors that may happen, or stop the progression is if the user has entered credentials that is registered to another user, or if they mistype something. This includes the likes of: email, name, password, or picture is the wrong format.</p> <p>Another error that may stop this is if the user has a poor connection to the database.</p>

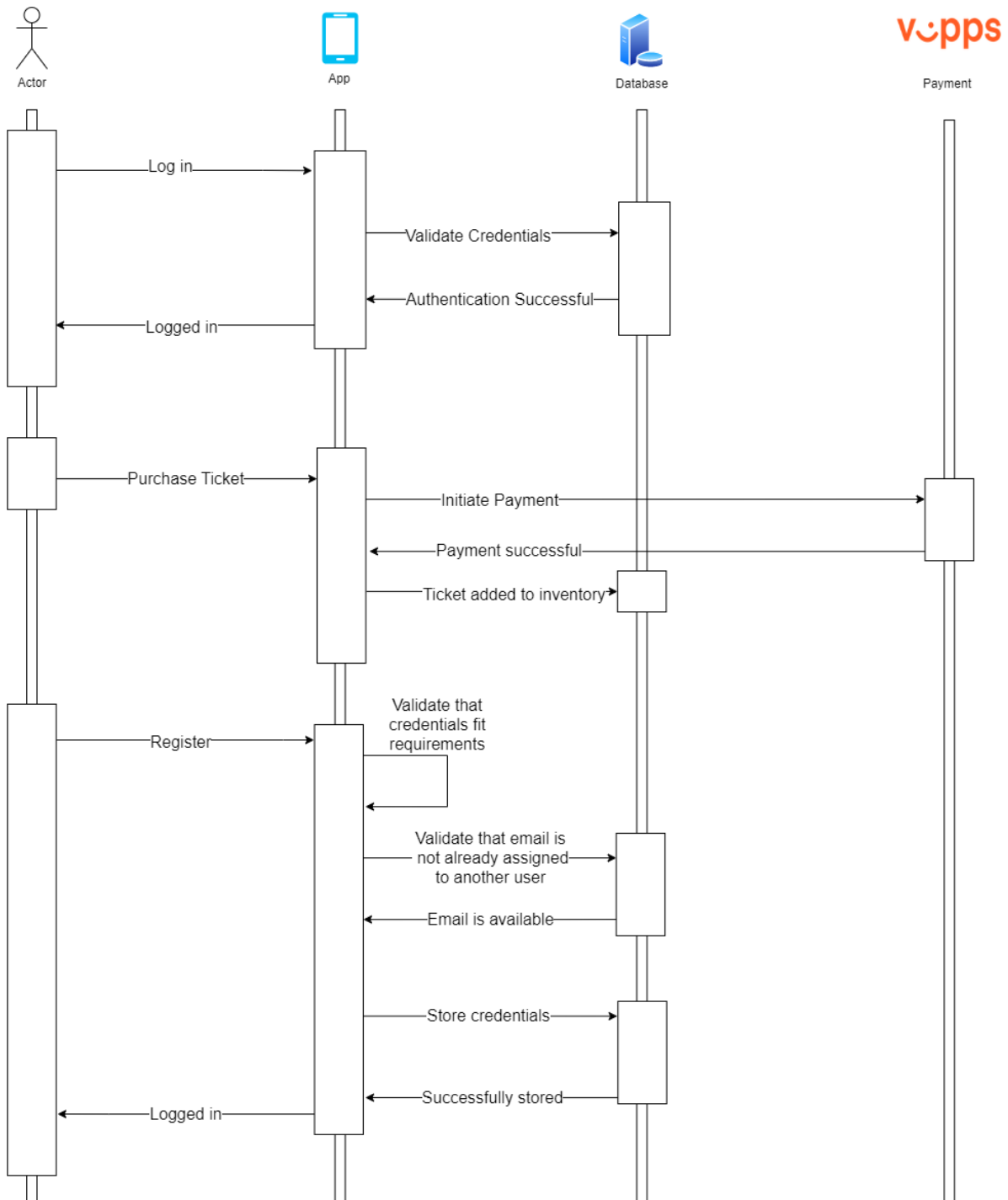
Action:	Receive push notification
Actor:	All Users have access to this action
Goal:	Receive an update from Vitensenteret
Prerequisite:	Their phone is turned on, connected to some sort of internet and has the app downloaded.
Aftermath:	The user has a notification on their phone from the app.

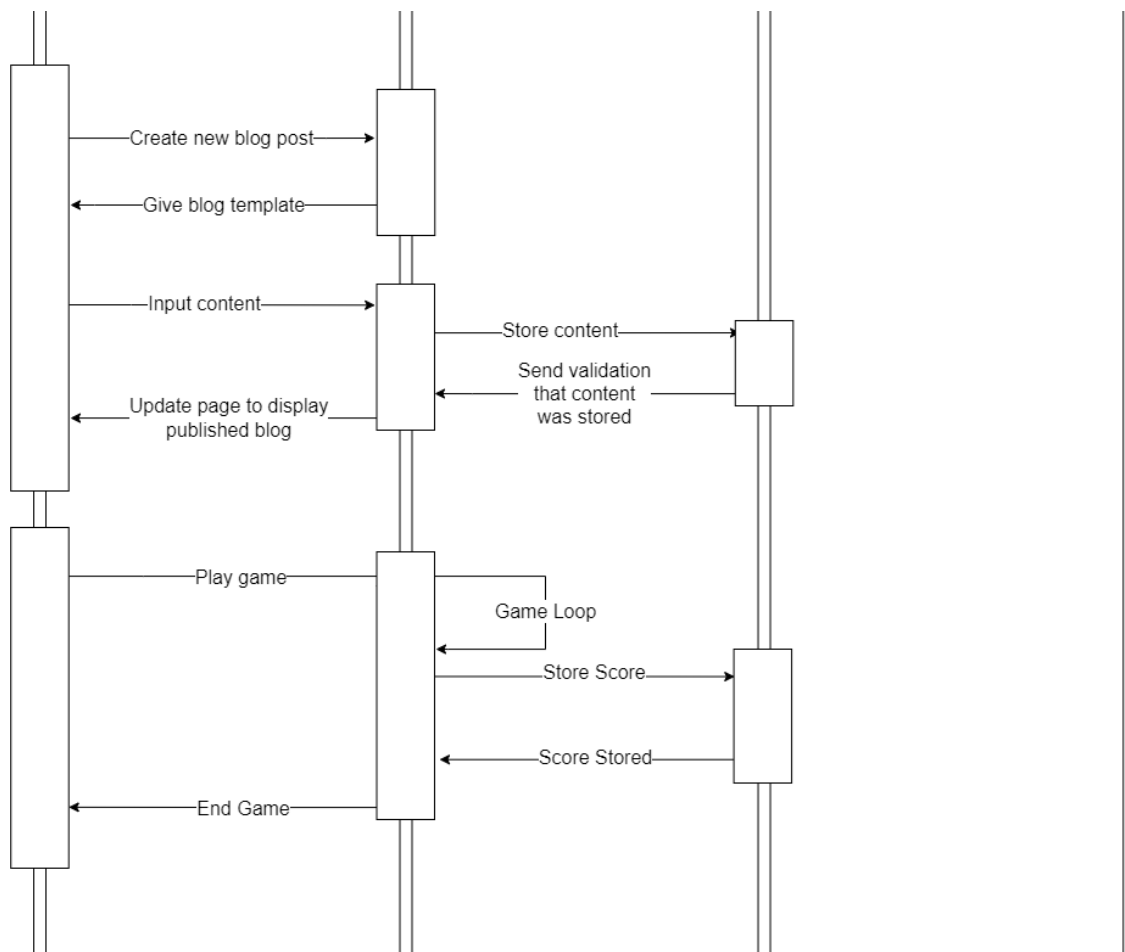
Description, normal flow:	While the user is not actively using the application they receive a notification from the app, which may prompt them to open the app to gain access to more information through the blog.
Description, alternative flow:	In the case where a notification is not present on the phone, the only way to receive the update is to go into the app, and go to the blog page, this page is also the homepage. Here the notification will be displayed as a blog post.
Error situations:	The notification does not appear, or the content of the notification is wrong. Another error that may stop this is if the user has a poor connection to the database.

Action:	Buy Ticket
Actor:	Registered User
Goal:	Purchase a yearly ticket
Prerequisite:	The user is logged in, and has some way or means to purchase the ticket.
Aftermath:	The user now has a yearly ticket in their name, on their phone.
Description, normal flow:	The user navigates to the ticket button. They will then be taken to a page which showcases any purchased tickets, if there is any to showcase. There is then a plus icon they can press next to the tickets. Pressing this button slides up a new menu with the options "Buy Ticket" or "Use Code". In this case the user navigates to "Buy Ticket" and is sent to another page. They are prompted with the option of "day ticket" or "Yearly Ticket" and select the later option. The price will be displayed and underneath the price will be an option to make this ticket a gift. Without pressing the gift button the user navigates down to see

	<p>that there are multiple payment options, they choose Vipps, and are then forwarded to that application to finish the payment. Once that has been processed the ticket page will show the updated yearly ticket that was purchased.</p>
Description, alternative flow:	<p>The user goes to buy a ticket by pressing the ticket button only to discover they already have one. So they dont need to buy another.</p>
Error situations:	<p>The user chooses the wrong kind of ticket, or the wrong kind of payment method and has to contact Vitensenteret to get it resolved.</p> <p>The user chooses the ticket to be a gift, even tho it isnt one.</p> <p>The payment handing was flawed and the payment was not processed.</p> <p>Error with the connection to the database, or lacking internet.</p>

Sequence diagram:





Product backlog:

The product backlog is presented physically at all the scrum sessions. This backlog is created and maintained on a reMarkable.

On this reMarkable, the group notes down all elements on the backlog and updates it accordingly on each scrum. All the tasks that are relevant for the coding aspects of the project, for instance developing registration or working with databases are subdivided and added into the GitLab as Issues to be referenced with each commit to the repository.

This is the content of the backlog as of 01.02.23:

Administrative:

- Requirement specification
- Status report x3 to mentor
- Final report
- Licence for publishing to App Store and Google Play Store

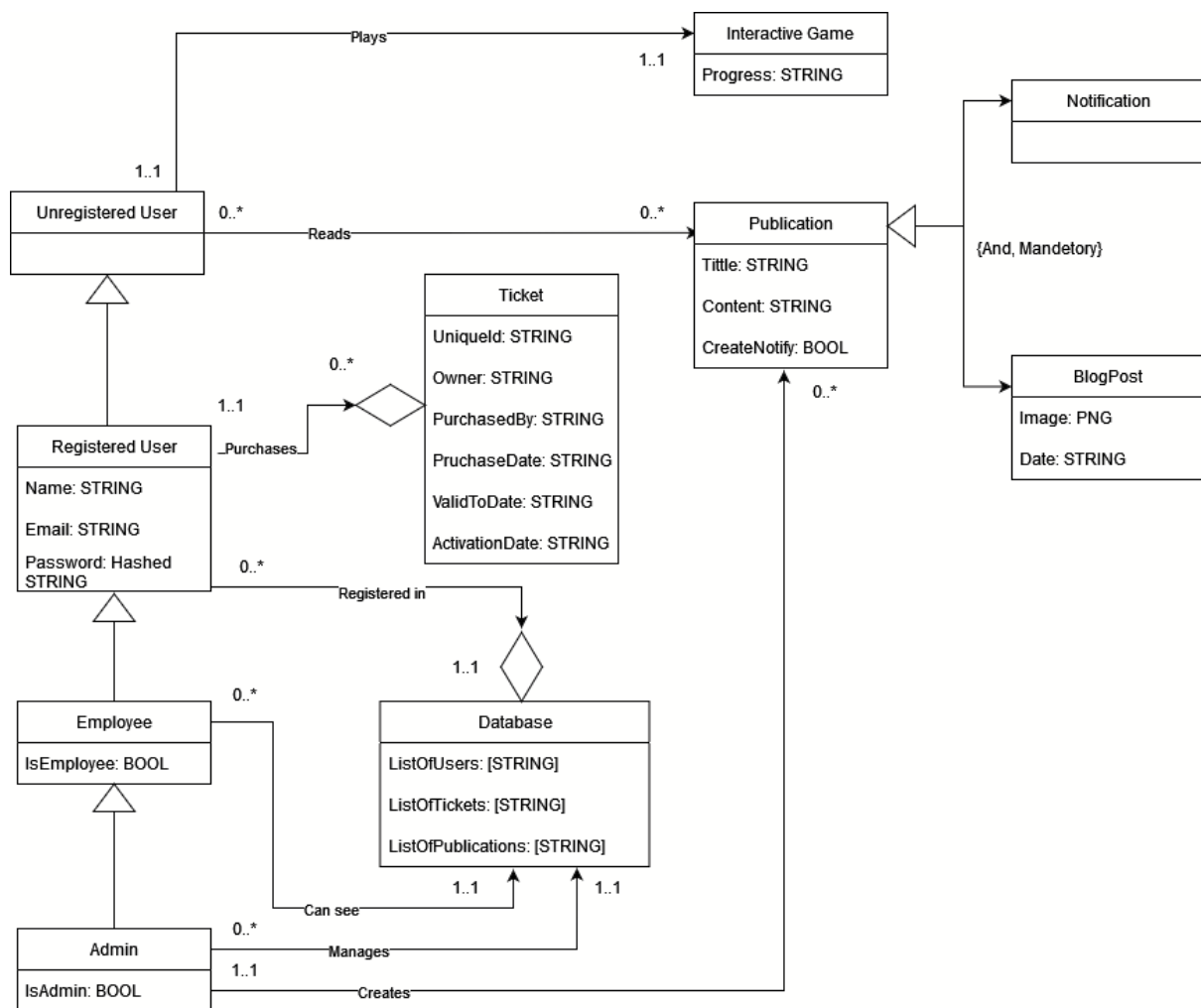
App:

- Login
- Registration
- Create blog entry
- Display blog entries
- Delete blog entries
- Payment system
 - Vipps
 - Visa
- Gift code input
- Bottom bar
- Profile page
- Settings
- Game page

Other:

- Gain access to database
- Backend connection to database
- GitLab CI pipeline

Domain model:



This domain model was created to show how everything is connected in the application. The entrypoint used in this brief explanation will be the unregistered user. We see that this kind of user has no information stored on themselves, and this is meant to represent users who may only use the application to read blogs, play the game, or receive notifications. The unregistered user can then register their information and become the registered user, which inherits all possible actions from the unregistered user. This user also unlocks the ability to purchase tickets in their name. Furthermore there are two additional users, this is the employee and the admin. Employees can see the contents of the database, like tickets purchased, and their owners, but admins have the ability to manage this database. Admins have the ability to manage tickets that have been purchased, this includes removing a ticket from a user, or giving them one, for instance in the case where a user may have accidentally used an activation code on themselves instead of gifting it. The admin is

also able to upload publications, which can either be a simple notification or a blog. In some cases the creation of a blog may also create a notification.

Operational requirements:

- Android API level 31
- iOS 15
- Windows 10
- macOS Catalina
- Firebase

The app is being developed for both Android and iOS. For Android the target API is 31, the current standard target API for Android apps (Google, *Unknown*). For iOS we aim for iOS 15, which is currently the version in use by 89% of all iOS devices on the App Store (Apple, *Unknown*). We will also develop the administrative part of the application for desktops only used by employees at Vitensenteret Innlandet. For this we will use a minimum of Windows 10 and macOS Catalina (10.15). The database in use will be the existing Firebase database, which will keep user data as well as ticket data.

Safety and misuse handling:

- User data has to be safely stored
- Safe to buy a ticket
- Hashing and encrypting
- Tickets can only be used by the owner or gift receiver
- Different security levels for users and employees at Vitensenteret

Since the app will be handling sensitive user data it has to follow standards and laws around that, like GDPR. All passwords have to be properly hashed and communications with the database encrypted. This will ensure that the users and their tickets are safe. Tickets and gift codes should be unique and only able to exist one place at a time, otherwise duplication and theft can happen. It should not be possible to use someone else's ticket to gain access to Vitensenteret. Buying tickets should be safe, and we will use existing systems for this. To ensure the integrity of the app, access to features will be restricted by security levels. At least four security levels are necessary; unregistered user, registered user, employee, and admin.

Testing

After each issue is completed, it is comprehensively tested by the one who implemented the solution. Each developer has their own branch on the git repository, and each scrum review, the group gets together to go over what has been implemented and then push the changes to the main branch.

In addition to this there are some testing pipelines. TBA malin.foss00@gmail.com

Licensing

The product will be developed by us, the student group at NTNU as a bachelor's thesis. The contract that we signed with Vitensenteret (**Add which appendix the contract is**) states that we retain no rights to ownership of the final product, and can only use the content that we created as a way to show our skills to potential future employers. This means that Vitensenteret has all rights to the software that we develop during this course, for this thesis.

Version updates

At the point in which we deliver our product and software to Vitensenteret and launch it, we will no longer be responsible for the updrift of this software. Therein lies the ownership transfer to Vitensenteret at the core of the agreement as with it comes the responsibility to update the software and keep it relevant for the users at Vitensenteret.

To make this task easier for Vitensenteret the code is written in modular steps which makes it easy for other developers to continue work on the software. Our design choices reflect this as well, as we deigned to have most other options like settings, profile and such in another more expandable menu.

We have also written the code with this in mind, and as such it has been commented and explained at crucial points in the code, what the different sections do. In addition to this, Vitensenteret will gain access to all our developmental documents, like the

domain model, use case diagrams and furthermore can expand on these as they see fit.

Storing personal data in accordance with GDPR

In the year 2018, the European Data Protection Regulation was applicable to all of the members of the European Union (Intersoft Consulting, 2018). This General data Protection Regulation, GDPR for short, ensures that the data stored on individual people is not in any way used for malicious intents.

In the app that is being developed by us for Vitensenteret, it is a necessity to store certain details about its members to obtain some functions of the app. Most of the app can be accessed without the application storing anything about the user. What is stored about the user, with the user not being a member of the app, and signed in, are as follows:

- Progression or streak in the games app.
- If the app is idle, then the app will store in its cache what page the user was most recently using, before leaving the app in its idle mode.

In relation to the definitions specified by the GDPR, personal data is the following: *“‘personal data’ means any information relating to an identified or identifiable natural person (‘data subject’); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;”* (Intersoft Consulting, 2018, Art. 4).

Abstracting from this, we can say that the data stored about unregistered users are not personal data, as the data cannot in any means recreate the persona they originated from. Therefore this data can be stored without consideration to the GDPR.

However, in the instance where a user will need to log into the account to gain access to the other functionalities of the application, we will need to store the following about the user:

- Name of the user.
- Email of the user.
- Password of the user.

The reason why the app needs to store this data, is because the app will provide a paid service, and to ensure that only the user who paid for their ticket is using it, we

need to store this information about the user. And as by the definition by the GDPR above, this clearly qualifies as personal data. This data, by the rules of the GDPR needs to be the following:

- The user will be made aware that by creating an account they consent to having this data stored about themselves.
- The user will be made aware that by deleting an account their personal data will be wiped from the databases, and is unrecoverable.
- The data will be stored in secure locations, and sensitive data, like the password, will be hashed.
- The data will not be processed in any other way than to ensure that the user is who they claim to be.

Other principles in regards to the GDPR can be read following the sources to the Regulation, however, the points listed above are those that have a concrete and important factor in the development of this app.

Interface requirements:

- Easy to navigate, even for younger users
- Has to follow the design guide from VI
- Tickets should appear both for user and in the database

The design of the application will follow the design conventions as stated by Vitensenteret themselves. In the early development of the application, the group received a pamphlet with these design conventions, and it can be found in the appendix.

The app itself has been designed for the most common use cases of the app, buying tickets, seeing the blog and playing the little game. Other functionalities lay easily accessible in a slide out menu. All these functionalities are placed in a bar on the bottom of the screen for ease and reach. The design follows the common “Z” pattern(Nick Babich, 2017), with an attention grab at the top, a scan of the middle, and ending up at the easily accessible lower bar. For the blog itself, we utilised the “F” pattern (Nick Babich, 2017) instead, as that is better suited for a more text centralised design.

Sources(For the requirement specification):

Google (Unknown). Fetched from:

<https://support.google.com/googleplay/android-developer/answer/11926878?hl=en> (Date 24.01.23)

Apple (Unknown). Fetched from:

<https://developer.apple.com/support/app-store/> (Date 24.01.23)

Nick Babich (Jun 16, 2017) *Z-Shaped Pattern For Reading Web Content*. Fetched from:

<https://uxplanet.org/z-shaped-pattern-for-reading-web-content-ce1135f92f1c> (Date 31.01.23)

Nick Babich (Apr 5, 2017) *F-Shaped Pattern for Reading Content*. Fetched from:

<https://uxplanet.org/f-shaped-pattern-for-reading-content-80af79cd3394> (Date 31.01.23)

Intersoft Consulting (2018) *General Data Protection Regulation GDPR* Fetched from:

<https://gdpr-info.eu/> (07.02.2023)

M Translated summary

Tittel: Viten i App / Knowledge in App / Knowledge in an App

Dato: TBA

Deltakere: Malin Foss, Philip Morud, Susanne Skjold Edvardsen

Mentorer: Tom Røise og Frode Haug

Oppdragsgiver: Vitensenteret Gjøvik

Nøkkelord: Programmering, mobilapplikasjon, Flutter, API

Antall sider: TBA

Antall vedlegg: TBA

Tilgjengelighet: Åpen

Sammendrag: Vitensenteret i Gjøvik, et høyprofilert vitenskapssenter, tok kontakt med NTNU med et ønske om å gjøre noen av deres tilbud tilgjengelig på nettet. For å gjennomføre dette ble gruppen enig om å utvikle en mobilapplikasjon **sammen** med et grensesnitt som kan bli brukt på enhver datamaskin. Denne mobilapplikasjonen inneholder et billettsystem, en blogg og en samling av noen enkle spill. Appen har fokus på en god brukeropplevelse og i tillegg gir **internett grensesnittet/grensesnittet** administrative muligheter, som publisering og håndtering av innholdet i appen og **administrering/behandling** av billetter.

Løsningen er en applikasjon **skrevet/programmert** i Flutter, i Android Studio, koblet til en database i Firebase. Denne databasen bruker både lagringsmuligheter og **en database for lagring/datalagring** i sanntid. I tillegg bruker applikasjonen en API som betalingsløsning. Appen har ikke blitt gjort tilgjengelig på nettet ennå, **men lite arbeid må til for å publisere den slik den er.**

I løpet av utviklingsprosessen for denne appen ble det satt stort fokus på et profesjonelt arbeidsmiljø. Vi brukte Scrum, Kanban og Git «issue boards» for å ha oversikt over alle arbeidsoppgaver, og vi skrev sammendrag fra alle møtene gruppen **deltok på/holdt.**

Kilder for termer:

Interface – grensesnitt: https://www.termportalen.no/NOT/Grensesnitt_-281-29

N Design Handbook for Vitensenteret Innlandet



VITENSENTERET INNLANDET
DESIGNHÅNDBOK
MAI 2021

1 BAKGRUNN



BAKGRUNN

Våren 2021 meislet Vitensenteret Innlandet ut en ny strategi og staket ut en ny kurs fram mot 2024. Strategien ble oversatt visuelt, i form av en ny visuell identitet som på en tydelig og enhetlig måte skal vise hvem de er og hva de står for.

VISJON

Vi skaper moro med mening!

VERDIER

Leken, inspirerende og alltid aktuell

HOVEDMÅL

Hovedmålet for Vitensenteret Innlandet er å styrke realfagkompetansen i hele Innlandet og vise at realfag er praktisk hverdagskunnskap og en nøkkel til bærekraftig samfunnsutvikling.

2 LOGO



VITENSENTERET

HOVEDLOGO

Logoen består av tre grunnformer, trekant, sirkel og rektangel, som tilsammen danner ordet "VI". VI er forkortelsen for Vitensenteret Innlandet, som kan føles stivt og noe tungt å kommunisere ut i sin helhet. Formene er avrundet for å skape et nært og lekent uttrykk.

Vitensenteret Innlandet er bevisst lagt vertikalt for å skape energi og bevegelse, selv når logoen presenteres i sin mest seriøse form. Fokus på merket fremfor tekst, opprettholdes og forsterkes ved å gi ordene likevekt.



VITENSENTERET
INNLANDET



VITENSENTERET

LOGO

Logoen kan benyttes med horisontal tekst. Størrelsesforholdene beholdes fra den vertikale hovedlogoen.

Teksten er plassert slik at den flukter med l'en for å opprettholde bevegelsen og lekentheten i logoen.



**VITENSENTERET
INNLANDET**



VITENSENTERET STØRRELSER OG AVSTANDER

Det er laget et beskyttelsesområde rundt logoen slik at den får den plassen den behøver for god synlighet. Logoen til Vitensenteret Innlandet skal alltid skaleres proporsjonalt.

Proporsjoner eller størrelsesforhold kan ikke endres.





VITENSENTERET
HOVEDLOGO

Dette er hovedlogoen og det skal etterstrebes
å benytte denne varianten.



VITENSENTERET
INNLANDET



VITENSENTERET
HOVEDLOGO NEGATIV



VITENSENTERET
INNLANDET



**VITENSENTERET
MERKE**

Merket kan benyttes alene og i hele profilens fargepalett.



3 TYPOGRAFI



VITENSENTERET
FONTBRUK

OPEN SANS

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Enkel, godt lesbar og tidløs.

LUCKIEST GUY REGULAR

Nær og leken



VITENSENTERET
FONTBRUK

VI ER ASTRONAUTER

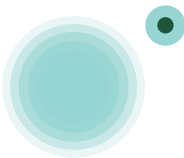
Lorem ipsum dolor sit amet, consectetur
 adipiscing elit, sed diam nonummy nibh euismod
 tincidunt ut laoreet dolore magna aliquam erat

4 FARGEPALETT

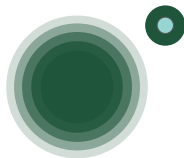


VITENSENTERET FARGEPALLETT

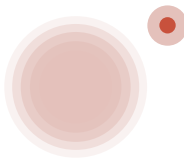
En glad og jordnær fargepalett til bruk på ulike områder, som snakker sammen.



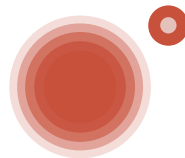
PANTONE: 2227 U
CMYK: 40 / 0 / 20 / 0
RGB: 150 / 213 / 210
#96D5D2



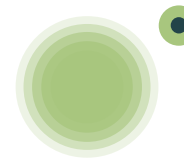
PANTONE: 329 U
CMYK: 60 / 60 / 70 / 0
RGB: 31 / 85 / 59
#1F553B



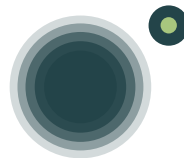
PANTONE: 2051 U
CMYK: 9 / 25 / 20 / 0
RGB: 228 / 193 / 187
#E4C1BB



PANTONE: 7627 U
CMYK: 10 / 80 / 80 / 10
RGB: 200 / 81 / 60
#C8513C



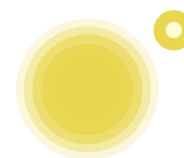
PANTONE: 577 U
CMYK: 30 / 0 / 60 / 10
RGB: 168 / 198 / 126
#F26F21



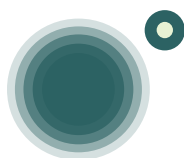
PANTONE: 3165 U
CMYK: 80 / 50 / 50 / 50
RGB: 35 / 68 / 73
#234449



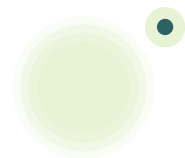
PANTONE: 20% Yellow U
CMYK: 0 / 0 / 20 / 0
RGB: 255 / 252 / 213
#FFFCD5



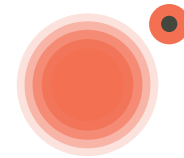
PANTONE: 604 U
CMYK: 5 / 5 / 80 / 5
RGB: 233 / 215 / 79
#E9D74F



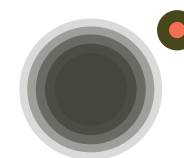
PANTONE: 322 U
CMYK: 80 / 40 / 50 / 30
RGB: 44 / 98 / 99
#2C6263



PANTONE: 614 U
CMYK: 9 / 0 / 20 / 0
RGB: 232 / 242 / 212
#E8F2D4



PANTONE: 7625 U
CMYK: 0 / 70 / 70 / 0
RGB: 243 / 112 / 83
#F37053



PANTONE: 412 U
CMYK: 70 / 60 / 70 / 40
RGB: 68 / 71 / 62
#44473E

5 DEKORELEMENTER

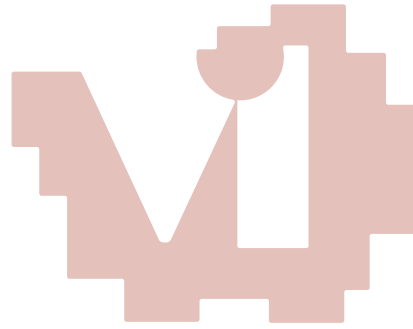


VITENSENTERET
DEKORELEMENTER
NATUR



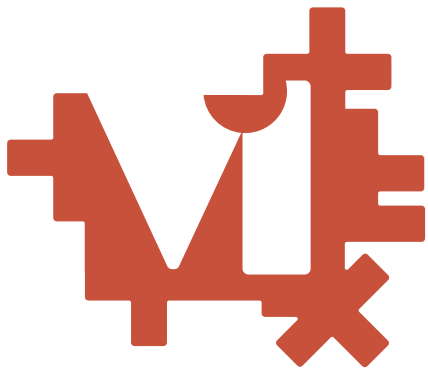


VITENSENTERET
DEKORELEMENTER
TEKNOLOGI





VITENSENTERET
DEKORELEMENTER
MATEMATIKK





VITENSENTERET
DEKORELEMENTER
ENERGI





VITENSENTERET
DEKORELEMENTER
ASTRONOMI



6 SLAGORD

VI SKAPER MORO MED MENING!

7 BILDEFILOSOFI

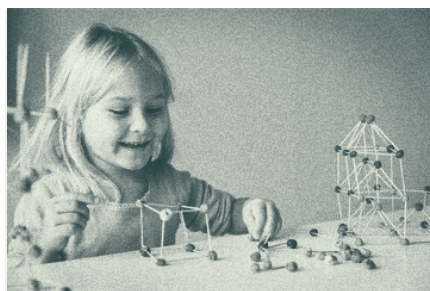
VITENSENERET

BILDEFILOSOFI

Bilder av glade barn og unge som skaper moro med mening.

Støyfilter for å skape litt "avstand" og opprettholde nysgjerrighet.

Varme og mindre metning. Bilder kan benyttes som entone-foto i respektive fargepalett.



8 IKONER



VITENSENTERET
IKONER

DET ANBEFALES Å VIDEREFØRE STILEN SOM ER I BRUK OG UTVIKLE NYE ETTER BEHOV.

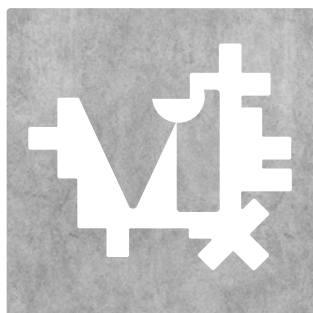


9 MATERIALBRUK



VITENSENTERET MATERIALBRUK

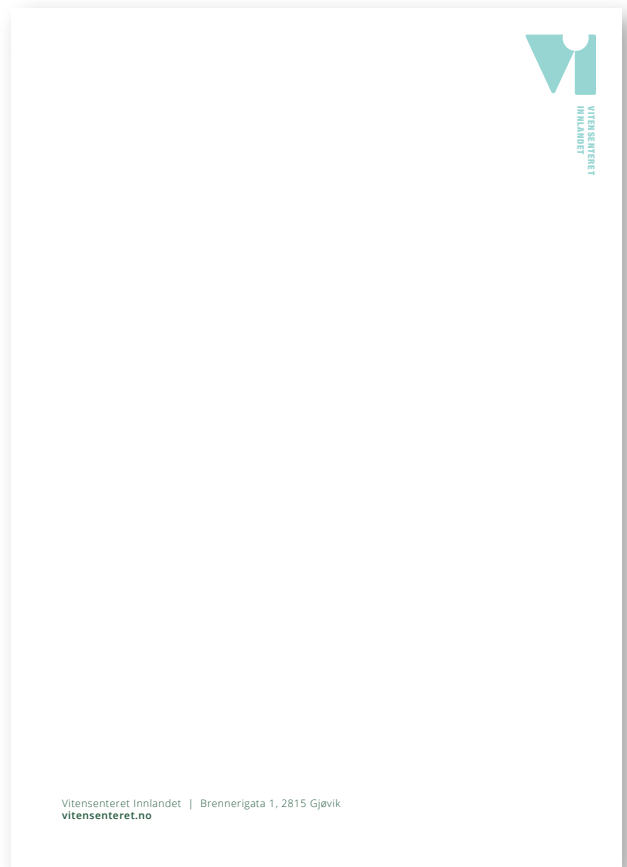
Det er mye farger og mange malte flater i Vitensenteret.
Bruk av ulike materialer til interiør og kulisser anbefales.
På denne måten vil også de fargerike flatene komme tydeligere fram.
Merke og dekorelementer kan benyttes som sjablon.



10 BRUK



VITENSENTERET
EKSEMPLER PÅ BRUK
POSTALE TRYKKSAKER





VITENSENTERET
EKSEMPLER PÅ BRUK
MAILSIGNATUR





VITENSENTERET
EKSEMPLER PÅ BRUK
GENERELLE PLAKATER

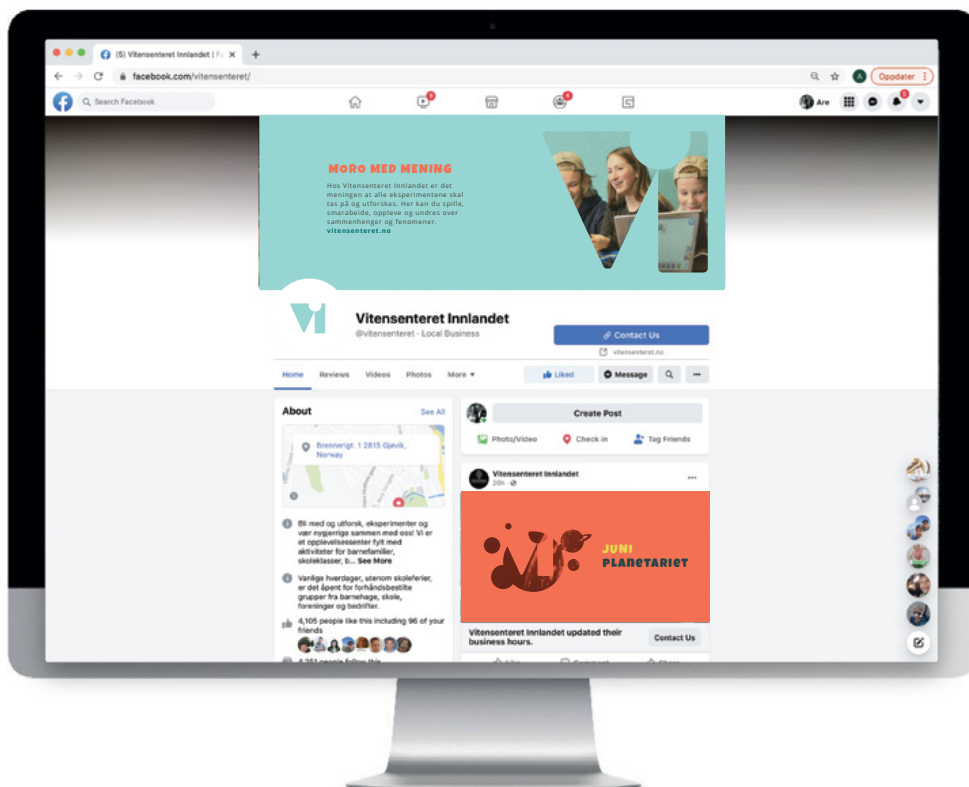




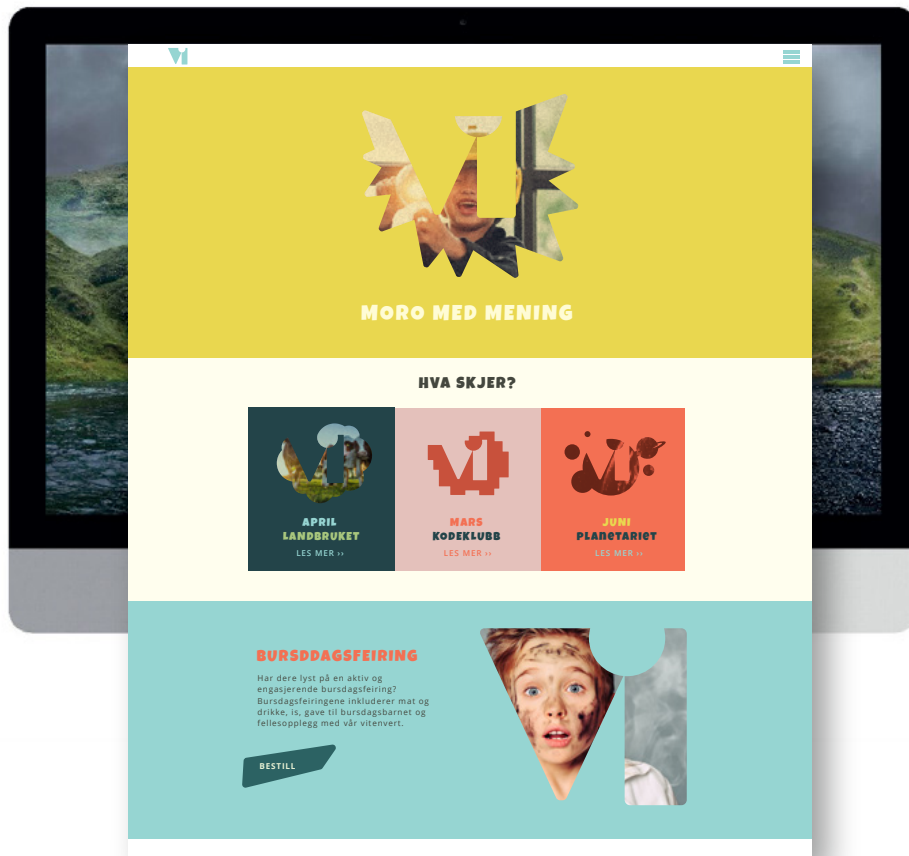
VITENSENTERET
EKSEMPLER PÅ BRUK
DIVERSE GENERELLE ARTIKLER



VITENSENTERET
EKSEMPLER PÅ BRUK
FACEBOOK



VITENSENTERET
EKSEMPLER PÅ BRUK
WEB





VITENSENTERET
EKSEMPLER PÅ BRUK
INSTAGRAM





VITENSENTERET
EKSEMPLER PÅ BRUK
SPESIFIKKE PLAKATER





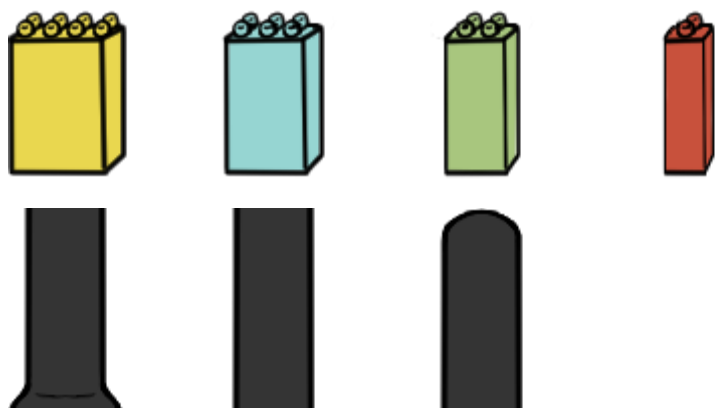
VITENSENTERET
EKSEMPLER PÅ BRUK
ANIMASJON OG LEK MED FARGEPALETT



/ 28

aredesign.no

O Assets



Hanois
Tårn



VIT
EN
ORD

P Backlog

Backlog-

Administrative

- ~~Konzept~~
- ~~Statusreport x3 #1 Trade~~
- ~~Entity report~~
- ~~Manual for admin page use (1 employee)~~
- ~~Lizens for a logo apper i appstore of Google Play store~~
- ~~GDPR~~
- X Vipps?
- ~~Stripe~~

App

- Login
- Registration
- ~~Create blog entry~~
- ~~Display blog entries~~
- ~~Delete blog entries~~
- Payment system / ~~direct vipps~~
- Gift code input
- Bottom bar
- Profile page
- Settings
- ~~Grant page + three games~~
- ~~Fix issues with web view login and registration~~
- ~~Display tickets~~
- Byte sprich
- Rituse penger
- ~~Edit blog~~
- ~~Vitenord~~
- ~~Honoris tesa~~
- ~~Vitenord~~
- ~~Delete user~~

Other

- ~~Connect to database~~
- ~~Reconnect connection to database~~
- X GitLab CI pipeline
- X Gain access to dwell API
- X Access vipps
- Tests
- Confirmation email

Q User Manual

Brukermanual for Vitensenter Appen



NTNU

Kunnskap for en bedre verden



VITENSENTERET INNLANDET

Den følgende manualen er ment å gi en innføring i hvordan bruke Vitensenter-appen som en ansatt ved Vitensenteret. Denne manualen forutsetter at programvaren er installert og klar til bruk.

Introduksjon:

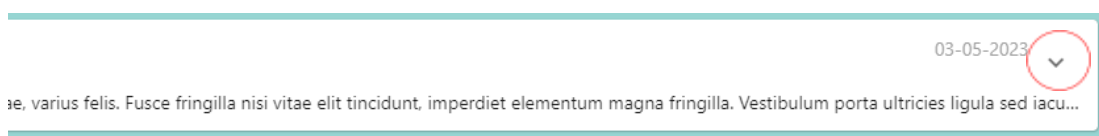
Denne appen er laget primært for brukere av vitensenteret, for at de lett skal ha en oversikt over nyheter relevant til Vitensenteret. Den har også billettene for brukeren her, samt noen spill. Vitensenter-appen har også unike muligheter for deg som er ansatt. Denne manualen går først gjennom grunnfunksjoner som er tilgjengelig uten å logge inn som ansatt, deretter går manualen over alle admin muligheter.

Blogg:

Bloggen er det første du ser når du åpner appen. Den ser slik ut:



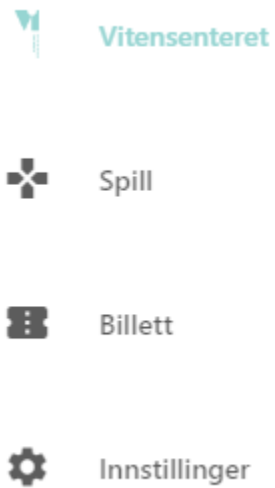
For å navigere denne siden kan du enkelt scrolle opp eller ned, ved tilfeller hvor blogginnleggene fyller siden. Blogginnleggene er presentert med en minimal forhåndsvisning. Du kan utvide blogginnlegget ved å trykke på den lille pilen ved siden av datoen til høyre:



Dersom du vil lukke den utvidede bloggen trykker du på samme sted.

Navigasjon:

På siden til venstre er det en navigasjonsbar. Denne ser slik ut:

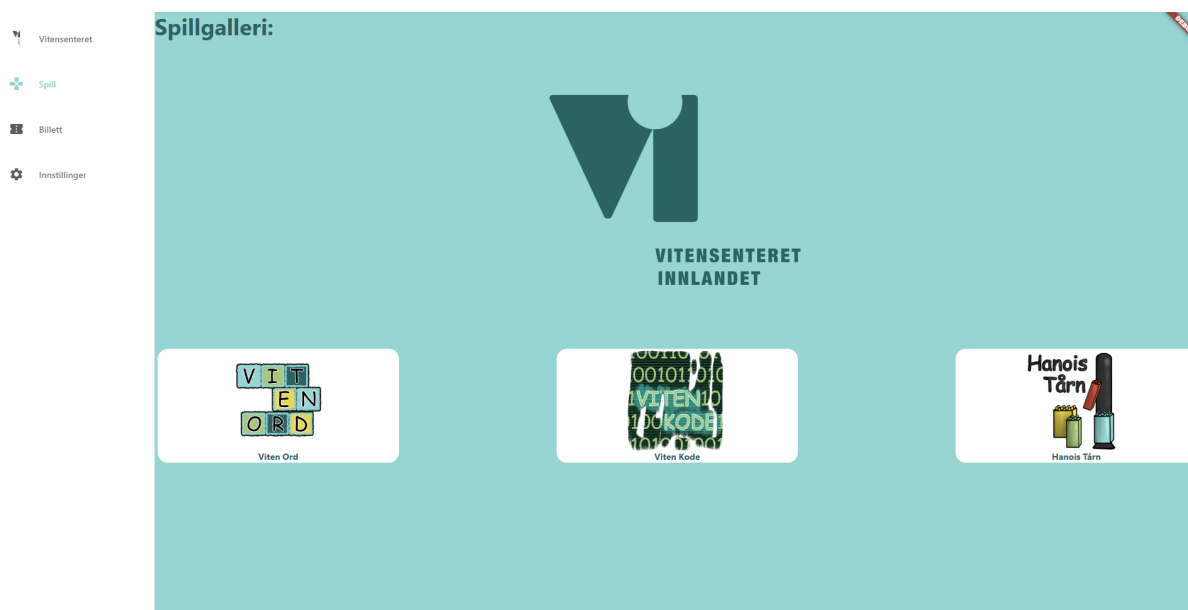


Denne viser de fire hovedsidene i appen. Bloggen er det som er kalt "Vitensenteret". I eksempelet vist, er den nåværende siden blogg. Dersom du navigerer til en ny side ved å klikke her, viser det nye siden valget ved å skifte farge på navigeringsmenyen:



Spill:

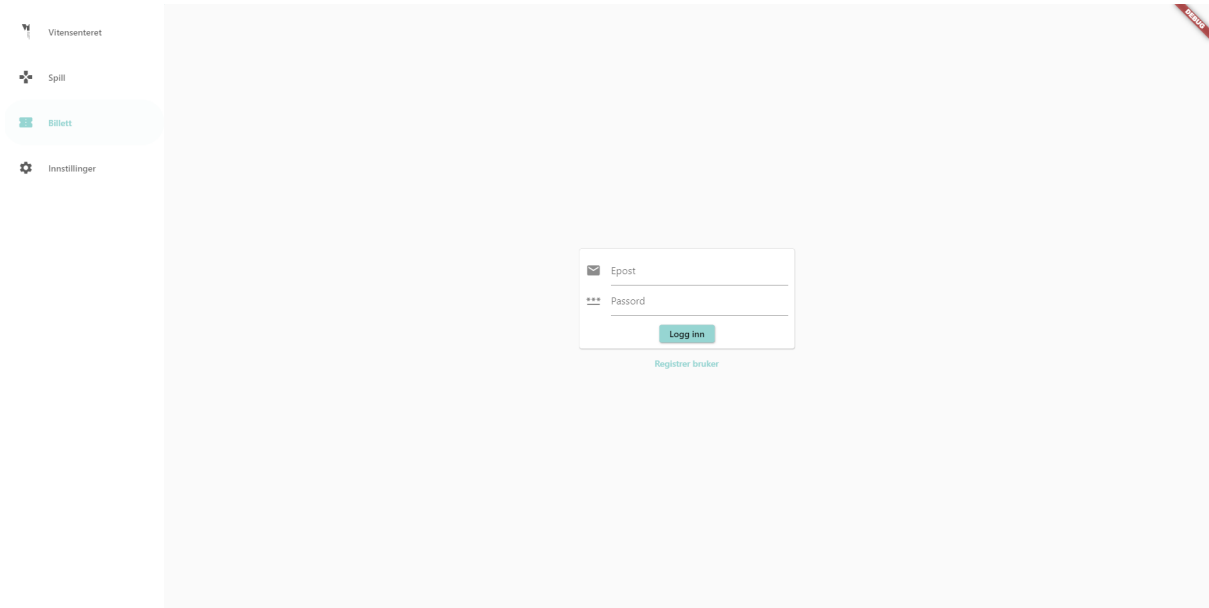
Spillene kan bli funnet under spillfanen i navigerings menyen. Den har en hovedside som leder til hver sine spill. Den ser slik ut:



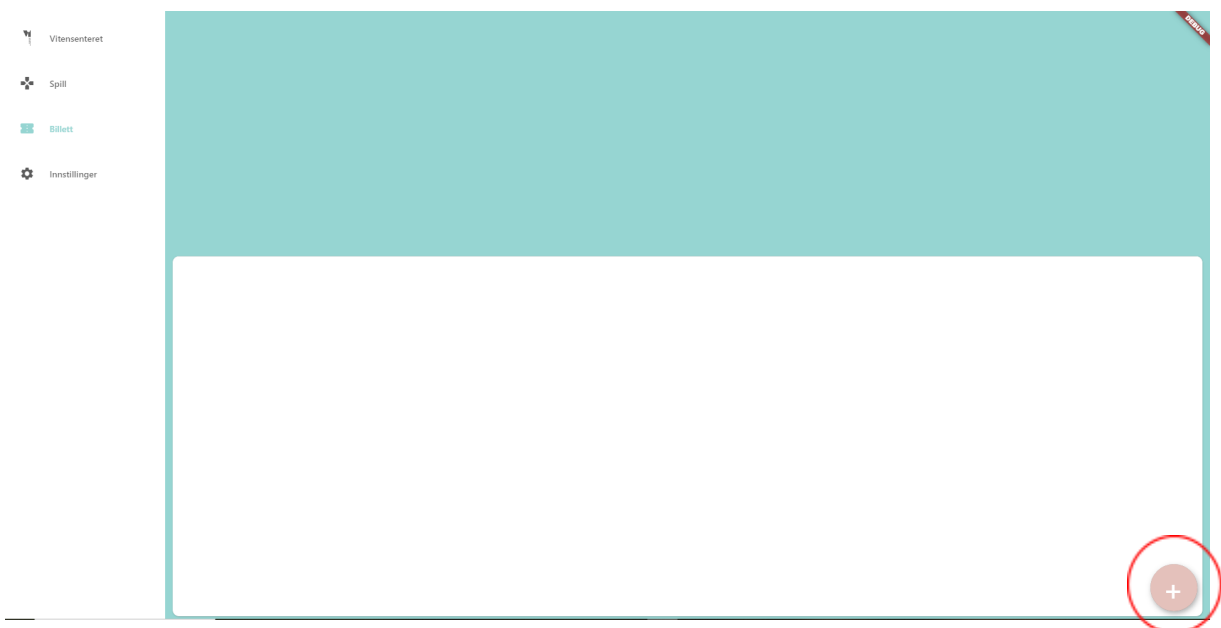
De tre spillene: “Viten Ord”, “Viten Kode” og “Hanois Tårn” kan spilles om du trykker i den hvite boksen, eller på bildet til spillene.

Billett:

Om du navigerer til billett fanen får du denne siden:



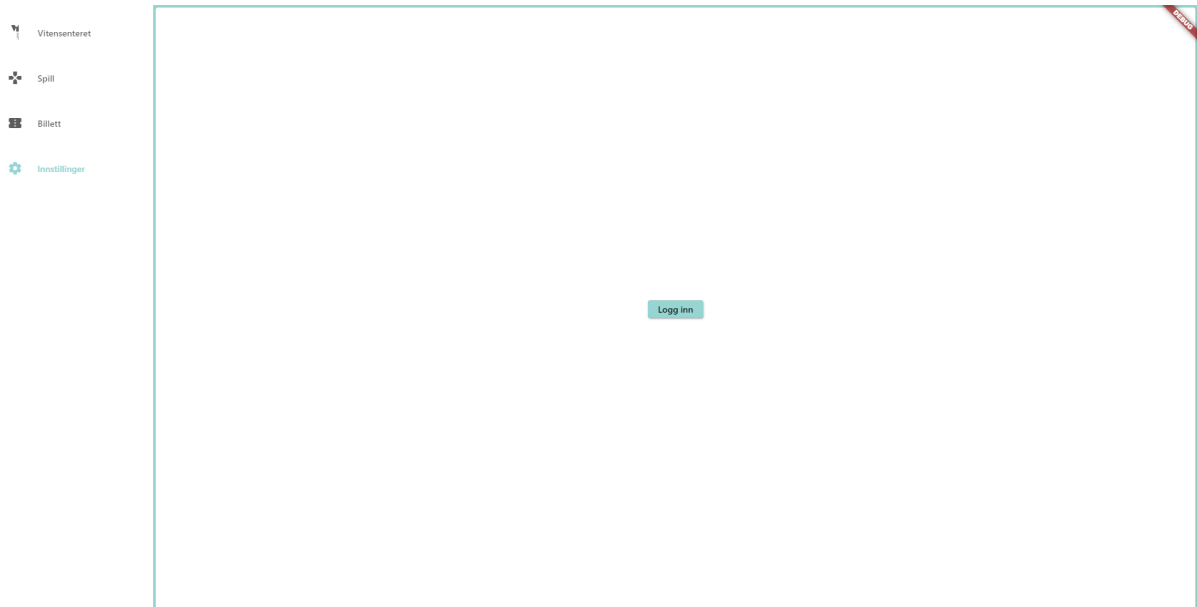
Dette er siden som inneholder informasjon om brukere. Om du logger inn med din bruker, får du tilgang til siden. Dersom du logger inn som en ikke-ansatt, får du ikke tilgang til ansattressursene. Siden til en ikke-ansatt ser slik ut:



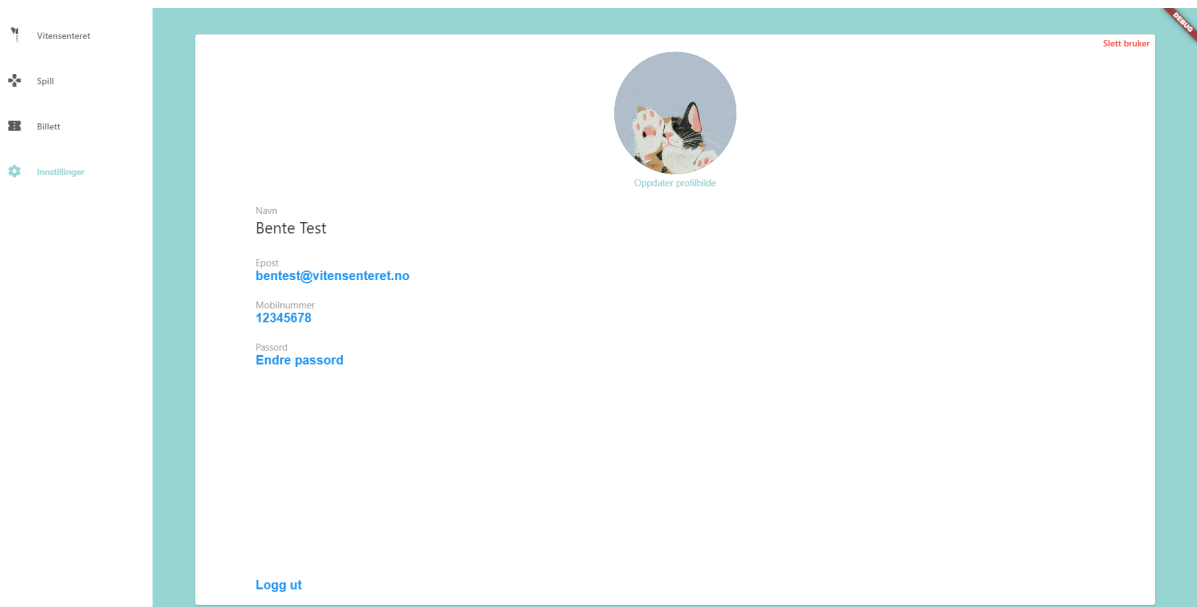
Hvor billetter blir listet i den hvite boksen. For å kjøpe eller legge til billetter via en kode kan en ikke-ansatt navigere til pluss ikonet. Siden for ansatte blir gått igjennom i den admin spesifikke delen av manualen.

Innstillinger:

Siden for innstillinger ser slik ut, om du ikke er logget inn:



Klikker du knappen, logger inn og skriver inn din innloggingsinformasjon kommer du til denne siden:



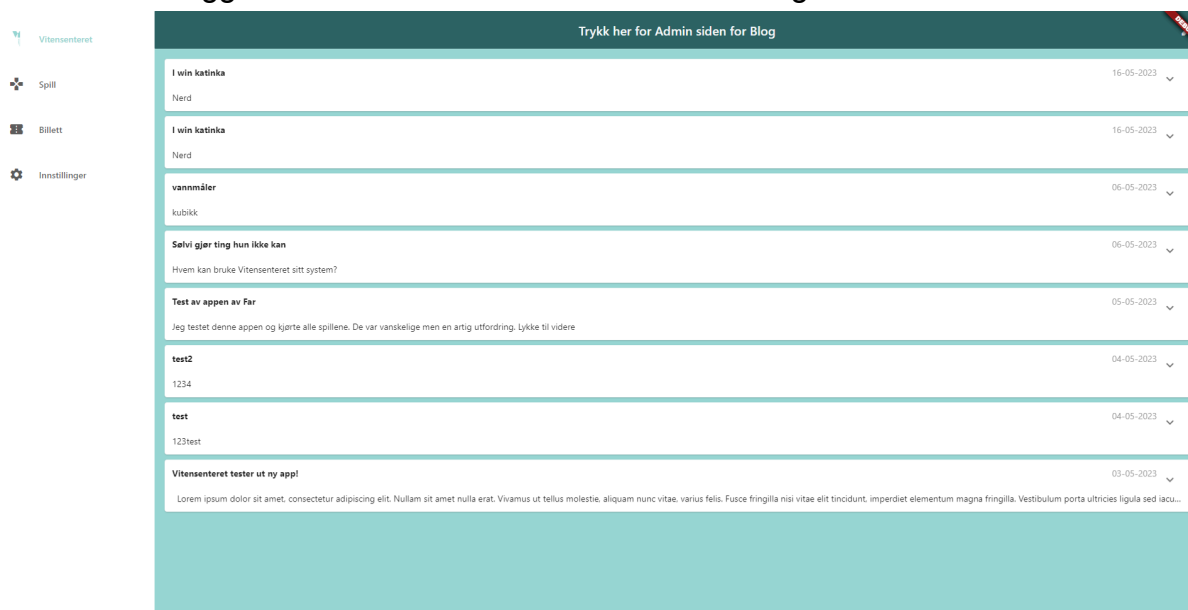
Her kan du endre informasjon, potensielt noe som kanskje ble skrevet feil, eller oppdatere noe informasjon som har forandret seg. For å gjøre dette trykker du på den informasjonen du ønsker å endre. Om du trykker "logg ut" blir du logget ut. Om du trykker "Slett bruker" blir brukeren slettet.

Bruk av produktet som ansatt:

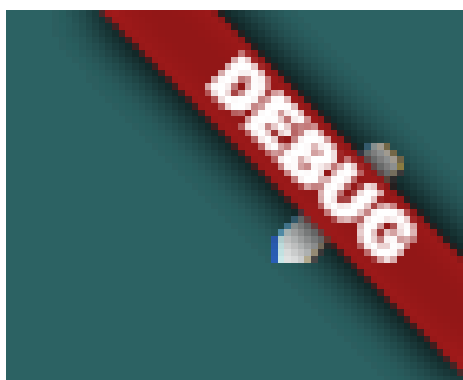
Som ansatt ved Vitensenteret kan du legge ut nye innlegg på bloggsiden og redigere dem som allerede er der. Du har også oversikt over alle brukerne av appen, i tillegg til alle billettene, både aktive og utgåtte. Resten av manualen går over bruk av disse funksjonalitetene.

Bruk av blogg som ansatt:

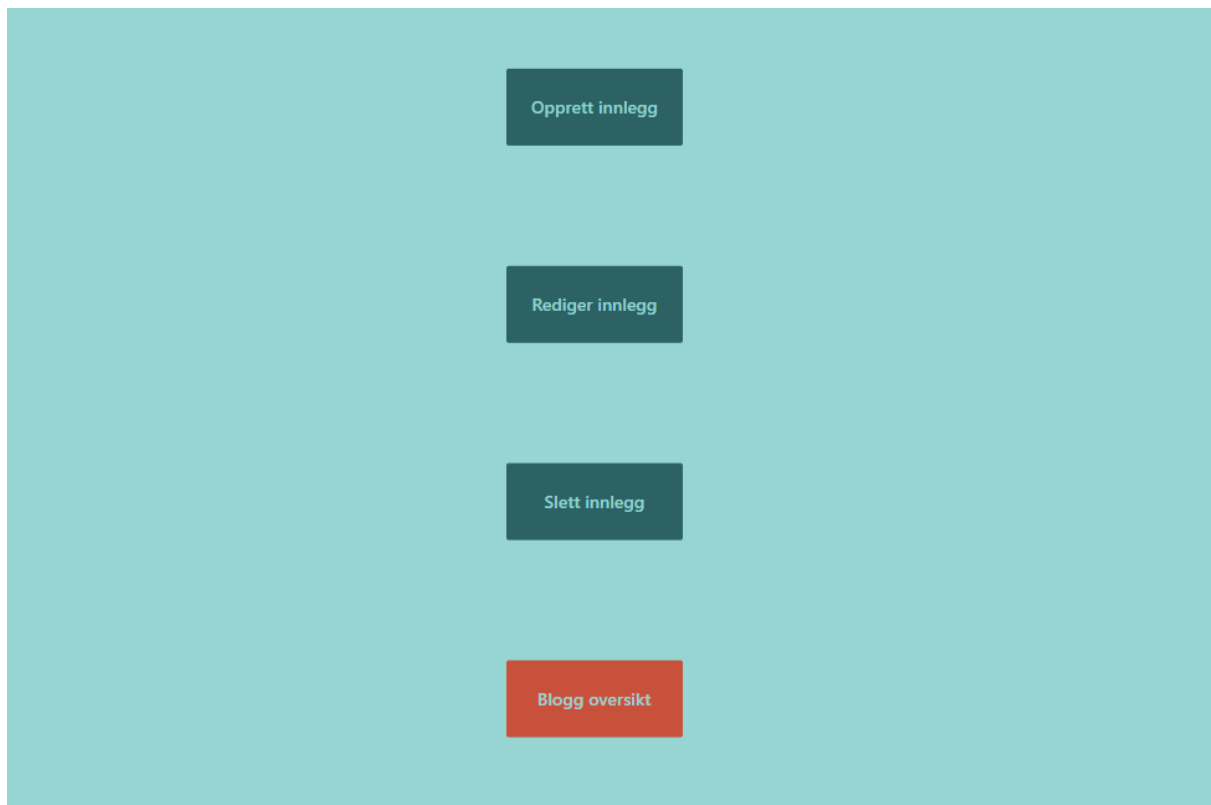
Når du er innlogget som ansatt ved Vitensenteret ser blog siden slik ut:



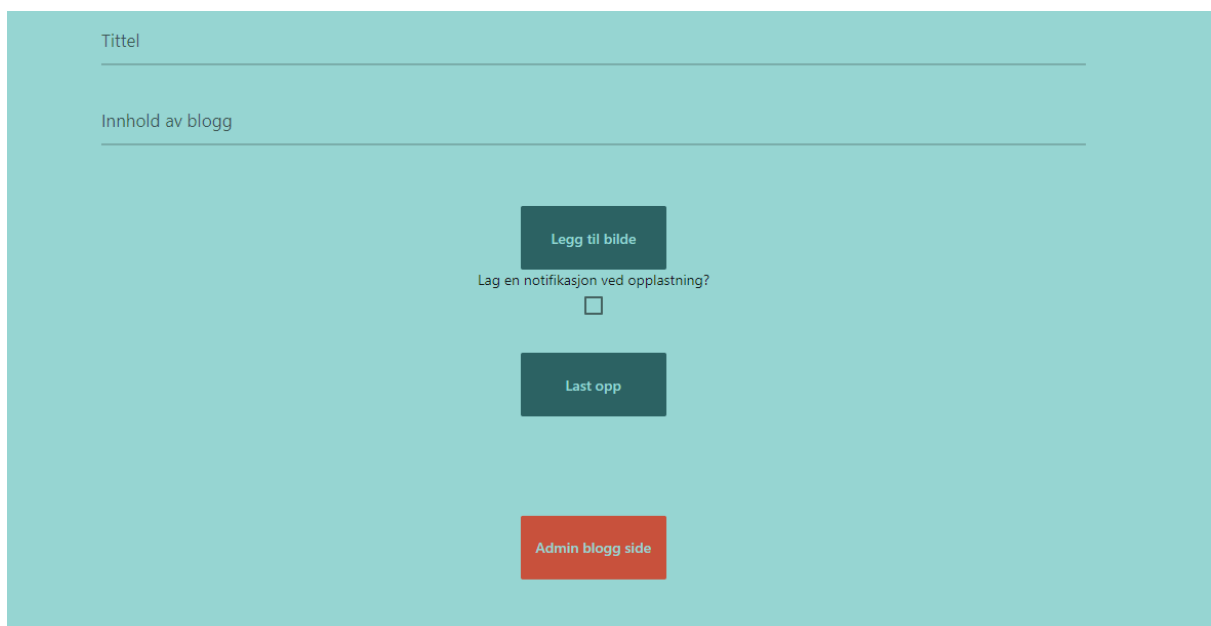
Hovedforandringen er den mørke baren på toppen av siden med teksten "Trykk her for Admin siden for Blog" Om du da trykker på blyanten til høyre, kommer du til admin-siden. Denne figuren er dessverre bak debug banneret, men den ser slik at:



Etter å ha trykket på dette ikonet, kommer du til følgende side:




Her kan du opprette innlegg, redigere dem, eller slette dem. Om du trykker på den røde knappen helt nederst kalt "Blogg oversikt" kommer du tilbake til bloggen. Om du trykker på "Opprett innlegg" kommer du til denne siden:



Her kan du legge inn informasjon om innlegget. Her er et eksempel på et innlegg med fylt ut informasjon før det blir lastet opp:

Manual

Dette blogginnlegget ble laget for manualen



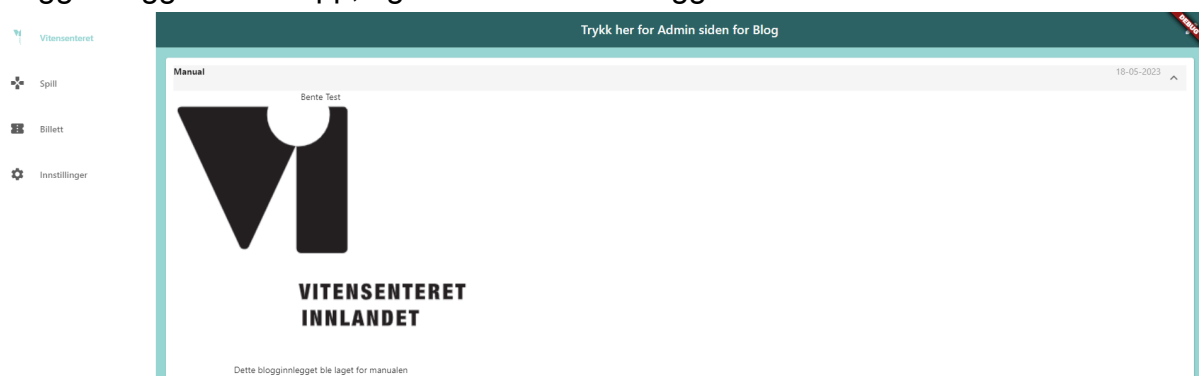
Bytt bilde

Lag en notifikasjon ved opplasting?

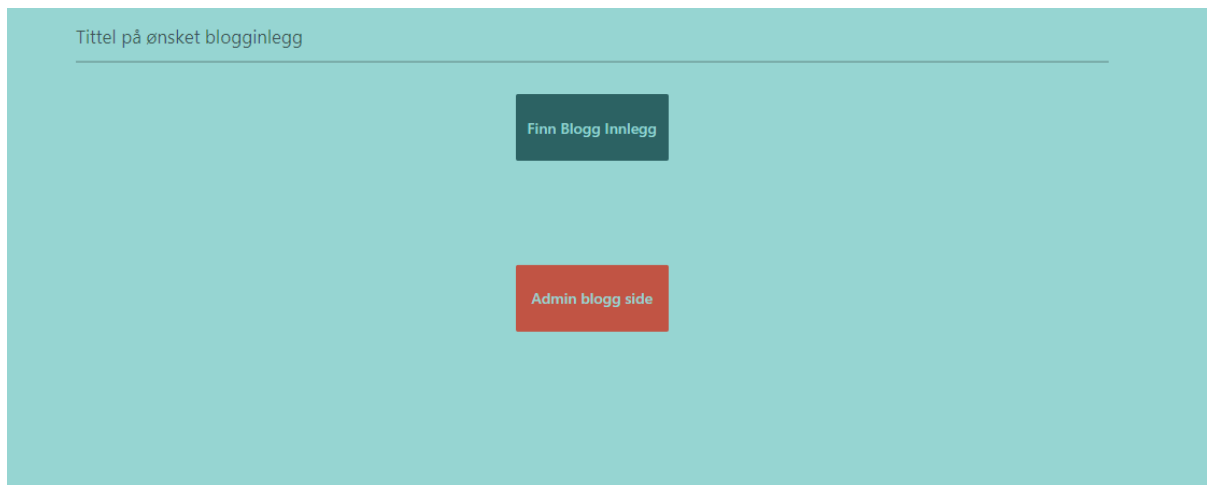
Last opp

Admin blogg side

Du kan til ethvert tidspunkt trykke på den røde knappen “Admin blogg side” for å returnere til admin siden uten å legge til et innlegg. Du kan også trykke av i boksen for å lage en notifikasjon ved opplasting. Dessverre er ikke dette koblet til noe videre funksjonalitet, så en notifikasjon blir ikke sendt ut selv om du markerer for det. Forfatter og dato blir automatisk lagt til. Om du trykker på “Last opp” blir blogginnlegget lastet opp, og kan nå finnes i blogg oversikten:



Dersom du på admin blog siden i stedet velger “Rediger innlegg” knappen kommer du til denne siden:



Tittel på ønsket blogginlegg

Finn Blogg Innlegg

Admin blogg side

Her kan du skrive inn tittelen på det innlegget som du ønsker å redigere. Du trykker på knappen “Finn Blogg Innlegg” for å søke. Om den ikke finner innlegget kommer denne meldingen opp:

Dette innlegget eksisterer ikke, vær obs å skrive HELT likt som tittel

Om den finner innlegget kommer du til denne siden, hvor du kan redigere innhold eller tittel:



Manual

Dette blogginnlegget ble laget for manualen, og er nå redigert

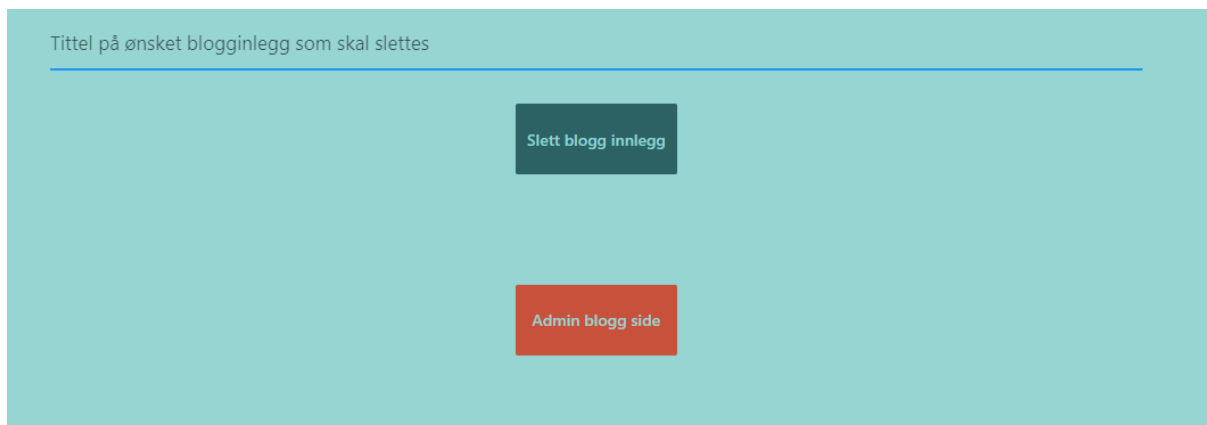
Oppdater

Admin blogg side

Om du da trykker på “Oppdater” blir bloggen oppdatert og den kan bli funnet i blogg oversikten:



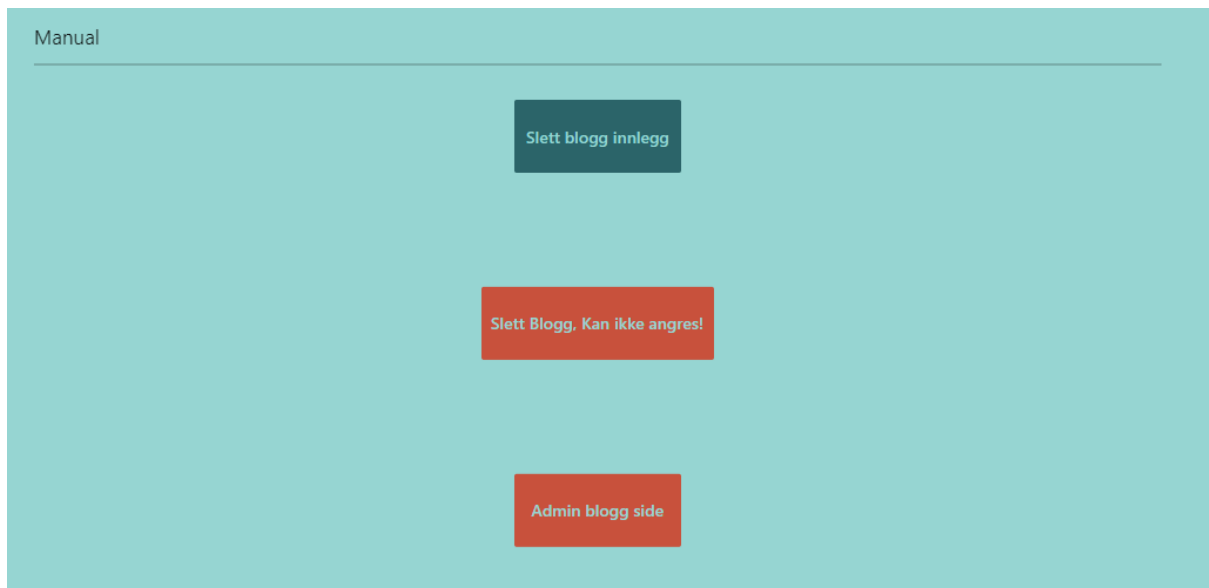
Tilbake på admin siden, dersom du trykker på “Slett innlegg” kommer du til denne siden, som lar deg søke etter det blogginnlegget som du ønsker å slette:



Her skriver du inn tittelen på innlegget du ønsker å slette. Deretter trykker du på “Slett blogg innlegg”. Om du skriver inn et innlegg som ikke eksisterer får du denne meldingen:

Dette innlegget eksisterer ikke, vær obs å skrive HELT likt som tittel

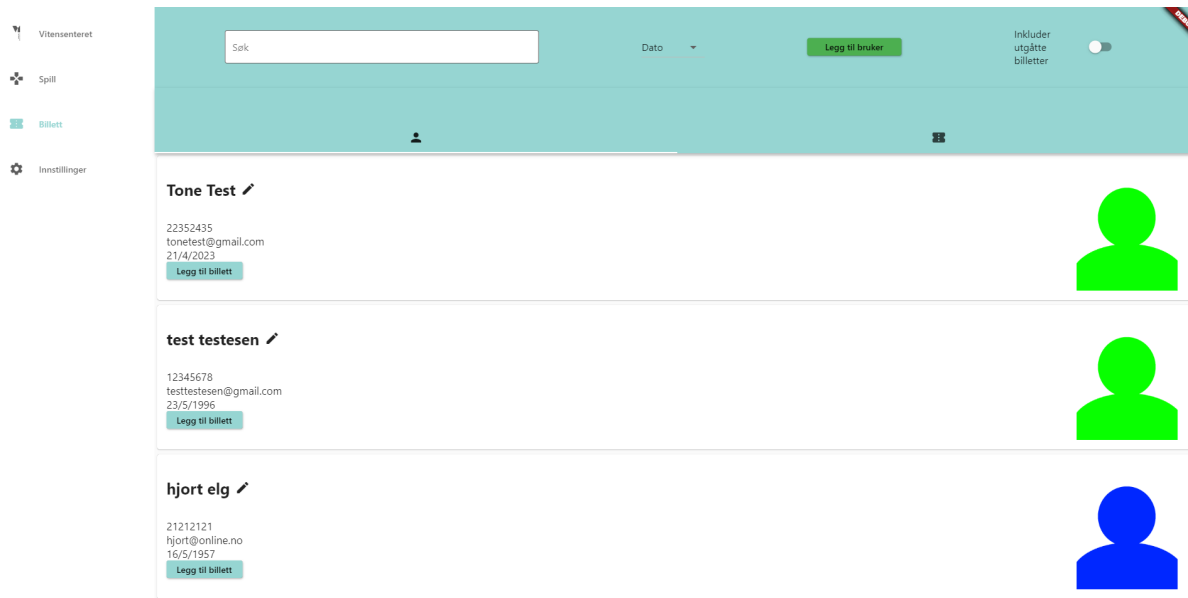
Dersom innlegget du ønsker å slette eksisterer får du denne nye knappen:



Dersom du nå trykker på “Slett Blogg. Kan ikke angres!” slettes innlegget. Du kan også søke etter et annet innlegg om dette var feil, eller angre deg ved å trykke “Admin blogg side”, som tar deg tilbake til adminsiden.

Bruk av oversiktsside for brukere og billetter

Denne siden er ment som et verktøy hovedsakelig for ansatte som jobber i kassen. Her ser man fullstendig oversikt over alle brukere, og alle billetter.

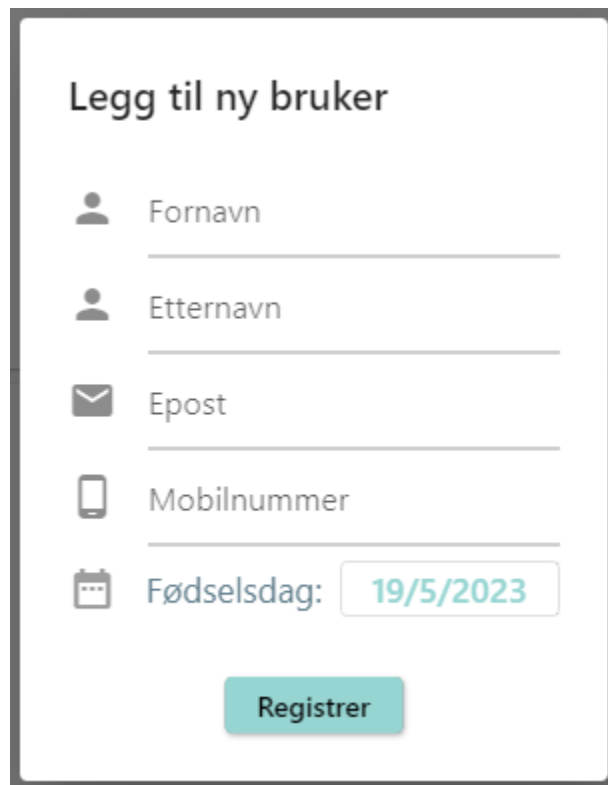


Øverst er det en del verktøy som kan brukes til forskjellige ting. Fra venstre: søkelinje, sorter etter meny, opprett ny bruker-knapp og bryter for å inkludere eller ekskludere utgåtte billetter.

Søkelinjen kan brukes til å søke etter navn på bruker, eller navn på billetteier. Man kan søke etter e-postadresse, mobilnummer og gavekortkoder. Søkeresultatet vises underveis når man skriver, og om ingenting matcher vil bruker- og billettfanene være tomme.

Menyen for å sortere brukere og billetter lar deg sortere etter en av to ting: dato eller alfabetisk. For billetter vil den da sortere etter kjøpsdato eller alfabetisk på hvem som kjøpte billetten. For brukere sorterer den etter fødselsdag eller navn. Merk at begge listene starter usortert når man åpner appen.

Når man skal opprette en ny bruker bli man møtt av denne dialogen:



The image shows a registration dialog box with the title "Legg til ny bruker". It contains five input fields, each with an icon to its left: a person icon for "Fornavn", another person icon for "Etternavn", an envelope icon for "Epost", a mobile phone icon for "Mobilnummer", and a calendar icon for "Fødselsdag:". The "Fødselsdag:" field is pre-filled with the date "19/5/2023". At the bottom center of the dialog is a teal button labeled "Registrer".

Her må man skrive inn informasjon om den nye brukeren, og deretter trykke registrer. Merk at dette vil *ikke* la denne brukeren logge inn i appen, da må det opprettes en ny bruker via appen.

Bryteren for å inkludere utgåtte billetter kan aktiveres når det er nødvendig å se alle billetter. Det anbefales å bruke denne sammen med søkelinjen, da det fort kan bli mange billetter å se gjennom om man ser etter noe spesielt.

Lenger ned på siden ser man de to fanene som holder oversikten over alle brukere og alle billetter. La oss ta for oss fanen med billetter først:

Kjøpt dato	Aktivert dato	Utløpsdato
01 / 01 / 2022	10 / 10 / 2022	09 / 10 / 2023
01 / 01 / 2022	10 / 08 / 2022	09 / 08 / 2023
13 / 04 / 2023	24 / 04 / 2023	23 / 04 / 2024
23 / 04 / 2023	14 / 04 / 2023	22 / 04 / 2024
24 / 04 / 2023	24 / 04 / 2023	23 / 04 / 2024

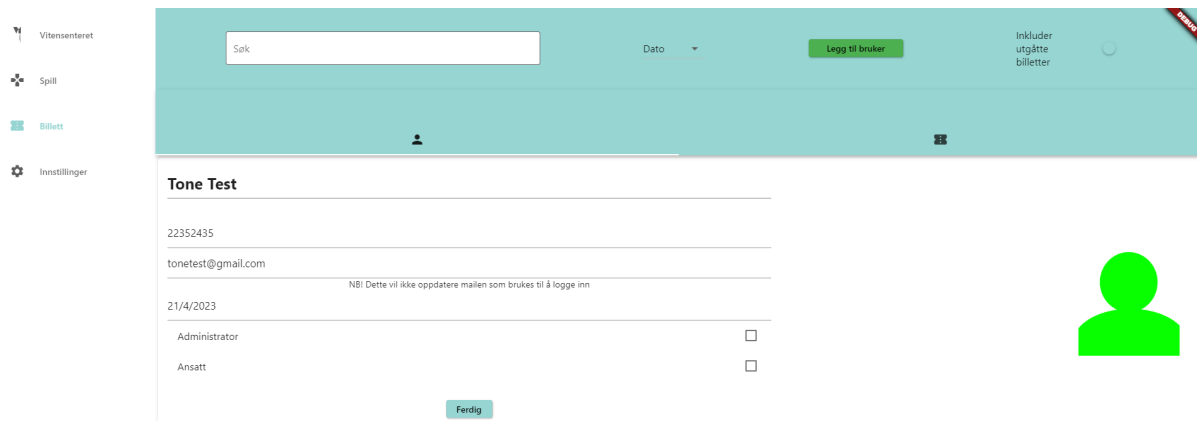
Her ser man all informasjon om hver enkelt billett. Hvis det finnes årskort som er utgått, vil disse ha en ekstra knapp. Denne knappen starter årskortet igjen, fra dags dato til et år frem i tid. Dette er ikke reversibelt.



I fanen med brukere ser man all relevant brukerinformasjon om alle brukere i systemet.

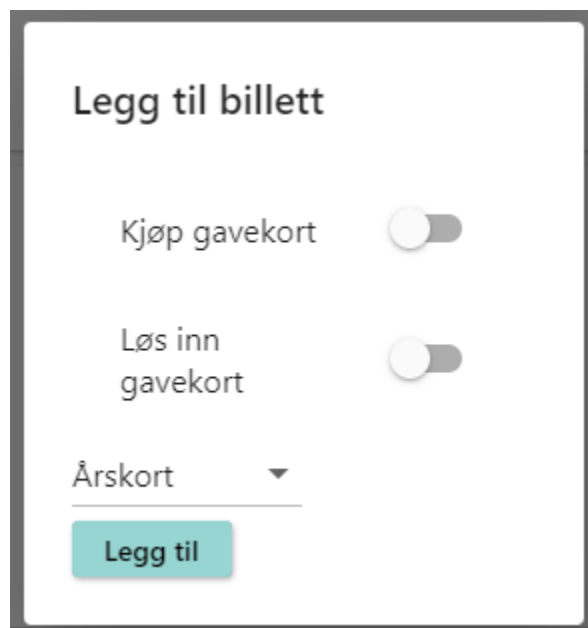
Navn	ID	Email	Fødselsdato	Legg til billett
Tone Test	22352435	tonetest@gmail.com	21/4/2023	Legg til billett
test testesen	12345678	testtestesen@gmail.com	23/5/1996	Legg til billett
hjort elg	21212121	hjort@online.no	16/5/1957	Legg til billett

Her kan man også gå inn å redigere brukerinformasjon, merk at dersom epost endres vil dette ikke påvirke eposten som brukes til å logge inn i appen med. Om den skal endres må det gjøres i appen.



The screenshot shows a user profile editing screen. On the left is a sidebar with menu items: Vitensenteret, Spill, Billett, and Innstillinger. The main content area has a teal header with a search bar labeled 'Søk', a 'Dato' dropdown, a 'Legg til bruker' button, and a toggle for 'Inkluder utgåtte billetter'. Below the header, the user's name 'Tone Test' is displayed. The profile information includes: ID '22352435', email 'tonetest@gmail.com', and date '21/4/2023'. A note states: 'NB! Dette vil ikke oppdatere mailen som brukes til å logge inn'. There are two roles listed: 'Administrator' and 'Ansatt', each with an unchecked checkbox. A green profile picture icon is on the right. A 'Ferdig' button is at the bottom.

Man kan også legge til billetter til brukere, dette gjøres via 'Legg til Billett'-knappen. Deretter skriver man inn all informasjon som trengs, og trykker Legg til. Billetten er nå registrert i system, og vil vises både i app og i fanen for billetter. Det kan hende at siden må lastes inn på nytt før den vises.



The screenshot shows a dialog box titled 'Legg til billett'. It contains two toggle switches: 'Kjøp gavekort' (off) and 'Løs inn gavekort' (off). Below these is a dropdown menu labeled 'Årskort' with a downward arrow. At the bottom is a teal 'Legg til' button.

Lykke til!



 **NTNU**

Norwegian University of
Science and Technology