Anovote

BACHELOR THESIS

IE303612 Bachelor thesis computer engineer

Emil Elton Nilsen, Sander Hurlen Olsen,

Steffen Holanger, and Christoffer Andersen Træen

Ålesund, May 2021

Total pages: 187

Supervisor: Arne Styve

Acknowledgements

We would like to dedicate this bachelor thesis to Arne Styve for his fantastic work as our bachelor thesis supervisor. Arne was always a great help with any questions we asked him, and he was never afraid to encourage our group and guide us forward.

We would also like to dedicate this bachelor thesis to Girts Strazdins for his work as the bachelor thesis coordinator for the computer science program at NTNU Ålesund. In the process of choosing our bachelor thesis and working on it, there was never any issue with management of the bachelor thesis.

We also express our special thanks to:

- Anders Søbakken who assisted us with configuring our hosting server
- All the professors at NTNU Ålesund who have shared with us their knowledge, nudging us in the direction of greatness
- Patricia Larsen and Start Ålesund for having the idea and entrusting us with the task

Thank you

Preface

About

The following document is a bachelor thesis written as part of Computer Engeneering at NTNU Ålesund. It has been conducted by Emil Elton Nilsen, Sander Hurlen Olsen, Christoffer Andersen Træen and Steffen Holanger as their final assignment.

The idea for this bachelor thesis was based on a request from Start Ålesund, who wanted a digital voting system to use at their general assembly. The assignment describes the process of developing such a digital voting applications.

What intrigued us with this bachelor project was the chance to develop a production-ready application with real-world applicability. There has been a learning curve throughout the bachelor thesis and something we are certain will benefit from when approaching a professional career.

Summary

In 2019, Start Ålesund wanted a digital solution to perform their elections at their annual general assembly, up until then all counting were done manually, which were prone to errors. In the subject ID102012 - webteknologi, a Proof-of-Concept (PoC) was developed. The solution worked during demonstration, but due to time constraint had many faults. Based on this PoC, we have created a production ready system to perform digital elections.

The application is developed as a Single-page Application (SPA) running in conjunction with a Node server that utilizes a combination of REST API and WebSockets for communication. The project has been applying agile methodology principles by using the SCRUM framework.

The end result is a production-ready application, with a full CI/CD pipeline, hosted in a containerized network.

We have found that working with the agile methodology has enabled us to create a product which can be enhanced incrementally. Working with modern JavaScript technology for a full stack solution has proven sufficient for the task.

Contents

Acknowledgement i

Preface ii

Summary iii

Acronyms 2

Glossary 4

1 Introductions **11**

1.1 Background 11

1.2 Problem Formulation 11

1.3 Objectives 12

1.4 Literature Survey 12

1.5 Limitations 13

1.6 Structure of the Report 13

2 Theoretical Basis **14**

2.1 Voting 14

2.1.1 Electoral System . . . 15
2.1.2 Elections . . . 16
2.1.3 Ballot . . . 16
2.1.4 Electronic voting algorithms . . . 16
2.1.5 Blockchain . . . 18
2.2 Security . . . 19
2.2.1 SQL Injection . . . 20
2.2.2 Cross Site Scripting . . . 20
2.2.3 Cross-origin Resource Sharing . . . 20
2.2.4 Brute Force Attacks . . . 21
2.3 Web Application . . . 21
2.3.1 Single Page App . . . 21
2.3.2 Progressive Web Application . . . 22
2.3.3 Universal Design for Web . . . 22
2.4 Design Patterns . . . 22
2.4.1 Singleton . . . 22
2.4.2 Module Pattern . . . 23
2.4.3 Observer pattern . . . 23
2.5 Client-Server Communication . . . 23
2.5.1 HTTP . . . 23

2.5.2 REST API . . . 25
2.5.3 WebSocket . . . 26
2.5.4 Token . . . 26
2.6 Frameworks and Technologies . . . 27
2.6.1 JavaScript . . . 27
2.6.2 TypeScript . . . 27
2.6.3 HTML . . . 27
2.6.4 CSS . . . 28
2.6.5 SQL . . . 28
2.6.6 Runtime Environment . . . 28
2.6.7 Virtual Machines . . . 28
2.6.8 Third Party Libraries . . . 29
2.6.9 Localization . . . 29
2.7 Continuous Integration, Delivery, and Deployment . . . 30
2.7.1 Continuous Integration . . . 30
2.7.2 Continuous Delivery . . . 30
2.7.3 Continuous Deployment . . . 30
2.8 Tests . . . 32
2.8.1 Unit Testing . . . 32
2.8.2 Integration Testing . . . 32

2.8.3 Test-driven Development . . . 32
2.8.4 User Interface Testing . . . 32
2.8.5 Prototype Testing . . . 33
2.8.6 Usability Tests . . . 33
2.9 Software Project Methodologies . . . 35
2.9.1 The History of SCRUM . . . 36
2.9.2 The SCRUM Team . . . 36
2.9.3 Activities in SCRUM . . . 37
2.9.4 SCRUM Artifacts . . . 39
2.9.5 Estimating work . . . 40
2.10 Version Control and Code Management . . . 40
2.10.1 Version Control . . . 40
2.10.2 Semantic Versioning . . . 43
2.11 Development Environment . . . 43
2.11.1 Integrated development environment . . . 43
2.11.2 Source-code Editor . . . 43
2.11.3 Linter . . . 43
2.11.4 Code Formatter . . . 44
2.12 Graphical Design Principles . . . 44
2.12.1 Don Norman's Design Principles of Interaction Design . . . 44

2.12.2 Mobile-first Approach 45

3 Method **46**

3.1 Organization 46

3.1.1 Project Group 46

3.1.2 Supervisor 47

3.1.3 Former client 47

3.2 Project planning 47

3.2.1 Preliminary report 47

3.2.2 Gantt Chart 47

3.2.3 Risks 48

3.2.4 Requirement Specification 48

3.3 Project Methodology 48

3.3.1 Roles 48

3.3.2 Sprints 49

3.3.3 Stand-ups 49

3.3.4 Product Backlog 50

3.3.5 Story Points 50

3.4 Version Control and Code Management 50

3.4.1 GitHub 51

3.4.2 Issue Tracking 51

3.4.3 Pull Requests 52
3.4.4 Labels 52
3.4.5 Semantic Versioning 52
3.5 Development Tools and Applications 52
3.5.1 Visual Studio Code 52
3.5.2 Nginx 53
3.5.3 DataGrip 53
3.5.4 Postman 54
3.5.5 Adobe XD 54
3.5.6 Adobe illustrator 54
3.5.7 Visual paradigm 54
3.5.8 Discord 55
3.5.9 Overleaf 55
3.5.10 Zotero 55
3.5.11 Typora 55
3.5.12 Docksify 55
3.5.13 Framework and libraries 56
3.5.14 Runtime Environments 59
3.5.15 Docker 60
3.5.16 Database 60

3.5.17 Preprocessors and Transpilers 60
3.6 Programming, Markup and Scripting Languages 62
3.6.1 TypeScript and JavaScript 62
3.6.2 HTML 62
3.6.3 SCSS 62
3.6.4 PostgreSQL 63
3.6.5 Yaml 63
3.6.6 JSON 63
3.6.7 Markdown 63
3.7 Quality Assurance 64
3.7.1 Jest 64
3.7.2 Prototype Testing 64
3.7.3 Usability Testing 64
3.7.4 Large-scale User Testing 65
3.7.5 Unit Testing 65
3.7.6 Integration Testing 65
3.7.7 GitHub Actions 65
3.7.8 ESLint and Prettier 66
3.8 Environment 66
3.9 Product Design 67

3.9.1 Wireframing . . . 67
3.9.2 UI Design . . . 67
3.9.3 Mobile-first . . . 67
3.9.4 Diagram and Model Documentation . . . 67
3.10 Hardware . . . 68
3.10.1 Personal Equipment . . . 68
3.10.2 Server . . . 68

4 Result **69**

4.1 General results . . . 69
4.2 Use-Cases of the Application . . . 70
4.3 Functionality of the system . . . 71
4.3.1 Election Organizer . . . 71
4.3.2 Eligible Voter . . . 75
4.4 PWA . . . 77
4.5 Client-Server Communication . . . 77
4.5.1 Basic Communication . . . 78
4.5.2 Request to the REST API . . . 78
4.5.3 Real-time Communication . . . 79
4.5.4 Event Handling . . . 80
4.6 Developing a Voting System . . . 81

4.6.1 Election Lifecycle 81
4.6.2 Anomyzation with Algorithms 82
4.6.3 Verifying Participants when Joining an Election 83
4.6.4 Socket Room Service and Initilizing WebSocket Channels 88
4.6.5 Creating the Election Room 88
4.6.6 Initializing socket room 91
4.6.7 Pushing a Ballot 91
4.6.8 Vote Handling 93
4.6.9 Validation of election organizer 94
4.7 Frontend architecture and technologies 96
4.7.1 React 96
4.7.2 Ant Design as a React helper 96
4.8 Backend architecture 97
4.8.1 Node server 97
4.8.2 Database 98
4.8.3 TypeORM, Class-Transformer and Class-Validator 98
4.9 Using third-party library 103
4.10 Localizing the application 103
4.10.1 Providing meaningful error messages 104
4.11 Continuous Integration and Delivery 106

4.11.1 CI/CD process overview . . . 107
4.12 Environment files . . . 107
4.13 Tests and Quality Assurance . . . 109
4.13.1 Unit, UI and integration tests . . . 109
4.13.2 Usability testing . . . 110
4.13.3 Large-scale User Test . . . 110
4.13.4 Code formating and linting . . . 112
4.14 SCRUM . . . 112
4.14.1 Work estimation . . . 112
4.14.2 Distribution of work . . . 113
4.14.3 Sprint . . . 113
4.14.4 Stand ups . . . 113
4.15 Version control and code collaboration . . . 114
4.15.1 Git, GitFlow and GitHub . . . 114
4.15.2 GitHub Issues . . . 114
4.15.3 Pull Request . . . 115
4.15.4 Semantic versioning . . . 115
4.16 Project developed tools . . . 116
4.16.1 GitHub SCRUM chrome extension . . . 116
4.16.2 Environment setup . . . 117

4.16.3 Document generator . . . 119
4.16.4 Docsify sidebar generator . . . 120
4.17 Design . . . 121
4.17.1 Wireframing . . . 121

5 Discussion **125**

5.1 General result . . . 125
5.2 Functionality . . . 126
5.2.1 Election . . . 126
5.2.2 Election Life cycle . . . 126
5.3 Voting Implementation . . . 127
5.3.1 Blockchain as an Election Mechanism . . . 128
5.4 Client-server communication . . . 129
5.5 Framework decision . . . 130
5.5.1 Database handling . . . 130
5.6 Third-party Library . . . 131
5.7 Localizing our application . . . 132
5.8 Code Style Aiding . . . 132
5.9 Version Control and Code Management . . . 133
5.9.1 Semantic versioning . . . 133
5.10 Self Developed Tools . . . 134

5.10.1 GitHub Scrum Master . . . 134
5.10.2 Anovote CLI . . . 135
5.10.3 start-production script . . . 135
5.10.4 Document generator . . . 136
5.11 Testing . . . 136
5.11.1 Usability Tests . . . 137
5.11.2 Large-scale User Test . . . 137
5.12 Modifying the design . . . 138
5.13 SCRUM . . . 138
5.13.1 Work estimation . . . 139
5.13.2 Motivation, psychology, and team work . . . 140
5.13.3 Remote vs on-location work . . . 140

6 Conclusions **142**

6.1 Problem solving . . . 142
6.2 Recommendations . . . 143
6.3 Our Contribution . . . 143
6.4 Further Work . . . 143

Appendices **145**

A Wireframes and UI Design . . . 145
B UML Diagram . . . 145

C Requirement Specification 145

D Large-scale user test & Usability test documents 145

A Wireframes and UI design **146**

A.1 Voter wireframe 146

A.2 Organizer wireframe 147

A.3 UI design organizer panel 148

A.4 Landing page wireframe 149

A.5 Landing page design 150

B UML diagrams **151**

B.1 Sequence diagrams 151

C Requirement Specification **155**

A Voter features 155

B Election administration panel features 155

C Ballot features 156

D Election features 157

E System features 157

D Test Documents **158**

D.1 Usability Test Documents 158

D.2 Large-scale User Test Document 172

Bibliography **177**

Acronyms

API Application package interface.

CDel Continuous delivery.

CDep Continuous deployment.

CI Continuous Integration.

CLI Command Line Interface.

CORS Cross-origin resource sharing.

CRA Create React App.

CRUD Create Read Update Delete.

CSS Cascading Style Sheet.

DOM Document Object Model.

DRY Don't repeat yourself.

ER Entity Relation.

ERD Entity Relationship Diagrams.

EVM Etherum virtual machine.

GHA GitHub Actions.

GUI Graphical User Interface.

HTML HyperText Markup Language.

HTTP Hyper-Text Transfer Protocol.

HTTPS Hyper-Text Transfer Protocol Secure.

ICT Information and Communications technology.

IDE Integrated development environment.

JS JavaScript.

JWT JSON web token.

NPM Node package manager.

OOP Object-Oriented Programming.

ORM Object-relation mapping.

ORMD Object Relational Mapping Diagrams.

PoC Proof-of-Concept.

PR Pull Request.

PWA Progressive web application.

SPA Single-page Application.

SQL Structured query language.

TCP Transmission control protocol.

TLS Transport Layer Security.

TS TypeScript.

tsc TypeScript Compiler.

UI User interface.

UML Unified Modeling Language.

UX User Experience.

VoIP Voice over IP.

Glossary

activity diagram diagram which is used to show how a system operate.

agile Is a continuous development and testing of a software system process.

backend Part of a software system that lies close to the data layer.

backlog A list of work tasks which needs to be solved to reach a bigger strategic plan.

ballot A series of candidates which can be chosen by a voter.

candidate One of several possible choices given during a ballot.

docker container technology for lightweight isolated virtualization.

Don't repeat yourself A software principle to avoid code duplication.

election A series of ballots, structured and organized by a election organizer.

election organizer A person or organization which can create, manage, and hold an election.

eligible voter A person that is eligible to vote in an particular election.

framework a supporting structure around which something can be built.

frontend Part of a software system that lies close to the user.

GitHub Cloud service for GIT.

HTTP Polling A technique where a client asks a server for changes in data by sending regular requests to retrive possible changes. This technique comes with additional overhead as the server must handle the request no matter if there is any changes to the data or not [120].

Minimum viable product A product state with the least amount of functionality to make the product usable [49] .

Object relation mapping Object–relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages[53]..

SCRUM Is an Agile process where the focus is to deliver business value in the shortest time..

universal design idea to design the society in a way so that anybody can participate actively, regardless of functional activity.

use case diagram Diagram which shows a system from the user point of view.

user interface design of what is shown to the user.

vote A complete and user chosen list of candidate(s) belonging to a ballot.

voter A person which can participate in an election.

websocket a data communication protocol which allows simultaneous communication channels in both directions over a TCP-connection.

Windows Subsystem for Linux WSL is a compatibility layer enabling Linux commands to be ran inside Windows 10 [97].

wireframe used to establish a structure of a page before visual elements and content is added.

List of Figures

2.1 CI/CD Pipeline. Image downloaded from: https://miro.medium.com/max/4000/1*TNJ7Rpr5G1OJHtKH-IBEFw.png 31

2.2 Usability Test Flow [Illustration] by Norman Nielsen Group. Image downloaded from https://media.nngroup.com/media/editor/2019/11/11/usabilitytesting101_final6-copy.png 34

2.3 An illustration of all phases in SCRUM. Illustration is downloaded from[72] 38

2.4 GitFlow release cycle. Image taken from: https://leanpub.com/site_images/git-flow/git-workflow-release-cycle-3release.png 42

4.1 Use-case diagram of the system 70

4.2 The "Create election" page with ballots 72

4.3 The "Create ballot" modal with candidates 73

4.4 The elections view displays all the elections an organizer has created 74

4.5 An election in progress with the end election confirmation dialog 75

4.6 Information shown to a voter when joining an election. From the top left, (1) the voter is accepted to the election, waiting to be verified, (2) verification happening, (3) upgrading the current window, (4) could not join because of missing verification code, (5) waiting for a ballot, (6) the election is closed, (7) election does not exist, and (8) verification code is invalid 76

4.7 Voting screen where a voter has the possibility to vote on a specific candidate. . . . 77

4.8 Network and communication infrastructure diagram 78

4.9 Simple illustration of join and verify 84

4.10 The join-election page. A voter needs to provide his/her email and an election code to join 84

4.11 Panel for an election in progress. From here, the election organizer can push the selected ballot. 91

4.12 Simple voting sequence 93

4.13 Container architecture 97

4.14 Entity relation diagram 98

4.15 Continuous integration routine for development 108

4.16 Continuous delivery routine for new releases 109

4.17 Autogenerated release 115

4.18 Scrum master work board preview 117

4.19 Scrum master issue/pull request preview 118

4.20 Generated sprint documets 120

4.21 Generated meeting documents 121

4.22 Output from navigation generation 121

4.23 The main flow for a voter, designed with wireframes. 122

4.24 Variants proposed for choosing how candidates should be displayed. "Row design" to the left. 123

4.25 All cases of displaying information of the ballot 123

4.26 Wireframes of all use cases for the organizer 124

A.1 All cases of displaying information of the ballot 146

A.2 Wireframes of all use cases for the organizer 147

A.3 UI design for Anovote organizer panel 148

A.4 Wireframes for landing and about page 149

A.5 Landing page design for `https://anovote.app` 150

B.1 Join and verification sequence diagram 153

B.2 Vote sequence diagram 154

List of Listings

4.1 The event handler implemented on frontend . . . 80
4.2 Showcase of acknowledgement on the backend. This event will be acknowledged with a *NotFoundError* . . . 81
4.3 Example verification URL . . . 86
4.4 Code snippet to initialize an socket room entity in ElectionService.ts . . . 89
4.5 Creation of socket room for election . . . 90
4.6 Push ballot logic from pushBallot.ts . . . 92
4.7 Event registration for submission of votes . . . 93
4.8 Error handling for vote submission . . . 94
4.9 Simplified version of verifyToken method . . . 95
4.10 Example definition of typeORM entity. The code does not include all fields and annotations . . . 99
4.11 Querying information with typeORM . . . 100
4.12 Definition of validation constraint for a unique election organizer . . . 101
4.13 A field with both custom and predefined restriction annotations . . . 101
4.14 Example of how we have stripped data that does not need to be exposed by API . . . 102
4.15 Class-transformer decorators used in election organizer . . . 102
4.16 A localized string in home/index.ts . . . 103
4.17 Full *ErrorCodeResolver*-class implementation . . . 105
4.18 Starting of development environment . . . 117
4.19 Re-building of development environment . . . 118
4.20 Start and build production with certificates . . . 118

4.21 Start and build production 119
4.22 Restart production 119
4.23 Generate documents for sprint 120
4.24 Generate documents for meetings 120

Chapter 1

Introduction

This chapter will introduce the background and problem formulation for this project. It will present the objectives for this thesis, and later share how we have structured this report.

1.1 Background

In 2019, Start Ålesund wanted a digital solution to perform their elections at their annual general assembly. Up until then the counting and registration of votes were done manually, which was prone to errors, and time and labour intensive. As a part of the thesis in the subject Webteknologi - ID102012, a PoC was developed. The solution worked during the demonstration, however, due to time limitations, the prototype had faults and missing functionality. This PoC gave a foundation for the project that we have chosen as our bachelor thesis.

1.2 Problem Formulation

Arranging an election requires the organizer(s) to print ballots, collect and count votes, and present the result. Doing these tasks analogue can be time consuming, labor intensive and prone to errors. It often requires repetitive tasks to be preformed by personnel. All these pro-

cesses can be simplified and enhanced by digitization. Our thesis aims to develop a digital voting solution that makes the process of holding an election easier, guaranteeing the integrity of votes and securing the anonymity of the voters.

1.3 Objectives

The objectives are as follows:

- Deliver a website for anonymous voting which support Progressive web application (PWA)
- The product satisfies the requirements for universal design.
- Develop and deploy a server and database architecture to support the application.
- An election organizer shall be able to create and organize an election
- An election organizer shall be able to add eligible voters
- Ballots could be received only by eligible voters
- Election organizers should be able to manage elections in real-time.
- Votes can be only be cast by eligible voters.
- A eligible voter should only be allowed to vote once
- Deliver a solution that is easy to maintain and is well documented.

1.4 Literature Survey

There has been done an extensive amount of research regarding electronic voting, voting schemas, algorithms and protocols. The International Association for Cryptologic Research (IACR) has made a list of all cryptography related articles dating from 1996 up until now [19]. This list consists of over 100 articles with the word "voting" in the title and over 240 articles with the word

"anonymous". We have simply not been able to read through all these articles. We have focused our attention on articles published within the last 5 years.

1.5 Limitations

Some limiting factors for this project is time constraints and the ongoing COVID-19 pandemic. The first and main limitation is time, as there is a great number of features to be implemented. The second limitation is the COVID-19 pandemic. The pandemic has forced the team to work in different environment and has made it harder to perform usability tests.

1.6 Structure of the Report

The rest of the report is structured as follows.

Chapter 2 - Theoretical basis: Describes the theoretical basis needed for this thesis

Chapter 3 - Method: Contains a description of the methodology and materials that are necessary for this project.

Chapter 4 - Result: Delivers the result from this thesis.

Chapter 5 - Discussion: Contains a discussion of the results we collected, an why we got them. This chapter will also describe some of our mistakes, and things we would have done differently.

Chapter 6 - Conclusions: This chapter will conclude the findings in this thesis.

Chapter 2

Theoretical Basis

This chapter will present the theoretical basis for this project.

2.1 Voting

Voting is a method for a group, meeting, or electorate (2.1.2) to make a collective decision or express an opinion [85]. Voting can occur in different ways, but it usually involves filling out a ballot (2.1.3) and casting the ballot via a voting system. There exist different voting methods [85]:

- **Paper-based methods** - Voters fill out their own selections/preferences on paper ballots. Either on a preprinted ballot or if possible, write out the name of their candidate (see 2.1.3).
- **Machine voting** - Electronic machines a voter can cast their vote with [86]
- **Online voting** - Voting performed via internet
- **Postal voting** - Voters are sent a ballot, which they can fill out and return by post
- **Open ballot** - Public voting. A method in which voters vote openly [55].

- **In person** - Votes can be carried in person if all eligible voters are present.

2.1.1 Electoral System

An electoral system or voting system is a set of methods and rules that determines how elections are organized and results calculated. When organizing political systems, governments are the organizers. Non-political elections may be organized by businesses, non-profit organizations, or informal organizations [29]. The rules for an electoral system regard when elections occur, who can stand as candidates, who is eligible to vote and how the vote can be cast, how the vote translates into an outcome, and other factors that can affect the result.

The electoral systems currently in use in representative democracies can be divided into two basic kinds [30]:

A **Majoritarian Systems** - In a majoritarian election, the candidate that attract the majority of most votes wins.

B **Proportional Representation Systems (PR Systems)** - The proportional Representation system are designed to allocate seats in proportion to the votes. The goal of using this system is to reflect the preference of the electorate. PR systems are not the most frequently used electoral systems in western democracies [30].

2.1.1.1 Plurality System

The simplest type of electoral system is the plurality system. Plurality voting is a system in which the candidate(s) with the highest number of votes wins and no majority is required. Plurality system is mostly known and referred to as *first-past-the-post* [29].

2.1.2 Elections

An election is a formal collective decision process for a population to select a person for a role or responsibility. Elections can also be in the form of accepting or rejecting a political proposition by voting [28]. An election can have the following characteristics [27]:

- **Suffrage** - The question of who may vote from the entire population.
- **Electorate** - Eligible voters for an election
- **Nomination of candidate** - A representative democracy requires nomination for a political office that is governed and very often mediated through a preselection processes by organized political parties.
- **Electoral system** - The system that converts a vote into a political decision. The primary goal is to tally the votes. Then determine the result on the basis of the tally.
- **Scheduling** - Fixed interval between elections for the elected to return to their voters and seek their mandate to continue in office.

2.1.3 Ballot

A ballot is an action or system of secret voting [22]. Each voter uses one ballot. In the simplest elections, a ballot can be a sheet of paper. The ballot should deliver which candidate the voter wants to vote on [8]. Usually governmental elections have the ballots preprinted, otherwise the voter must fill out the desired candidate itself.

2.1.4 Electronic voting algorithms

Sfîrnaciuc, Vasilescu and Simion [109] describes the following characteristics of an electronic voting system:

- **Eligibility** - just legitimate voters can vote and only be able to vote once
- **Fairness** - results should not be published early to avoid influencing the remaining voters
- **Vote-Privacy** - ballots and all events during the voting process should remain secret
- **Receipt-freeness** - a voter does not gain any information (receipt) which can be used to prove to a coercer that she/he voted in a certain way.
- **Coercion-resistance** - a voter cannot cooperate with a coercer to prove to him that she voted in a certain way
- **Integrity of the vote** - both voter verification; "I can check that my vote was submitted correctly" and public verification; "anyone can check that all recorded votes were counted correctly"
- **Correctness of counting** - the final tally should be an accurate count of the ballots that have been cast.

Different voting schemes have been proposed to comply with these characteristics. Mix nets, blind signatures, cryptographic counters, and central authority-based protocols are just a few examples. As described by Haines and Roenne [114], even the simplest of the algorithms are prone to bugs, and any attempts at implementation are prone to security vulnerabilities. Having time to implement and test the proposed solutions is therefor a technical, difficult and time-consuming task [114, 115].

2.1.4.1 Mix Nets as an Example of Electronic Voting Algorithms

In this algorithm, each voter encrypts their vote with a public key given by a decryption authority connected to the voting system. Only the decryption authority is able to decrypt the votes. The votes are then given to an organization that mixes all votes, often called a mixing authority. The purpose of mixing is to avoid the votes being traceable. The mixed votes and verification can then be passed to a sequence of additional mixers for added security. At the end of the mixing stage, the votes are passed back to the decryption authority to be decrypted and tallied [18].

This approach only needs one mixing authority to be trusted to work. Finding organizations that have the technical insight and resources to have appropriate uptime can be a demanding task [123].

2.1.5 Blockchain

Blockchain is a decentralized shared memory network that consists of a linked list of records. The records are what defines the blocks, and the linked list combining the records creates the chain [11]. Transactions in the network trigger the creation of new blocks, where every computer in the network are used to validate and agree upon the transaction before it is permanently stored on the chain.

Blockchains require a consensus mechanism to perform the distributed agreement process. Proof-of-work is the most adopted consensus algorithm [104]. The proof-of-work consensus algorithm enforces any one to add new blocks to solve a computation heavy puzzle. This computational task is what is called mining, and by solving the puzzle, the node has proven it has spent the computational resources required to add a new block. Solved puzzle are rewarded with cryptocurrency.

Cryptocurrency is a digital asset persisted in the blockchain. The currency is a decentralized digital asset which fuels the blockchain. The currency is only created through the consensus mechanism and is not created by any central authority [17]. The currency can be transferred inside the network to other nodes, at the cost of a network fee. Bitcoin is the most known blockchain, and the currency is also called Bitcoin.

Each block in the chain consists of a timestamp, transaction data and a cryptographic hash from the previous block [10]. Blockchain is created in such a way that data modification is almost impossible, as all subsequent blocks in the chain have to be modified [11]. All transaction data in the chain is publicly available.

2.1.5.1 Etherum

Etherum is a blockchain implementation that also contains a virtual machine. The virtual machine is called Etherum virtual machine (EVM) and is capable of executing arbitrary computation. Each node in the Etherum network holds a copy of the EVM state. Any node of the network can broadcast a request to execute a computation on the EVM. The state update of the machine is done through the proof-of-work consensus mechanism [42].

2.1.5.2 Etherum Smart Contracts

Etherum smart contracts are programs uploaded on the Etherum blockchain. Smart contracts are the building blocks of the evm, and any one can upload them to the network. Smart contracts is publicly available once uploaded, and procedures can be called on it when it is submitted on the network. A fee has to be paid to the network to execute and upload smart contracts [42].

2.1.5.3 DApps

DApps are decentralized application that has no central place of execution or authority. DApps can be created on the etherum blockchain using smart contracts. Applications created on the blockchain are accessible by every one [44].

2.2 Security

A fullstack software solution is vulnerable to several security exploits and attacks. We will describe some of them below.

2.2.1 SQL Injection

SQL injection is the most common attack on the internet [119]. It is done by exploiting weaknesses in SQL statements to gain access to SQL databases. The attack is easily prevented with prepared SQL statements. Prepared SQL statement is a technique where variables in the SQL statements are not passed in directly but are parsed first. Prepared statements are easy to understand, fast to write, and make the SQL database run faster than regular SQL expressions when using the same statement for different values in a query [121, 71]. Many ORM's use prepared statements by default when interacting with the database.

2.2.2 Cross Site Scripting

Cross site scripting is one of the most common attacks on the internet. It can occur if a malicious user is able to insert JavaScript into a form and upload it to the server, without validation or encoding happening first. The uploaded JavaScript can then be executed if the content is displayed on the HTML page without sanitizing the content first [16].

2.2.3 Cross-origin Resource Sharing

Cross-origin resource sharing (CORS) is a browser implemented security feature, where a website is by default not allowed to make requests to resources other than its origin. E.g., a website at domain-a.com cannot make a request to domain-b.com. This can be bypassed if domain-b.com adds a HTTP header specifying which websites are allowed to get content. This is implemented to protect the server from malicious third-party websites being able to access content and functionality which they are not suppose to [15].

2.2.4 Brute Force Attacks

Brute fore attacks is an act of performing trial and error attacks on login credentials, routes or security keys. This is done by issuing attacks with every possible combinations. A login form may be brute forced by inserting a known username, and programmatically submitting the form with a new password combination until it succeeds [138, 107].

2.3 Web Application

Web applications are software that runs on a web server. From the web server, a client device can access the software by requesting content from the server. A web browser is used for requesting content from the server and displaying it. Therefore, a web application can be reached from any device with an active network connection [87]. In this section, web application theory is discussed in detail.

2.3.1 Single Page App

A Single-page Application (SPA) is a form of a Web application that loads all the content it needs from a single web document load, i.e. the document is never reloaded and control is never passed between documents. To enable a single document to stay in control, while still allowing users to navigate with URLs, the SPA handles navigation between pages. Any state and other web application patterns is also handled by the SPA to give the user a functional and dynamic web page [69]. The most common approach for adopting SPA principles are using JavaScript frameworks. The most used frameworks are [73]:

- React.js
- Vue.js
- AngularJS

2.3.2 Progressive Web Application

Progressive web application (PWA) are web applications that use modern web browser API's and features to create a native app-like experience to web applications. Progressive web application are a useful design pattern for the web, and allow web developers to easily create an app-like experience for all devices without developing native applications for a specific device [60].

2.3.3 Universal Design for Web

Universal design is the design of buildings, products, or environments to make them accessible to all people. Regardless of age, disability, or other factors [80]. Universal design is required by regulations on universal design of ICT-solutions. Internet solutions must be designed in accordance with the success criteria in "Guidelines for accessible web content (WCAG)" [68, 32].

Universal design for web pages ensures a good, simple and accessible solutions for users with disabilities, promoting equal democratic rights, access to information and opportunity for everyone.

2.4 Design Patterns

Design patterns are established best practices for solving certain tasks in software development. We will describe some relevant patterns for this project.

2.4.1 Singleton

The singleton pattern is a creational design pattern, where you ensure that an object only has one instance and provides a global access point to the one instance. The singleton pattern aims to ensure that there is only one instance of an object, this is usually to control who has access to a shared resource. The singleton pattern also aims to have a global access point to the instance

that controls a resource, this is because it is very handy to be able to access the resource safely from everywhere in a system [74].

2.4.2 Module Pattern

The module pattern is defined as a way to both private and public encapsulation for classes. The goal of the pattern is to provide a wrapper for both public and private methods and variables, and protect these from leaking into the global scope. When using the module pattern, only a API is returned for the encapsulation, keeping everything private [50]. The module pattern is used as a key principal in Object-Oriented Programming.

2.4.3 Observer pattern

The observer pattern is a behavioral pattern, which facilitates for passing events in an application. The pattern aims to avoid unnecessary work by providing subscribers the ability to subscribe to a publisher. The subscriber will silently wait for and message from the publisher and then act accordingly. The publisher does not need to know anything about the subscribers other than that they want to receive notice when an event is fired. [54] [113].

2.5 Client-Server Communication

To make the client/server communication possible, some form of transmission protocol has to be established. Some relevant protocols will be discussed here.

2.5.1 HTTP

HTTP is a request/response protocol that requires a client to send a request for a document to a server which will reply with data as a response, usually in the form of a document. HTTP is a

stateless protocol [38, 122], thus the server is not required to hold any status or information of any client during the duration of a request[39].

The protocol has a set of request methods that indicates what action the server should take on the request. Common methods are:

- GET - Used to retrieve a resource from the server and should not have any side effects.
- POST - Used to upload or insert a new resource to the server.
- PUT - Used to update an existing resource, or create it if do not exist
- PATCH - Used to partially modify an existing resource.
- DELETE - Used to delete a resource.

The server response message contains information about the request. A response is required to have the following fields:

- **Status code and reason**
  Describes the result of a request. A status code of *400 BAD REQUEST* would imply that the client sent a request for a valid resource, but invalid or malformed data. A "200 OK" status code would imply that the request was successful.
- **Response headers**
  Headers are additional information of the response. The required fields are Content-type and length of the response. Content type tells the client what type of resource the server has returned (examples: text/html, application/JSON, application/xml) Length is the total size of the content body
- **Optional message body**
  The message body is optional, but required if the server returns a resource to the client.

2.5.1.1 HTTPS

HTTPS is an extension of the HTTP protocol by including security. It protects the communication between client and server, which ensures integrity and confidentiality. Transport Layer Security (TLS) is used as the protocol used in HTTPS and provides encryption, data integrity, and authentication[76].

2.5.2 REST API

A REST API, also known as RESTful, is an API that adheres to a set of defined constraints. For an API to be RESTful, it must satisfy all constraints[94]. The definition of a REST API is:

- **Client-Server**
  Messaging exchange must be a client/server architecture.
- **Stateless**
  All client requests must contain the required information for the server to understand the request.
- **Cacheable**
  Resources returned from the server must specify the cachability of the resource.
- **Uniform interface**
  The API should follow the principles of generality and provide a uniform interface.
  - *Resources must be identifiable in the request* - A HTTP REST API can use a URI to identify a resource.
  - *Must provide an interface for manipulating resources* - In HTTP, a HTTP method with a URI can be used to describe the action.
  - *The messages must be self-descriptive* - MIME types can be used to describe the type of resource

- **Layered**

  REST allows the system architecture to be layered. The API interface can be deployed on one server and storage on another. [62]

2.5.3 WebSocket

WebSocket is a communication protocol, which enables two-way communication over a single TCP connection. WebSockets is compatible with the HTTP protocol, which means that it works on ports 80 and 443. WebSockets can be used for interaction between two computer systems in real-time and provide less overhead than HTTP polling, by preventing asking the server for new data. WebSockets achieve this by having a standardized way for the server to send data to the client without the data being requested, and keeping the connection between the server and the client open [88].

2.5.4 Token

Since HTTP is a stateless protocol, the client needs to verify its identity on each request. This can be done by using tokens. A token is an object which represents the right to perform some action in a computer system.

Access token An access token is a token that contains the security credentials for a user, which is used by the computer system to keep track of who and which part of the computer system the user has access to [2].

2.6 Frameworks and Technologies

2.6.1 JavaScript

JavaScript (JS) is a high-level programming language that is used as a core technology for the World Wide Web. JavaScript is mainly used for client-side web page functionality, where it is popular to incorporate third party libraries. JavaScript uses the ECMAScript specification and most modern browsers have a dedicated JavaScript engine to execute the code in the user's device [46].

JavaScript is a dynamically typed language, meaning a variable can change datatype during runtime. Both Object-Oriented Programming (OOP) and functional programming is supported by JavaScript

2.6.2 TypeScript

TypeScript (TS) is a super set of JavaScript that adds support for static type definitions. The type definitions are only for development and are removed when the code is transpiled. TypeScript code is transformed into JavaScript via a transpiler, which transforms the TypeScript code into JavaScript, which is what browsers understands [77].

2.6.3 HTML

HyperText Markup Language (HTML) is a markup language used to define the semantics of a web document and is supported by all modern internet browsers. HTML has a range of tags along with attributes that define the semantics of a web page. HTML tags and specific attributes can be interpreted by screen readers to present the structure of the website [128].

2.6.4 CSS

Cascading Style Sheet (CSS) is used to create visual modifications to a HTML page in the form of styling. Together with JavaScript and HTML it forms the backbone of the World Wide Web.

2.6.5 SQL

Structured query language (SQL) is a programming language used for designing and maintaining data held in an database management system. SQL is especially powerful in regard to handling relational data [70, 106].

2.6.5.1 ORM

Object-relation mapping (ORM) is a technique for converting data between incompatible systems by using object-oriented programming languages. In an ORM, a "virtual object database" is created, which can be accessed directly by the object-oriented programming language [52].

2.6.6 Runtime Environment

A runtime environment is where the code of a program is being executed, and when the code is being executed, it is in a runtime state. In the runtime state, the code can send instructions to the CPU and access the computers memory and storage [63].

2.6.7 Virtual Machines

Virtual machine is the technique of virtualization or emulating of a computer system. Virtual machines provide the functionality to run multiple operating systems simultaneously on a physical machine. This reduces the need of running multiple machines to provide multiple systems. Virtual machines shares the resources of their host machine on which they run [83, 134].

2.6.8 Third Party Libraries

Third party libraries are used in almost all software projects. According to Papadopoulos [130] there are some great pro's to use third party libraries, but there are also some drawbacks that one needs to be aware of.

The main reason to use third party libraries is to utilize pretested and modular code, and to avoid reinventing the wheel. Others might already have solved the problem that you are facing. If the solution is already available and thoroughly tested, there is no need to do it all over again. Modular code makes it less coupled and easier to reuse.

According to Papadopoulos, the main drawbacks are dependency, risk of lacking support, dependency conflicts, and security issues. When using software that someone else wrote, the product becomes dependent on that software. If this software is not maintained, the software might not be able to run on new hardware or operating systems.

Using many different third party libraries might also cause conflicts between libraries, resulting in bugs that might be hard to resolve.

Any software, including third party libraries, are vulnerable to security issues. To mitigate this problem one have to choose libraries with care.

2.6.9 Localization

Localization (or language localization) is a process of adapting a translation to a specific country or region. Localization is most used in the adaption of translation in websites, video games and technical communication [100].

2.7 Continuous Integration, Delivery, and Deployment

To have software projects that have consistent code quality, fewer errors, and an automatic release system. Some common practices/philosophies in the software industry can be integrated in the development workflow to perform these actions automatically.

2.7.1 Continuous Integration

Continuous integration is an automated process that performs building and testing of software. It is a software development practice integrated into the workflow of a software project to continuously verify the code. The automated setup runs, tests, and builds the software continuously as developers submit code into a central repository, like GitHub. CI helps to detect bugs, maintain code quality, and validate the software throughout the project [92].

2.7.2 Continuous Delivery

Continuous delivery prepare the software project for release deployment. The currently submitted code for the project is deployed to a testing/staging environment before it is deployed to production. Deployment to production is a manual approval process. CDel extends CI by also performing load and integration testing, as well as UI testing. The goal is to always have a production ready release candidate[91].

2.7.3 Continuous Deployment

Continuous deployment is similar to continuously delivery except that the release to production is also automated.

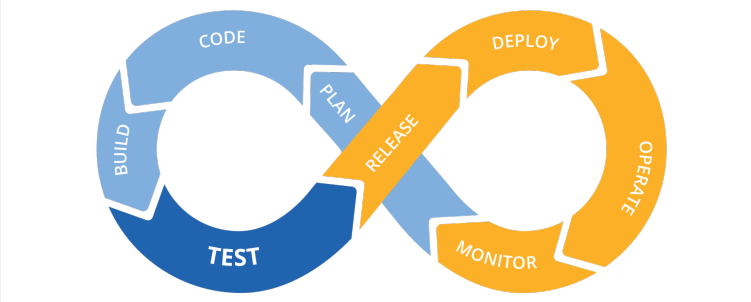


Figure 2.1: CI/CD Pipeline. Image downloaded from:
https://miro.medium.com/max/4000/1*TNJ7Rpr5G1OJHtKH-IBEFw.png

2.8 Tests

When doing engineering projects, tests are used to verify that the result will perform as expected. In software projects, tests can be classified into different subclasses, each with their unique features and use cases. We will describe some of the different types of tests. We will also briefly discuss a development methodology used in software projects that utilizes the potential in tests.

2.8.1 Unit Testing

Unit testing are tests performed on a specific unit. A unit is a function of a class or a stand-alone function. Unit testing ensures that a procedure outputs the desired results for a given input or handles false/wrong values without crashing.

2.8.2 Integration Testing

Integration tests are tests that test the implementation of software modules as a whole. Integration test has the purpose of ensuring that multiple modules interact with desired results [99].

2.8.3 Test-driven Development

Test-driven development is a method of writing software where tests are written before the implementation. Implementation should only be written once the test is covering the expected behavior of the implementation [45].

2.8.4 User Interface Testing

User interface testing are tests that test the visual presentation of an application. The tests ensure that the UI reacts or displays the information or components that it is supposed to do for a given action.

2.8.5 Prototype Testing

Prototype testing is a test method to analyze the prototype's quality, features, and other components. A prototype test should occur before a release or further development, so the team can determine the quality or get feedback. The feedback gathered from these sessions will point out deviance's in the product or software being tested [61]. Prototype testing can be performed at all stages of a product development, from wireframes, to design or new release. Prototype testing is a form of usability testing.

2.8.6 Usability Tests

Usability testing is a UX research methodology. The goal of usability testing is to identity problems in the design of a product or service, uncover opportunities to improve or learn about the target user's behaviour and preferences.

> In a usability testing session, a researcher (called a "facilitator" or a "moderator") asks a participant to perform tasks, usually using one or more specific user interfaces. While the participant completes each task, the researcher observes the participant's behavior and listens for feedback [111].

Facilitator The facilitator or moderator administrates the usability test and tasks for the participant. As the participants are performing the tasks, the facilitator observers the behavior of the participant and listens for feedback. The facilitator should ensure the test result is high-quality, valid data, without accidentally influencing the participant[111].

Participant The participant of a usability test should be a realistic user of the product or service being studied. The participant might also have a similar background to the target user group, or might have the same needs. The participants are encouraged to think out loud during the testing. The goal of this approach is to understand the participants behaviors, goals,

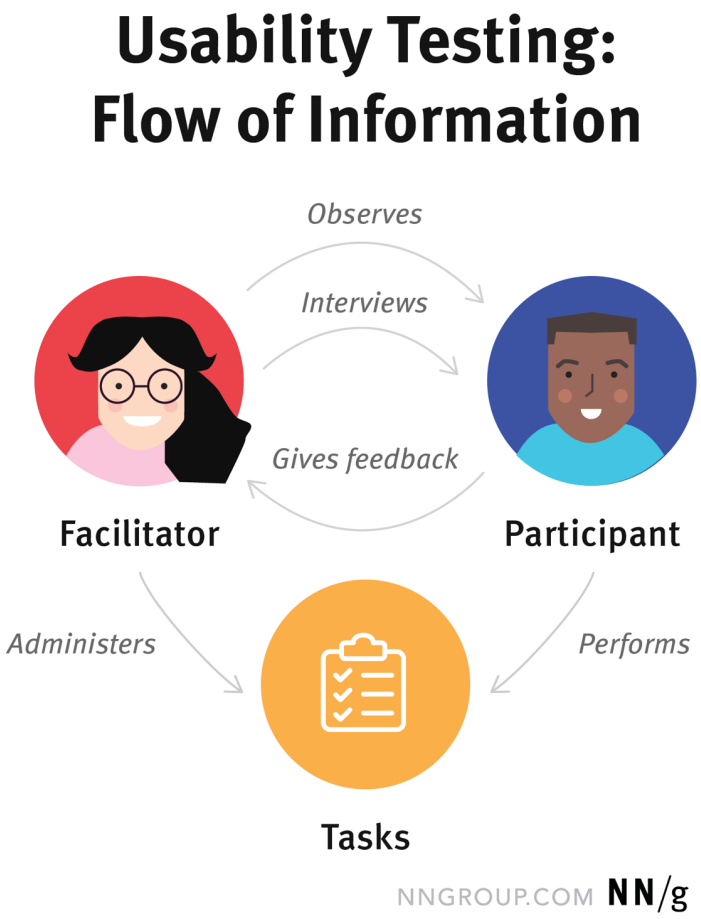


Figure 2.2: Usability Test Flow [Illustration] by Norman Nielsen Group. Image downloaded from https://media.nngroup.com/media/editor/2019/11/11/usabilitytesting101_final6-copy.png

thoughts, and motivations [111].

2.8.6.1 Tasks

The tasks in a usability test program should be very specific and something a realistic user of the product often will perform or more open for the user to explore. Defining which elements to test for by importance and prioritization can be done by using the 7 steps model [110]:

1. Determine the most important user tasks.

2. Discover which system aspects are of most concern
3. Group items from 1 & 2, then sort issues by importance to users and organization.
4. For each top issue, condense the information into a problem statement.
5. For each problem statement, list research goals.
6. For each research goal, list participant activities and behaviors.
7. For each group of goals, write user scenarios.

2.8.6.2 Qualitative vs. Quantitative

Usability testing can be either qualitative or quantitative. Qualitative testing is best for discovering problems in the user experience. Quantitative testing is preferred for collecting benchmarks.

2.8.6.3 Feedback

Information collected during a usability test session is called feedback. The information is expressed by the participant when interacting with the product or solution and is highly valuable. The two most valuable parts of feedback regard the understanding of why a participant thinks, does, or misunderstands the task of interacting with the product. Another feedback can be from observing what a participant is doing in real time [129].

2.9 Software Project Methodologies

Software projects are often organized differently than other engineering projects. One of the main methodologies of today to organize a software project is SCRUM.

2.9.1 The History of SCRUM

SCRUM and the agile methodology started to surface in the late 80's and early 90's, as a reaction to the way software project was organized at the time [136]. Projects would usually follow the waterfall approach, which tried to identify all problems and solutions in the early stage of the project. By the end of the project, the solution would be delivered as its entirety [125].

This approach had some serious drawbacks. Making the necessary documents to comply with the waterfall model would often take a lot of time. Meanwhile, the needs of the client could have changed, resulting in documentation being outdated even before any actual software development had started.

SCRUM tries to solve these drawbacks by restructuring the way a project is developed. This is done by clearly redefining and structuring how teams should work. Responsibility is split between roles, work is organized in predefined activities, and meetings, and teams are given tools in form of artefacts to aid the process of delivering an incremental product to the customer or client [133].

2.9.2 The SCRUM Team

SCRUM consists of a SCRUM team. The recommended size of the team has changed over the years and the recommended size today is between three and ten people [124]. The SCRUM team is divided into clearly defined roles, each with their own responsibilities.

2.9.2.1 Product Owner

The Product owner is responsible for the product backlog. The work of the Product owner may be delighted, but the responsibility remains with the Product owner. The product owner is one person and can not be a committee [133].

2.9.2.2 Development Team

The development team consists of everyone that contributes to resolving sprint tasks. The development team decides on how to split up tasks and organize them to deliver the product. The team should not be subdivided and tasks should be split among all members. Even though some members might have special skills, the responsibility of fulfilling tasks lies within the team as a whole [133].

2.9.2.3 SCRUM Master

The SCRUM master is responsible for promoting and supporting the scrum process. This is done by informing everybody, both the team and within the organization about the SCRUM process and ensuring that the team complies with SCRUM values and ideology. The SCRUM master also helps the team follow the practices and processes that they have agreed on [95].

2.9.3 Activities in SCRUM

2.9.3.1 Sprint

Sprints are the period where the development team conducts their work. In this period of time, a sprint goal should be set and a subset of issues from the backlog is added to the sprint backlog. Additional work should not be added during a sprint and the quality of the goals should not decrease. The scope may be clarified and renegotiated between the Product Owner and the development team as new insight is gained. According to the Scrum Guide 2017 [133], sprints should have a duration limit of no more than four weeks.

2.9.3.2 Stand-up

Stand-ups is a daily routine in SCRUM. The goal is for the teams to inform each other about the progress of the sprint and possible uncertainties or hinders that might block or otherwise slow

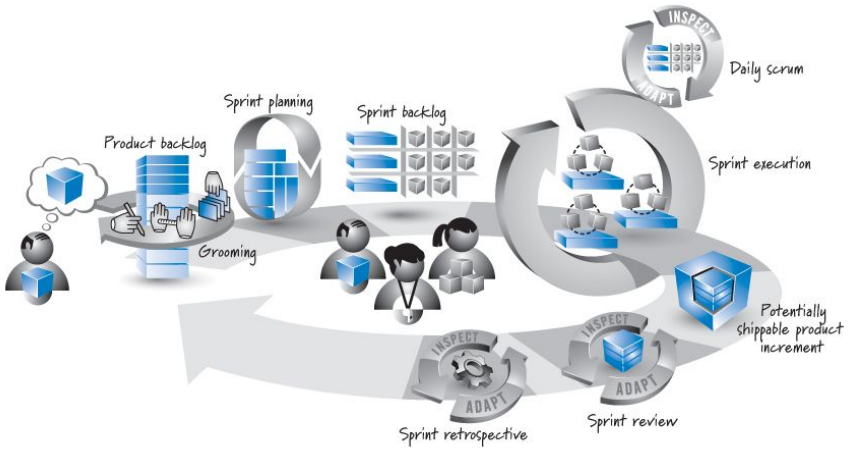


Figure 2.3: An illustration of all phases in SCRUM. Illustration is downloaded from[72]

down progress. The time frame for the stand-up should be no more than 15 minutes and is only an informative meeting. Findings during the meeting might lead to new meetings where parts of the team can discuss and solve the problems disclosed during the stand-up.

2.9.3.3 Sprint Planning

Sprint planning is the process of finding out what can be delivered within the upcoming sprint and what work needs to be delivered for the goal to be achieved. The planning is performed by the entire scrum team. The planning is time boxed. A sprint of four-week time span should not use more than eight hours for planning. Shorter sprints should have a smaller time box for planing.

2.9.3.4 Sprint Review

During the sprint review, the team inspects what has been done to increment the product since the last sprint review. The product backlog can be adapted to the findings during this meeting. The sprint review is supposed to be an informative meeting and not a status meeting.

2.9.3.5 Sprint Retrospective

The Retrospective is a chance for the scrum team to inspect itself and find possible improvements that can be done to further elevate the team. The team discusses the actions and behavior that has helped the team, and what has contradicted or decreased the team effort. Findings can be concluded into actions that the team can plan to implement or focus on. The purpose of the Sprint Retrospective is to: [133]

- Inspect how the last Sprint went regarding people, relationships, process, and tools;
- Identify and order the major items that went well and potential improvements; and,
- Create a plan for implementing improvements to the way the Scrum Team does its work.

2.9.4 SCRUM Artifacts

Scrum consists of several artifacts. Artifacts are concepts and tools that aim to guide the SCRUM team in reaching their goals. Below we discuss some relevant artifacts.

2.9.4.1 Product Backlog

The backlog consists of all features, enhancements, tasks, bug fixes, or other work that needs to be accomplished or resolved to increment the product. The product owner is in charge of organizing and keeping the product backlog up to date [90] [117]. Tasks that are not accomplished during a sprint can be moved back into the product backlog.

2.9.4.2 Sprint Backlog

The sprint backlog consists of all tasks for a given sprint. The content of the sprint backlog is chosen by the development team in cooperation with the product owner. Chosen tasks are then subdivided into smaller sprint items that the team can take action upon. [117]

2.9.5 Estimating work

To plan a software project, the workload of each task has to be estimated. Traditionally, estimates have been given in a time format; days, week, and months. Agile teams have transitioned into using story points as an alternative to aid the process. Story points shall indicate the complexity, amount of work, and risk or uncertainties of solving the task. Points can be given, e.g., by the Fibonacci numbers (1, 2, 3, 5, 8, 13, 21). The team decides on a baseline task which other tasks are measured against. If a task exceeds a certain point threshold, splitting up the task into more manageable pieces should be discussed [132].

During the process of deciding story points, the development team can play planning poker. The task at hand is briefly discussed while each team member makes an individual estimate. Then all team members present their estimates, usually in the form of a card with a number. If all agree, the score is settled. If the span of the presented scores is large, it can lead to further discussion before the team plays a new round or decide on an appropriate score.

The number of story points a team is able to conclude at the end of a sprint is called the team velocity and is unique for each development team and each project.

2.10 Version Control and Code Management

2.10.1 Version Control

In software development, version control is the act of managing changes to the code base or documents. It is also known as revision control, source control, or source code management. [82]. There exists different systems to handle version control. Some popular alternatives are Subversion, BitKeeper, and Git.

2.10.1.1 git

Git is the most popular version control system used today. Development was started by Linus Thorvald, but quickly grew in popularity, primarily because it was used in combination with the development of the Linux kernel. Today, git is developed by a community of developers and are distributed as open source. [33, 34].

Code is organized in repositories, which again can be split up into branches where individual features or fixes can be developed in parallel. A repository can be uploaded to a git server for backup and ease of collaboration. Popular git server providers are GitLab, Bitbucket, and GitHub.

2.10.1.2 Git flow

Git flow is a branching strategy model for Git, created by Vincent Driessen [108]. The key benefit of using Git flow is that it enables parallel development. The collaborators on a project can isolate new development from finished work by using different branches. New development is done in feature branches and is only merged back into the main branch when the developer(s) agree that the code is ready for release [43].

Figure 2.4: GitFlow release cycle. Image taken from: https://leanpub.com/site_images/git-flow/git-workflow-release-cycle-3release.png

2.10.2 Semantic Versioning

Semantic versioning is a way of organizing the versioning of an application. Semantic visioning is a proposed rule of how and when to increment an application version. Versions are indicated by MAJOR.MINOR.PATCH (e.g. 1.0.3) where MAJOR updates are breaking changes that are not backwards compatible with earlier releases, MINOR changes indicates feature updates that are backwards compatible with release and PATCH indicates fixes that are backwards compatible.[131]

2.11 Development Environment

2.11.1 Integrated development environment

An Integrated development environment (IDE), is a tool in software development that usually facilitates a complex source code editor, automated building of code, and an advanced debugger. Modern IDE's also include compilers, interpreters, and intelligent code completion, which help speed up the process of software development [41] [40].

2.11.2 Source-code Editor

A source code editor is a text editor, which is designed for the editing of source code. The source code editor can either be a standalone application, or integrated in an IDE [75].

2.11.3 Linter

A linter is a tool used for code analysis, which flags code errors, bugs, style errors, and code that do not follow a specified rule-set [48].

2.11.4 Code Formatter

A code formatter is a tool used in software development that updates how source code is displayed in a source-code editor. A code formatter is not a functional need for the source code to execute properly, but a tool which makes the source code more readable and consistent [13].

2.12 Graphical Design Principles

To create a functional and useful Graphical User Interface (GUI), there has been established different design principles. We will describe some of the principles relevant to this project.

2.12.1 Don Norman's Design Principles of Interaction Design

Regarding web and interaction design, Donald Norman provides six key principles to abide by [25, 23]:

- **Visibility** - The functionality of an object needs to be visible to the user.
- **Feedback** - Every action needs a reaction. The user must receive some feedback on the status of the users action.
- **Constraints** - Restrain the users allowed actions.
- **Mapping** - There should be a relationship between controls and their effect.
- **Consistency** - Similar actions should have similar design.
- **Affordance** - The object should signal what kind of action it represents

2.12.2 Mobile-first Approach

Mobile-first approach refers to designing and developing application and software that is fully compatible, and designed to work with smaller screen sizes. This approach ensures that the design is flexible and effective on smaller screens [93].

Chapter 3

Materials and methods

This chapter describes all materials, i.e., software and hardware, and methods that were used while conducting the project thesis. We will also describe how the project was structured and developed. It concerns everything from the organizational to the development environment. This chapter will also include what languages and utilities that have been used.

3.1 Organization

There are two separate organizational parties for this thesis. In this section, we will describe their roles.

3.1.1 Project Group

The project group consists of students from NTNU in Aalesund, Emil Elton Nilsen,
Sander Hurlen Olsen, Christoffer Andersen Traeen and Steffen Holanger. The students act as both the client and contractor in the project, driving the interests of the project forward.

3.1.2 Supervisor

During the thesis, we have had bi-weekly scheduled meetings with our supervisor from NTNU in Aalesund. The role of the supervisor is to supervise the progress and assist in technical discussions.

3.1.3 Former client

In 2019, Start Aalesund was the client and test group for developing this application. They are no longer a part of the thesis.

3.2 Project planning

3.2.1 Preliminary report

In parallel with the course IF300114 - Engineering systems and systems development, a preliminary report was written as an obligatory assignments. The goal of the preliminary report was to plan the project and was written during the planning phase of the project

3.2.2 Gantt Chart

In the planning phase of the project, a gantt chart was created to organize where in our timeline the different phases of the project should take place. This includes when planning, software development, and finalizing of the project should occur. The different sprint was also planned in the gantt chart.

3.2.3 Risks

A risk assessment matrix was used to analyze what sorts of risks that could occur during the project, and how severe their impact would be.

3.2.4 Requirement Specification

A requirement specification was created as a description of the system (see appendix C), and includes:

- Voter features.
- Election administration panel features.
- Election features.
- Ballot features.
- System features.

3.3 Project Methodology

During every stage regarding our bachelor thesis, we have followed the SCRUM working principles. We have made the following adaptions of SCRUM to fit our project.

3.3.1 Roles

Since the project did not have an external customer or client, the product owner role was vaguely defined. This role has been blended in as part of the development team, and therefore the product owner responsibility has been a constant discussion within the team. One of the team mem-

bers was assigned to keep the product backlog updated, but all team members contributed to adding tasks throughout the project.

Due to the limitations of the project resources, we have not had an explicit SCRUM master. This role has also been blended in as part of the development team's responsibilities. No one was given and explicit delegation of the SCRUM masters responsibility. We handled this role as a community effort, where we as a team took responsibility for the weekly actions and management of the team.

3.3.2 Sprints

By recommendation by our supervisor, we decided to run sprints lasting one week. The decision was made due to the short development period of the project (five months), and the inexperience of the team. By having shorter sprints and thereby a higher meeting frequency, the project could adapt quicker. It was believed that finding and stabilizing the teams velocity would happen quicker with shorter sprints. Appropriate time boxing at sprint meetings was believed to be better.

Review, retrospective and planning meetings have all been held on Fridays. The combined time frame for all meetings has been from 08.00 - 12.00. Meeting with our supervisor every second week, has also been part of this time frame and has usually come as an addition to other meetings, blending in with the retrospective and planning.

3.3.3 Stand-ups

Stand-ups were held daily, after lunch. After lunch was chosen for a few reasons. First of, having the stand up after lunch would allow the team to start the day with their current task, get an insight in their task(s) and helping to prepare them for the stand up. Having the stand-up later in the day would also ensure that all members were present. Lunch, just like the stand up, is an interrupting event during the day. Chaining these interruptions could increase productivity

[135].

3.3.4 Product Backlog

Any features, bugs, or refactoring tasks have been reported as issues on GitHub. All issues have been connected to a common project board and appropriate labels (3.4.4) to categorize the tasks have been applied; e.g., bug, feature, chore. The tasks have also been labeled with a priority ranging from high to low. Later in the project we also introduced a "nice to have" label. All issues were put in a backlog column on the project board.

3.3.5 Story Points

Even though SCRUM does not imply any specific technique for estimating work [126], we have chosen to use story points.

We choose this method as it is considered an industry standard and a widely adopted practice. All team members had prior experience using story point estimation. We have used two different techniques for estimating points; brief discussion and planning poker.

At the start of the project we would discuss the task and give an estimate on what score the task should have. Later in the project we transferred to use SCRUM planning poker for estimating the task. The estimation strategy is discussed in section 5.13.1.

3.4 Version Control and Code Management

We have used Git and GitHub as our system for keeping track of code changes. We have primarily divided our project into three repositories; one for the frontend implementation, one for the backend implementation and last one for organizational documents. The frontend repository consists of any code that is ran directly by the user (eligible voter or election organizer), while the

backend consists of code that is closely related to data handling. The organizational repository has held meeting notes, diagrams and planning documents.

We have implemented the GitFlow branching strategy for our development repositories. Any new features or fixes had to be reviewed through a pull request. Necessary changes had to be made before the changes could be merged into the project and branches related to the work was deleted.

3.4.1 GitHub

GitHub is a code hosting platform for version control and collaboration [36]. It enables us to collaborate on projects, and provides essentials like *repositories*, *branches*, *commits*, *issues* and *Pull Requests*. GitHub have support for organization features to have a single organization were all our repositories can be stored.

Workboard By using GitHub, we could utilize their project board feature. GitHub's project board made it possible to track all issues and pull request's from our three main repositories: *frontend*, *backend* and *org*. By using this feature, we could easily keep track of sprints and and the overall project development. The project board supports well-known agile development methods and SCRUM. As our main repositories were linked to the work board, we could utilize GitHub's work board automation. New issues and pull requests would automatically be added to the board. Automation between columns was configured to move issues and pull requests to certain columns based on state. A Google Chrome extension was also developed to enhance the SCRUM experience using GitHub project boards, see 4.16.1.

3.4.2 Issue Tracking

For creating user stories, reporting bugs, features, or other tasks, we used the issue functionality built into GitHub. Issues are connected to the repository it was created for.

3.4.3 Pull Requests

Opening pull requests, let us review and comment suggestions to others code before merging the branch into the proposed branch. Pull requests need to be reviewed and tested before merging, forcing us to review changes.

3.4.4 Labels

Labels is a feature provided by GitHub to be able to annotate pull requests and issues. We used labels to organize and give additional meaning to a pull request or issue. It is possible to create and modify labels to your needs. Some of the labels we used was be *Bug, Feature, Refactor, (Story) points:[Fibonacci sequence til 21] and priority:[high, medium, low]*.

3.4.5 Semantic Versioning

We have used recommended standards of labeling our versioning in accordance with Preston-Werner [131] and in conjunction with the GitFlow toolkit. This was done by creating release branches that then were tested before merging into the main branch.

3.5 Development Tools and Applications

We have used different software programs and tools during development. Below we will describe the most important and how they were used.

3.5.1 Visual Studio Code

Visual Studio Code is a free source code editor developed by Microsoft for Windows, Linux, and MacOS [84]. Visual Studio Code has support for debugging, syntax highlighting, intelligent code

completion, snippets, code refactoring and embedded Git. It also lets you install extensions to create a better development environment suited for your you or your team. All team members have used Visual Studio Code for developing our application.

3.5.1.1 ESLint

ESlint is a tool that identifies and delivers reports on patterns in JavaScript/TypeScript code. It can be configured to either show warnings on linting rule sets not followed, or even compilation errors when linting rule sets are not followed. ESLint also automatically fixes most errors on save [26].

3.5.1.2 Prettier

Prettier is a tool that automatically formats source code after a specific ruleset and programming language. Prettier supports formatting for most web-related programming languages like JavaScript, HTML, and CSS, as well as JSON and YAML [59].

3.5.2 Nginx

Nginx is an HTTP server with reverse proxy, mail proxy, and generic UDP/TCP proxy support [51]. We have used Nginx to serve our static website documents (HTML, CSS, and JavaScript) as well as a reverse proxy for our backend API and WebSocket server. It was also responsible for serving the TLS certificates for HTTPS.

3.5.3 DataGrip

Data Grip is a database management IDE developed by JetBrains. It allow for managing multiple databases through a graphical User interface. Data grip has been used to view and edit data stored in our Postgres database while developing [20].

3.5.4 Postman

Postman is an HTTP API testing application that enables developers to create collections of API endpoints. It includes support for managing, executing, and testing API endpoints. Postman also has support for collaboration where a team can manage and collaborate on a collection of API endpoint's for an application. Postman can be used as a CLI tool as well for testing endpoints in a server environment without a graphical user interface [1]. We used Postman in collaboration to test our REST API during development, without the need of a frontend implementation.

3.5.5 Adobe XD

Adobe XD is a vector-based user experience and interface design tool for web applications and mobile apps. The software is developed by Adobe Inc [3]. Since some of our group members were confident with using Adobe XD it was a natural decision to use it for designing various parts of UI. It supports wireframing and easy to create click-through prototypes.

3.5.6 Adobe illustrator

Adobe Illustrator is a vector-based application which is used to create and edit vector graphics and is developed by Adobe Inc [98]. Adobe illustrator was used to create the Anovote logo.

3.5.7 Visual paradigm

Visual paradigm is a UML utility for creating various diagrams and providing modeling support. Visual Paradigm supports both Entity Relationship Diagrams (ERD) and Object Relational Mapping Diagrams (ORMD). ERD is used to model the relational database. Visual paradigm also supports the creation of sequence, use-case, class and activity diagrams.

3.5.8 Discord

Discord is an free Voice over IP (VoIP) application and digital distribution platform designed for communication between users in channels by using text, media, video, or sound [24]. Discord also supports code snippets for easy sharing of a code-related question. The team has used Discord mainly for communication through chat and voice calls.

3.5.9 Overleaf

Overleaf is a cloud-based LaTeX editor which supports real-time collaboration [101] and editing of LaTeX documents on the web. Overleaf was used to write the thesis for the project.

3.5.10 Zotero

Zotero is a reference management tool used to manage literature references and bibliography. Zotero can be used as a browser extension and/or as astandalone software. Zotero was used to manage all references for the project and thesis.

3.5.11 Typora

Typora is a free markdown editor for editing and creation of markdown files. Typora has been used to create and edit our documentation files.

3.5.12 Docksify

Docsify is a JavaScript application running on a website that creates documentation websites from markdown files. We have used Docsify to make our organizational documents in our organization repository available on `https://docs.anovote.app`. Navigation bars can be displayed by creating _sidebar.md files in directories where navigation is wanted. We have created

a tool that auto-generates these for us (see 4.16.4). The page content is served using GitHub pages.

3.5.13 Framework and libraries

To be able to build our application, we have used a range of third-party libraries and frameworks. Below we describe some of the most important.

3.5.13.1 React

For the UI of the system, we decided on using react as the UI framework of choice. React is a well-established UI framework and has extensive support for TypeScript. React also makes it very easy to reuse UI components and control the state of the UI components [64].

Create React App is a program to initialize a React project with the recommended settings for developing and building a web application. Create React App (CRA) is developed by Facebook and can be started with different templates to fit individual project needs, e.g., TypeScript template [14].

Ant Design For helping us create a more robust UI, we have used the react component library Ant Design. Ant Design provides a range of basic React components, which is the basis of all web sites. Using ant design meant we saved time on creating components, which we spent on more complex problems [4].

React Beautiful Drag-n-Drop A component library for creating drag and drop logic. Used to reorganize ballots and elections [5].

React Router A controller that is used to control which view the user can see based on the browser URL [66].

react-i18next A internationalization framework that makes it possible to implement several language packages to a react app [65].

3.5.13.2 Internet Access and Communication

expressJS Express is a web framework for server-side applications that provides a set of features to create web applications and craft web APIs. Express is unopinionated and leverages middleware to create web applications [31]. Middlewares in express is a thin layer that handles an incoming HTTP request. A middleware can either terminate an Hyper-Text Transfer Protocol (HTTP) request or pass it to the next middleware layer. By utilizing middlwares, we can create reuseable code that can be plugged into a collection of/or a single endpoint to run on every http request.

axios Axios is a HTTP request handler for the browser and nodeJS [7]. It is used to send HTTP requests from frontend to backend.

HTTP status codes A library with enums for all HTTP response codes [127].

Socket IO Provides functionality for bidirectional real-time communication between a client and a server. The protocol enables client-server communication over web sockets,but can fallback to http polling if necessary [102]. It makes use of the observer pattern (2.4.3) and utilizes events which the client and server subscribe to. Events emitted from a client can also provide an acknowledgment callback to get notification from the server. The acknowledgment mechanism can be used to ensure the sender that the receiver has received the event.

3.5.13.3 Data Handling

TypeORM TypeORM is as an ORM for TypeScript and JavaScript. It is responsible for mapping JavaScript classes to relational entities and vice versa. Entity columns are managed with anno-

tations on a given entity class. TypeORM is also responsible for generating database tables with appropriate types, constraints, and relations.

class-validator A framework to validate fields on a class instance. Field criteria(s) are set using annotations. A range of predefined annotations are provided by the library and the ability to create custom ones [79].

class-transformer A transformer library is used to convert and transform class/object instances from class instances to standard JavaScript objects and vice versa. Used to exclude fields from objects before passing them along. Class fields that need transformation are annotated [78] or can be mapped to another class/object by their property names.

All libraries used for data handling are developed by TypeStack.

3.5.13.4 Security and Authentication

bcrypt A hashing library [9]. It was used to hash passwords.

jsonwebtoken A library for creating and validate JSON web token (JWT) [6]. JWT is a from of access token that can be used to authenticate users of a system (2.5.4. All authorization on the server was using JWT.

helmet A middleware library for changing http headers aiming to make the application more secure [37].

3.5.13.5 Other

html2canvas A library for creating a canvas out of an HTML page. It is used to create the image output for the result report PDF [137].

papaparse A library for parsing CSV files [56]. It has been used to parse CSV files uploaded when adding eligible voters to an election by CSV.

date-fns A date formatting library for handling dates in JavaScript applications [21]. Handles the date objects on the backend and frontend, it makes use of JavaScript's date object but provides additional features for manipulating dates.

pdfobject Enables PDF embedding into a HTML object with JavaScript [118].

jspdf Library for client-side generation of PDF's [116]. Has been used to generate the PDF for the election results PDF. The described libraries and frameworks in this section were chosen to be used in our project due to their popularity, reliability, good documentation, and team members previous experience.

3.5.14 Runtime Environments

3.5.14.1 NodeJS

NodeJS is a JavaScript runtime that allows us to run JavaScript outside the browser. This makes it possible to run JavaScript in a server environment. The node standard library contains the necessary libraries to develop an asynchronous HTTP server. NodeJS also supports the use of packages from NPM registry to be able use open source third party libraries.

Yarn Yarn is a package manager for NodeJS and consumes packages from the NPM registry. NPM is the default package manager that comes with NodeJS, but was replaced in favor of Yarn due to its parallel handling of packages and caching mechanism. Yarn is responsible for all dependencies in our project, both frontend and backend.

3.5.14.2 Deno

Deno is a new JavaScript runtime for running JavaScript outside the browser. It is created by the same person that created NodeJS and had its first stable release in 2020. It was created to fix some core issues with NodeJS and provide native support for TypeScript. Deno runs as a single executable and incorporates its own code formatter, package handler, typescript compiler and bundler. Deno was chosen as our JavaScript runtime because of its new take on server side JavaScript with the built-in tooling and the benefit of a built-in TypeScript compiler.

3.5.15 Docker

Docker and docker-compose was used to containerize our application. By using docker we eliminating the need built-in for installing special software on a physical server (apart from docker). Each component of the application was divided into its own container. Using docker ensured that all of the members worked on the same environment, mitigating any environment dependency issues.

3.5.16 Database

For storing data in our system we choose PostgreSQL, as we wanted a feature rich and reliable relation database. PostgreSQL is a free and open source relational database manager [58]. The database was ran in its own container using docker. Setting up the database was done within a docker-compose file and the database tables management and CRUD operation was done by TypeORM (3.5.13.3).

3.5.17 Preprocessors and Transpilers

Preprocessors and transpilers are tools used in software development to transform or alter code. Preprocessors are software that processes a given input data to produce an output used by an-

other program. A transpiler is a software used to transform one language to another.

3.5.17.1 TypeScript Compiler

TypeScript Compiler (tsc) is an executable that lints and/or transpiles TypeScript to JavaScript that the browser or NodeJS understands.

3.5.17.2 JSX

JSX is a syntax extension of JavaScript. JSX is a template language with full support for JavaScript. It is the preferred syntax for defining React elements. JSX allows us to write rendering logic inside the UI components as they are tightly coupled. This includes data preparation for display, event handling of user actions, and state changes.

3.5.17.3 SCSS

SCSS is a preprocessor scripting language that is compiled to CSS. It adds more functionality to CSS development by including support for variables, mixins, functions, nesting, and seperation of files. SCSS was used to create CSS for our frontend. SCSS allowed us to create a more structured CSS code base.

3.5.17.4 TruffleSuite

TruffleSuite is a collection of application and utilities to create and test blockchain application on Etherum. It provides a framework to develop, test and deploy smart contract to a etherum blockchain. It also includes a application to host a etherum development blockchain for testing on your local machine. Network fees, initial accounts with ether and mining can be configured for the particular chain before setup.

3.6 Programming, Markup and Scripting Languages

We have used a variety of different languages to develop the product. We will describe some of the languages and how they where used.

3.6.1 TypeScript and JavaScript

We used TypeScript as the primary programming language for both the frontend and backend of the system. TypeScript enabled us to create a web page, while keeping control and quality of the code high.

We have used both OOP and functional programming techniques when developing. For the most part we would use OOP as it utilizes the module pattern (2.4.2) and provides structure to the code. Functional programming were used to create simpler helper functions and to make React Components (3.5.13.1), as this is recommended by the developers of React.

3.6.2 HTML

HTML has been used to create the markup for our web frontend.

3.6.3 SCSS

SCSS is a superset of Cascading Style Sheet (CSS) and provides additional functionality. See section 3.5.17.3 for more details on SCSS.

By using classes and element selectors in CSS, we can target specific elements to alter the visual look, position and scale. CSS classes are reusable which lets us keep the code DRY.

Animations are also supported in CSS and we have used it to enhance user feedback on certain elements. A responsive website is created with CSS by altering the style depending on the screen size. This makes it possible to deliver a product for all devices and screen resolutions.

3.6.4 PostgreSQL

We have used PostgreSQL which is a dialect of the SQL language. PostgreSQL was mainly chosen because it was well known to the team, and it integrates well with other technologies we are using.

3.6.5 Yaml

Yaml is a data-serialization language, with focus on human-readability. Yaml was used to write the docker-compose and GitHub workflow instruction files.

3.6.6 JSON

JSON was used to write most config files, i.e., ESlint configuration. It has also been used for data import and export. It was chosen because it is the standard for most config files and data handling in JavaScript/TypeScript.

3.6.7 Markdown

Markdown is a markup language used to create formatted text. Markdown was used to write sprint report and meeting reports. Markdown was also used to create any text related on GitHub.

3.7 Quality Assurance

3.7.1 Jest

Jest is a JavaScript testing framework maintained by Facebook. The framework focuses on simplicity and support for large web applications [47]. We chose to use Jest on both frontend and backend, as Jest is already configured in Create React App. Some of our group members had also previous experience with working with Jest.

React Testing Library Testing the frontend code requires a testing library for testing UI components. React testing library provides light utility functions and encourages better testing practices. The utilities this library provides facilitate querying the DOM in the same way the user would [67]. We chose this testing library over others because of its simplicity.

3.7.2 Prototype Testing

We used prototype testing to get an early indication on the first design/wire-frame draft. Adobe XD was used to create the prototype tests. The tests were used to get an indication on how our initial design was received by users of the application.

3.7.3 Usability Testing

As described in 2.8.6, usability tests are used to see how a user interacts with the application. We utilized this testing method with a qualitative approach, focusing on getting feedback from one to one session. We focused on the qualitative approach to gain information on how users interact with the application.

3.7.4 Large-scale User Testing

We performed large-scale user testing to see how a user group worked with the application. A large-scale user test is a formal process of letting inexperienced users act on the application. The goal of the test is to spot difficulties for the contestants and see if the system is capable of handling the traffic.

3.7.5 Unit Testing

Unit tests were written for both frontend and backend. The would test functional requirements for the unit a test was written for. Unit tests were run on the developer's machine and on pull requests.

3.7.6 Integration Testing

We used Postman for integration testing on the API endpoints. These were run on a development server on pull requests to develop/main. The tests were written in the postman application for different HTTP endpoints.

3.7.7 GitHub Actions

GitHub Actions (GHA) is an automation CI/CD tool provided by GitHub that enables workflows to be run on predefined events [35]. We had different workflows that are automatically run when creating a PR. The workflows was configured with *yaml* (3.6.5) files in the respective repository under a .github folder, which is automatically parsed by GitHub.

3.7.8 ESLint and Prettier

On the local development environment we would have two tools that would support our process. ESlint (3.5.1.1) was set up to catch typical errors and code style violations. Prettier (3.5.1.2) would help format the code to a common rule-set. All members of the development team was using the same IDE, with shared settings, ensuring that formatting and linting was always ran on file save.

3.8 Environment

The environment configuration for our application changes depending on the state it is running in and its environment. e.g. locally or on a server. The states can be testing, development, or production. To manage these different states, we have used configuration files to quickly and easily allow the change of state variables.

3.8.0.1 Environment Variable Files

Environment variable files have been used in our environment setup to provide configuration options for our application without hard coding values in the source code. Environment files can contain information about ports, security keys, credentials, paths, etc. Environment files are named by a .env.

3.8.0.2 Tools

Some group developed tools have been created to streamline the management of any given environment and simplify the process of doing so. These tools are discussed at 4.16.2.

3.9 Product Design

3.9.1 Wireframing

We used wireframing for designing the website at a structural level. A website wireframe is a page schematic and a visual structural guide of a website [89]. By designing with wireframes early in the development process, we could focus on the structure of the page before visual elements and content was added. The use of Adobe XD (3.5.5) enabled us to create prototypes of the wireframes to test the structure.

3.9.2 UI Design

After creating wireframes, we could create mock-ups for the User interface (UI) design. UI is the design of user interfaces for software. The focus is to maximize usability and user experience [81]. We created mock-ups of the UI design in Adobe XD and utilized the possibility of creating prototypes to test.

3.9.3 Mobile-first

For creating the voter screens we used a mobile-first approach (see 2.12.2).

3.9.4 Diagram and Model Documentation

We have used Visual paradigm (see 3.5.7) to create UML diagrams for this application. UML diagrams is a way to visually represent the architecture, design, and implementation of complex software systems [96, 112]. We created UML diagrams when modeling difficult situations, scenarios or architectural choices.

3.10 Hardware

The following section describes what hardware was used to host and develop the product.

3.10.1 Personal Equipment

Development was done on Linux/Ubuntu, Mac and Windows in Windows Subsystem for Linux. PWA was installed on Windows, Android, Linux, iOS and MacOS.

3.10.2 Server

The server used for running our services is a virtual machine provided by NTNU Ålesund AutoDeploy, with the operating system Ubuntu 20.04.1 running on it.

The server has the following hardware specifications:

- 6 cores of an Intel Xeon E5-268Zw CPU running at 3.00GHZ
- 16gb of RAM
- 40gb of storage on a Harddisk

Chapter 4

Result

In this chapter, we will describe the results that we were able to find and construct during the project. This chapter will describe in detail the results of the different subsystems that make up the application and the results of our project methodology.

4.1 General results

As part of this project, we have been able to develop, test, and deploy a production ready application. The application is available at https://anovote.app. The web application is developed using React.js (3.5.13.1) and can be downloaded as a PWA (2.3.2). All core functionality that is being described in 1.3, has been implemented. The finished result is a fullstack web application running in a Docker container network behind a reverse proxy. The documentation of our project development is hosted via https://docs.anovote.app.

We have conducted usability tests and large-scale user tests. The test results will be examined in section 4.13.2 and 4.13.3.

4.2 Use-Cases of the Application

Early in the project we defined the use-cases of the system. Figure 4.1 displays the different actors. A voter is a user only when an election organizer organizes an election. The voter can join an election. An election organizer cannot join as a voter for its own election, but can be a voter for any other election. The organizers have organizational rights to administer elections, add eligible voters, and organize it. For the rest of the chapter, we will refer to two perspectives; *election organizer* and *voter*.

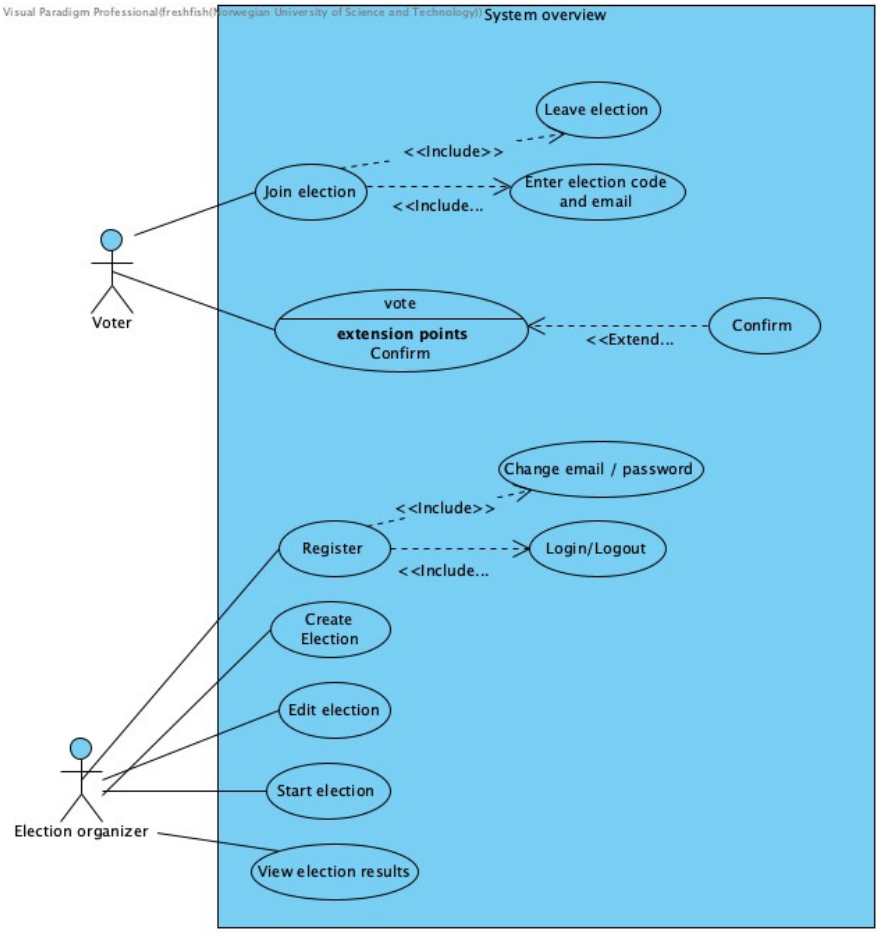


Figure 4.1: Use-case diagram of the system

4.3 Functionality of the system

This section focuses on the functionality of the system, from the perspective of the election organizer and the voter.

4.3.1 Election Organizer

The election organizer create and organize digital voting system (see *Online Voting* 2.1), where the election organizer can organize an election. The election organizer is responsible for selecting the *electorate* (see 2.1.2) and can create ballots (see 2.1.3) for an election.

4.3.1.1 Election management

Election management by an election organizer is the core of the systems functionality. The election organizer can CRUD an election as well as start, hold, and end an election.

Creating a Election When creating an election, the election organizer must provide a title and description. An election must have a unique title. This is implemented to avoid confusing the election organizer by having several elections with possible duplicate titles. The election organizer can add eligible voters to the election by two different approaches; manually, or by uploading a CSV or JSON file. Elections can be scheduled with open and close dates (see *Scheduling* 2.1.2) as any ordinary election. Ballots that eligible voters can vote on can be created and edited. There is no limitation to how many elections an election organizer can create.

For each ballot, a title and description can be added. Currently, the voting system only supports plurality system elections (see 2.1.1.1). The election organizer needs to add at least one candidate to the ballot. It is possible to edit or remove the candidate from the ballot. The order of the candidates can also be edited by simple drag and drop in the UI. All ballots will automatically have a default blank candidate, enabling voters to vote blank. See Figure 4.3.

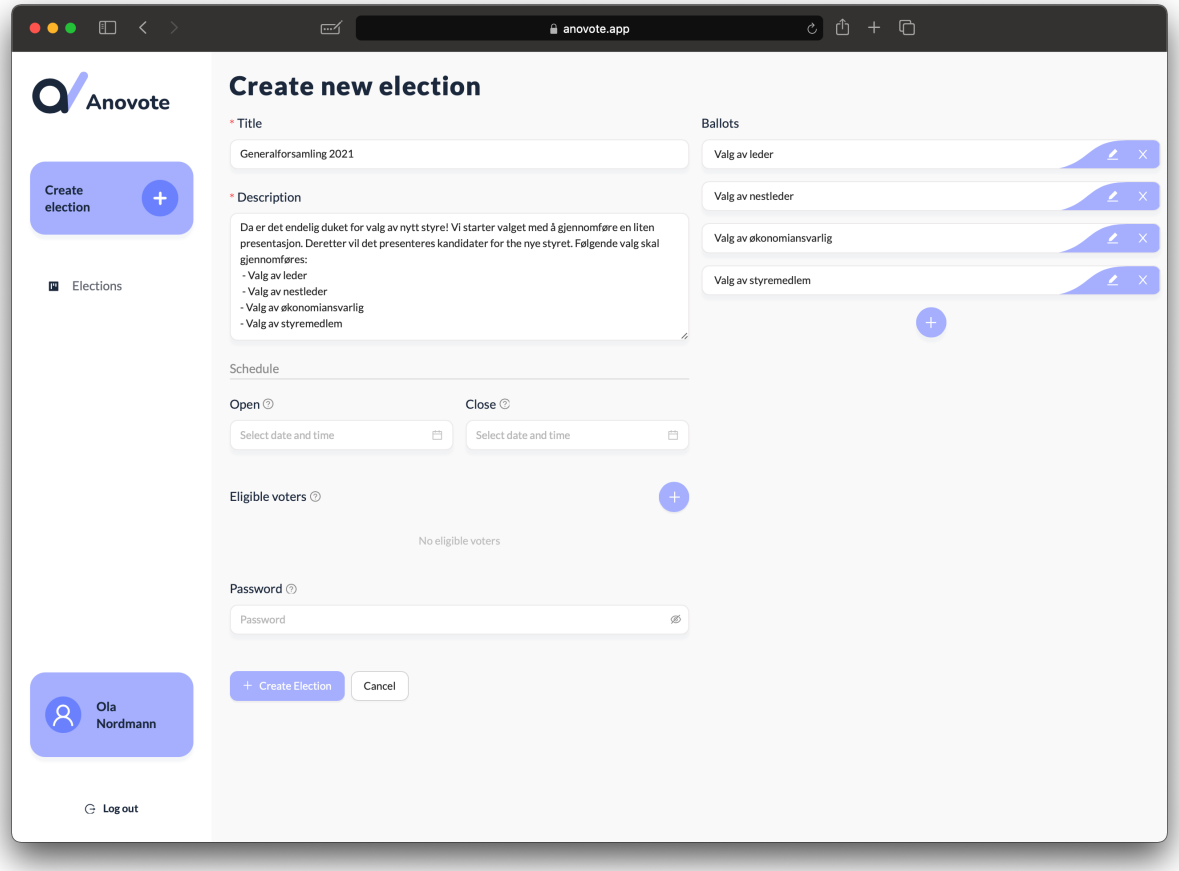


Figure 4.2: The "Create election" page with ballots

After the ballot has been created, the election organizer can choose to edit or remove the ballot from the election.

View all Elections The election organizer can view all their elections (see Figure 4.4). All elections will be categorized based on the election state (see 4.6.1 for more details on state).

Updating an Election From the election homepage, an election organizer can edit an already existing election. The edit dialog is the same as the create election, and should be familiar to the organizer. After editing the election, the updated state is saved to the database and the related views are updated. An election can be edited until the election is started.

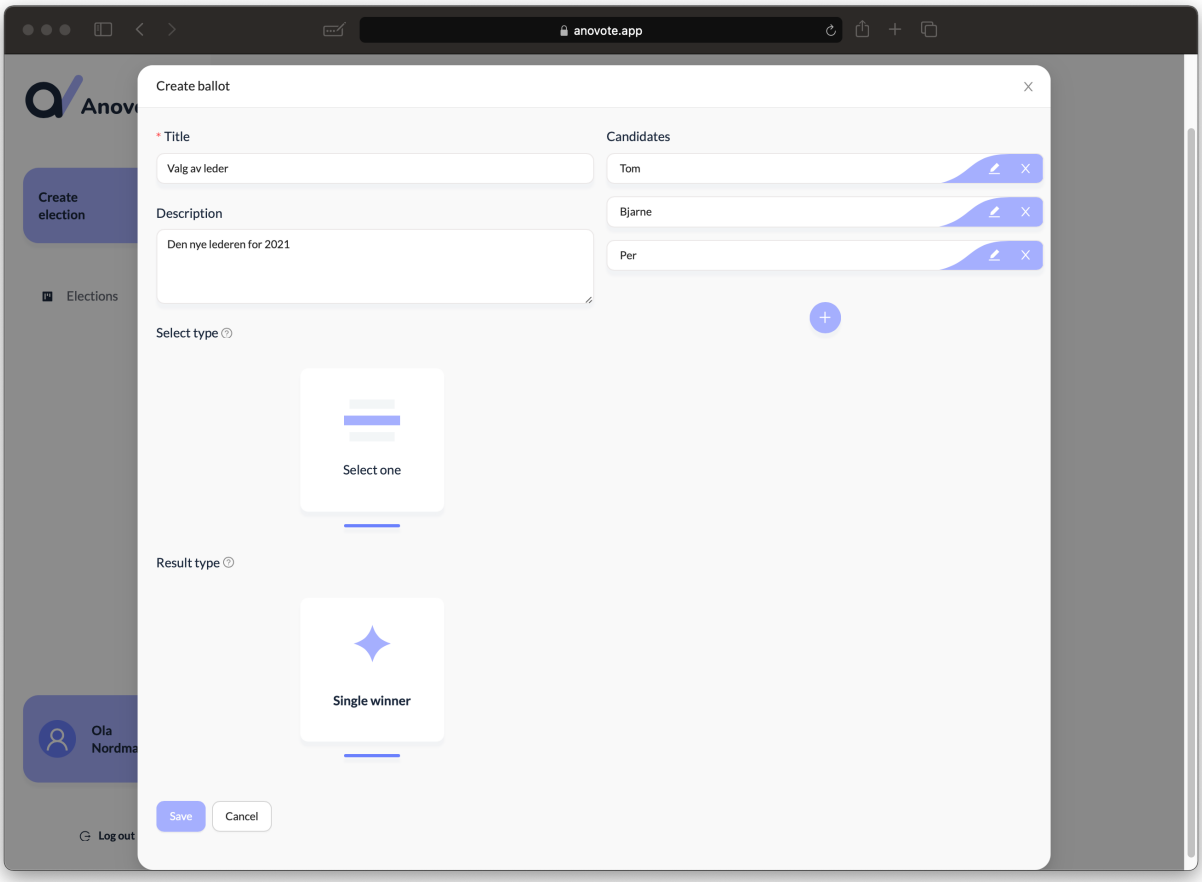


Figure 4.3: The "Create ballot" modal with candidates

Delete an Election To delete an election, the organizer must choose the relevant election and press the "delete election" button. The organizer will be prompted for conformation before delete. Deletion is not possible if the election is in the *Started* state (see 4.6.1).

Start an Election An election can be started in two possible scenarios; forced or automatic. An election can be configured with automatic open and close dates. These dates will automatically change the state of the election. An organizer can also force a state change, ie. trigger an election to start and end. This will overwrite the relevant dates and set them to the time of state change.

Hold an Election When an election is *Started*, the election organizer can manually push ballots to the voters. The push mechanism is described in more detail in 4.6.7. While an election is

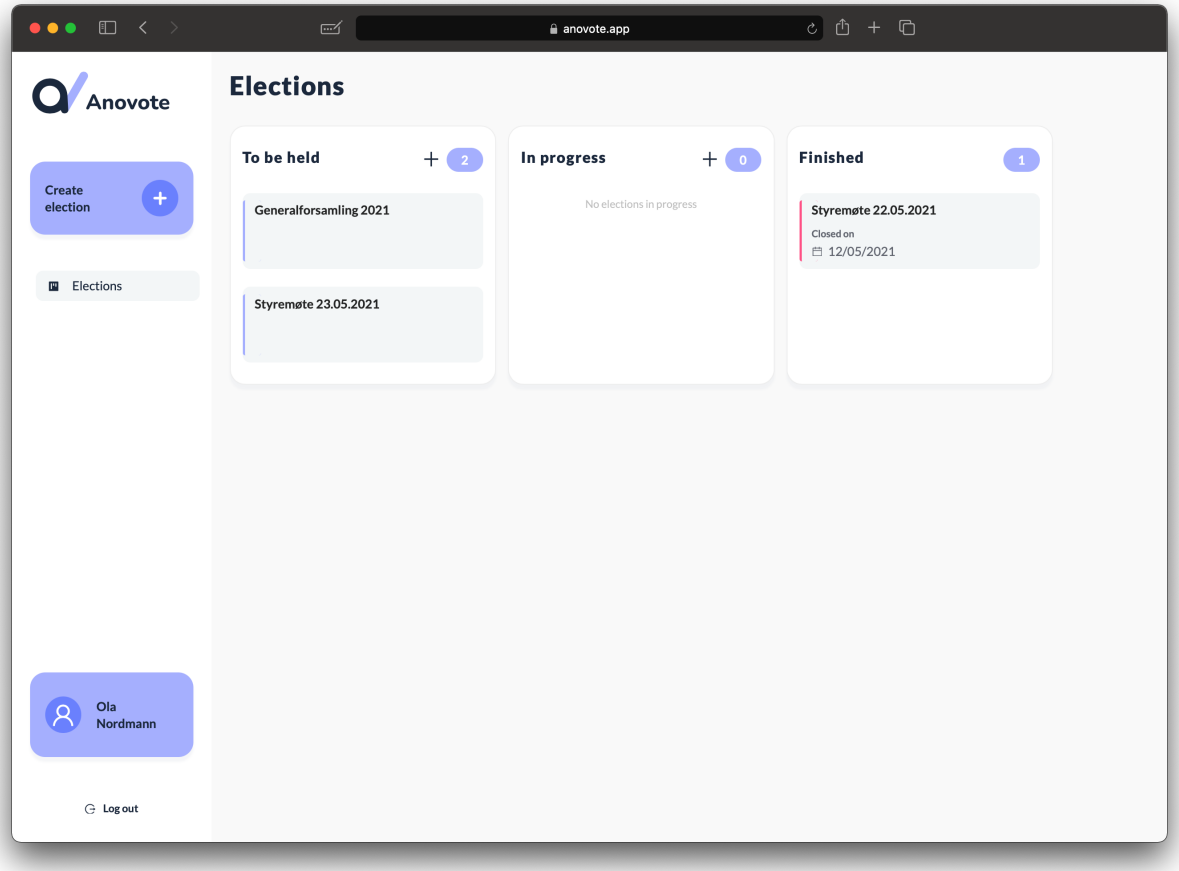


Figure 4.4: The elections view displays all the elections an organizer has created

running, the organizer can view how many voters that are connected, how many have voted on each ballot, and see the distribution of the votes. During this state, all communication between the server and the client is handled by Socket.io, through its event driven design (2.4.3).

Ending an Election When an election is ended, either automatically or by force, all connected voters are being logged out, no more votes are registered.

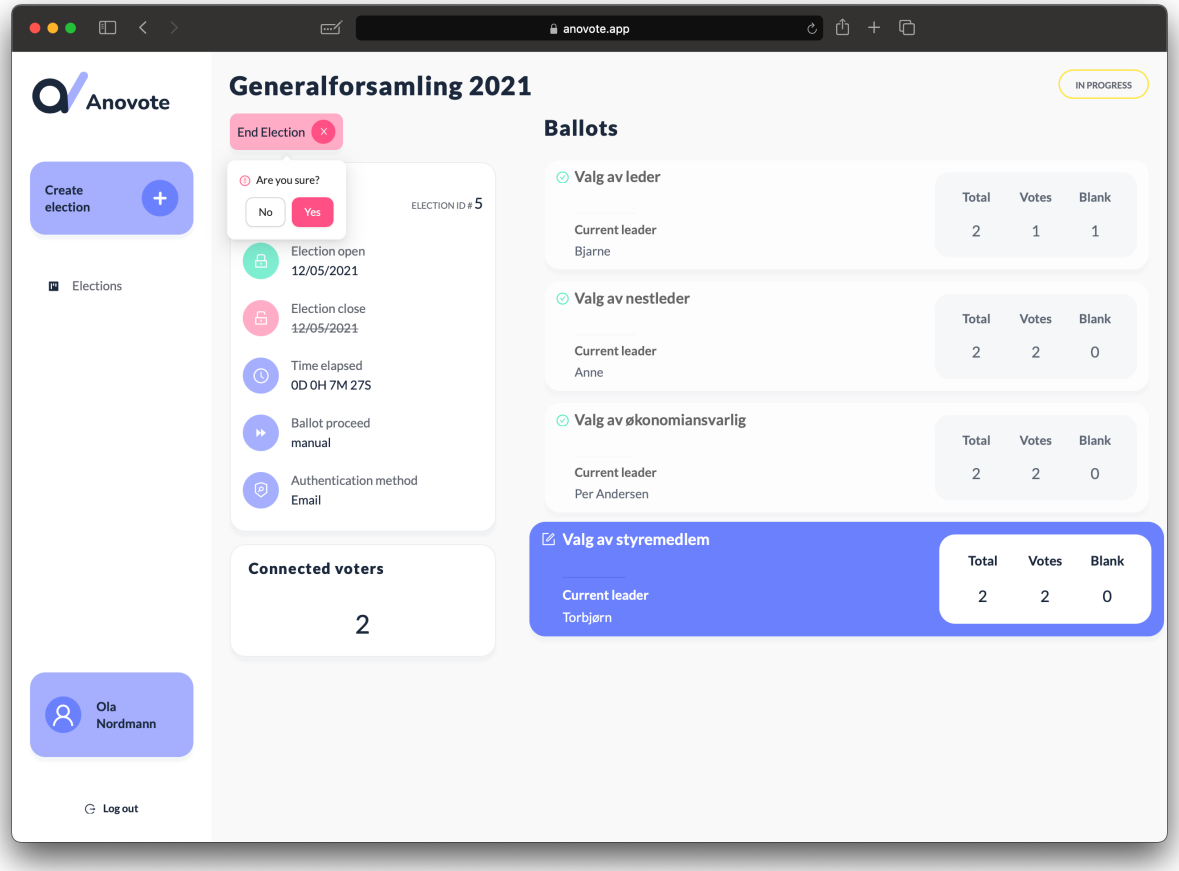


Figure 4.5: An election in progress with the end election confirmation dialog

4.3.2 Eligible Voter

4.3.2.1 Joining an election

The eligible voter joins an election by using the "join election" page. Joining is done by presenting an email address and an election code provided by the election organizer. The joining mechanism is described in details in 4.6.3

Voters are only able to join an election that is *Started*. If the election is *Not Started*, the voters are presented with a screen telling them to wait for the election to start. If the election is *Finished*, voters that try to join will be informed that the election is finished. If voters try to join an election that does not exist, either because it was never created or it has been deleted, they are informed that the election does not exist. The different views are depicted in figure 4.6.

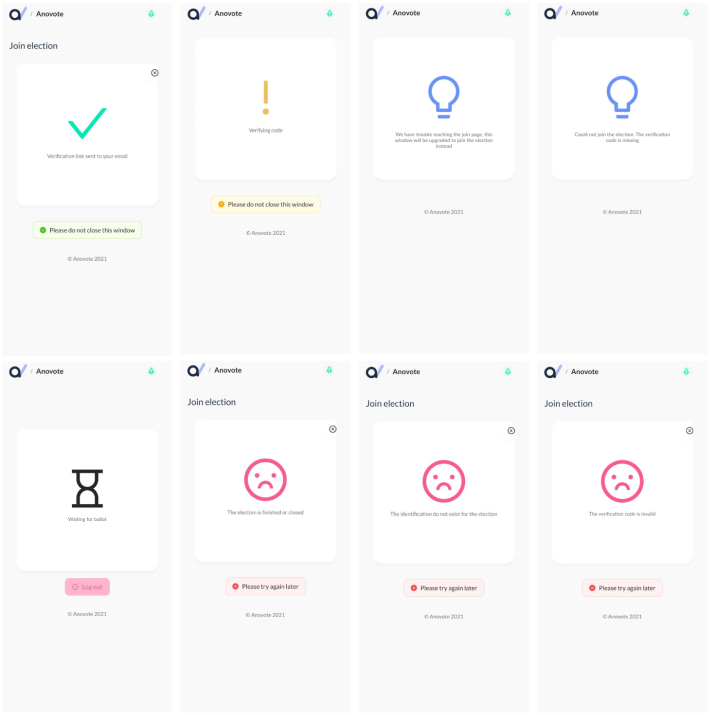


Figure 4.6: Information shown to a voter when joining an election. From the top left, (1) the voter is accepted to the election, waiting to be verified, (2) verification happening, (3) upgrading the current window, (4) could not join because of missing verification code, (5) waiting for a ballot, (6) the election is closed, (7) election does not exist, and (8) verification code is invalid

4.3.2.2 Voting in an Election

When a voter receives a pushed ballot from the election organizer, the ballot will get displayed on all connected voter's screens. The voter will be displayed the title of the ballot, and a list of all candidates on the ballot and a "vote blank" option. The voter then can choose a candidate from the list of candidates to vote on, or choose the vote blank option. After choosing how eligible voters want to vote, he/she can press the "submit vote" button to cast their vote. After voting the eligible voters will be placed back into the waiting state of the election. A full sequence diagram can be be seen here: B.2

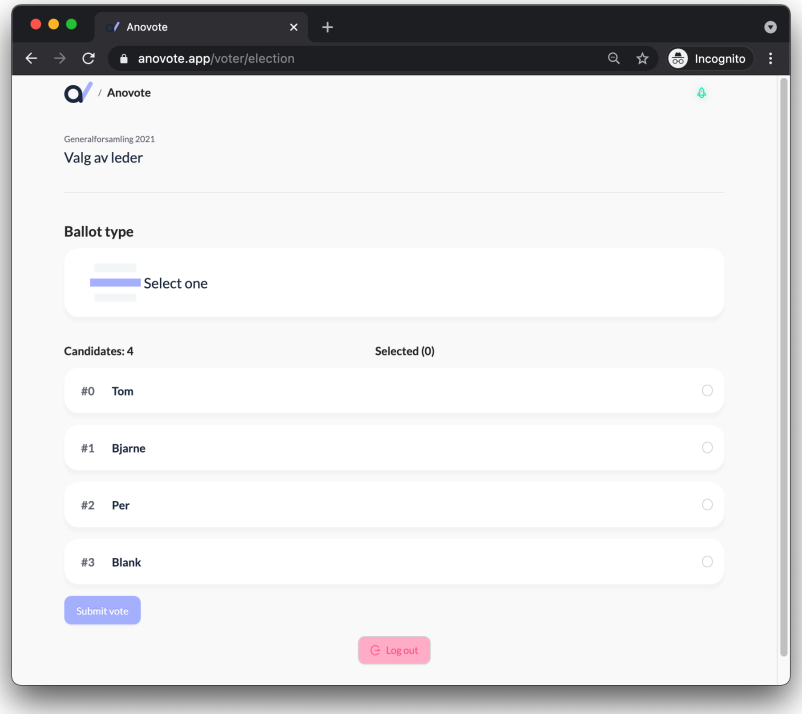


Figure 4.7: Voting screen where a voter has the possibility to vote on a specific candidate.

4.4 PWA

The application is able to be installed as a PWA on supporting device. It works as expected on most devices, except under Ubuntu (Linux). This is a known bug on Ubuntu, and is therefor outside our realm [12].

4.5 Client-Server Communication

In order for the client-server to interact with one another, we have used different forms of communication protocols. We will describe how this communication is resolved during the different stages of an election.

The following diagram describes how the network and communication infrastructure is organized.

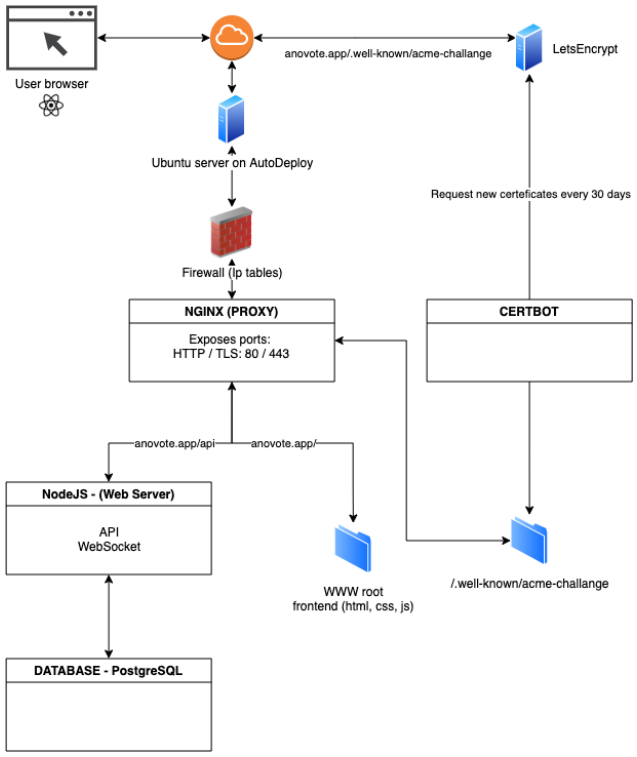


Figure 4.8: Network and communication infrastructure diagram

4.5.1 Basic Communication

To be able to use the system, a user first must download the frontend application. This is done by enabling the server to serve the application documents. These documents are static files that are served through nginx (3.5.2) and can be accessed by going to `https://anovote.app`. When the documents are loaded by the client browser, the user can start to use the application.

4.5.2 Request to the REST API

For most of the functionality, the user will have to fill out forms in the application and submit them to the server. When a user submits a form, the frontend application will make a HTTP

request to the server's REST API. These request is made by using *axios* library and requests are sent to `anovote.app/api/`. Any request to the server is then handled by *express*. The communication with the REST API is the backbone for all creation and deletion of entities in the system, e.g., registration, login, and creation of elections. Fetching of data is also handled by the REST API. The API is also guarded with rate-limiters to prevent miss-use of the API, and hinder brute force attacks (2.2.4) on authentication endpoints.

4.5.2.1 Express

To create the REST API 2.5.2 we have utilized express (3.5.13.2). Express will handle all incoming HTTP requests and map them to the appropriate service class. E.g., when a new election organizer would get registered on the frontend, an HTTP POST request would be sent to `anovote.app/api/election-organizer/`. Express will then look at the endpoint address and unpack the request and send it to the election organizer service class for validation and storing in the database (4.8.3). Based on the results from the service class, the express REST API responds with a HTTP response code and relevant data.

4.5.3 Real-time Communication

When an election is in progress, data will be sent back and forth between the backend and several different clients. Because we would like all voters to be updated immediately after a ballot has been pushed, clients have to be notified in real-time. For handling this, communication is happening in real-time through the use of socket.io web-socket server. When a websocket event arrives at the websocket server, the websocket server reads the event name and sends the websocket event to its corresponding event handler. The event handler would then unpack the websocket event and handle it. The implementation of event handling can be found in 4.5.4.

4.5.3.1 Socket.io Websocket Events

Instead of having to tell the server that some data is coming, the data is just sent and you hope that the server has a way to handle it. Socket.io implements the observer pattern and is event driven (2.4.3). Each websocket emittion has an event name that the server or client can subscribe to. When an event is fired, the server will read the event name and if it recognizes the event name it will know how to handle the incoming data. E.g., when a voter presses the "vote" button, a websocket event with the voters ballot will get sent to the server with the event name *push-ballot*. The server will then know how to handle the ballot by reading the event name. An event sent to the server can trigger the event handler to submit new events to the clients, where appropriate.

4.5.4 Event Handling

To accomplish the communication between client-server with websockets in a reliable way, we expanded the structure around the Socket.io (3.5.13.2) implementation. Socket.io provides methods to pass events with acknowledgments. Since events can be any data type, we wanted to control how the events were received and handled on the client side.

```
export function WebsocketEvent<T>({
                    dataHandler,
                    errorHandler
                }: IWebsocketEventHandler<T> = {}): EventExecutor<T> {
    return function executor(payload: IEventResponse<T> | undefined): void {
        if (dataHandler) {
            if (payload?.data) dataHandler(payload.data)
            // Handles data that is undefined, or has a payload that is not
            // wrapped in {data, error} object
            else if (!payload?.error) dataHandler(payload as T)
        }
        if (errorHandler && payload?.error) errorHandler(payload.error)
    }
}
```

Listing 4.1: The event handler implemented on frontend

Listing 4.1 shows the function for assigning an incoming event to separate data- and errorhandlers.

4.5.4.1 Responding with Event Message or Error

To directly respond to an event, known as acknowledgement, we can use the *EventResponse*.

```
export const join: EventHandlerAcknowledges<{ email: string; electionCode: string }>
     = async (event) => {
    // ...

    return event.acknowledgement(
        EventErrorMessage(new NotFoundError({ message:
        ServerErrorMessage.notFound(entity), code }))
    )
}
```

Listing 4.2: Showcase of acknowledgement on the backend. This event will be acknowledged with a *NotFoundError*

For Listing 4.1 the *WebSocketEvent<T>* on Listing 4.2 will now propagate the acknowledgement as an error to the errorhandler.

4.6 Developing a Voting System

4.6.1 Election Lifecycle

An election can have the following three states.

- Not Started
- Started
- Finished

These states are defined in the election entity class. An *Election* can also be deleted. This will remove the entity from the database, triggering a cascade delete, removing all related data.

In the two other states, *Not Started* and *Finished*, the communication between the client and server is happening through the REST API (2.5.2) over HTTPS.

To render appropriate information to the election organizer, each life cycle state has its own component that is rendered based on the election state. The election organizer can use the same URL, regardless of the election state, to get to the election.

4.6.1.1 Automatic Opening and Closing of Elections

To check if any election should be started or stopped regarding their open and close dates, Node CRON job is used. Every minute the CRON job checks if the current date and time is the same as the open and close dates for all elections. If it is, the CRON job will either open or close the election.

4.6.2 Anomyzation with Algorithms

As described in 2.1.4, implementing a proper anonymous, secure and tamperproof digital voting system is a time-consuming and difficult task. Due to time constrains, and by recommendations from our supervisor, we choose to simplify our implementation to reach our primary goals.

To ensure that only eligible voters should be able to vote, there is a need to verify the identity of all voters. We have implemented a two-factor authentication system for this purpose. The implementation will be disclosed in 4.6.3. The election organizer will provide a list of emails for voters that are allowed to participate. These voters can then join the election with their email (something they own/have) and an election ID (something they know).

When a voter requests to join an election with the email submitted by the organizer, a verification email is sent to the particular address. The email consists of a verification link, which includes an encrypted token as a query parameter. The link directs the voter to a verification

page that sends the token to the server for validation. On successful verification, the voter is automatically allowed in to the voting view on the page that was used to join the election. The system implementation is described in more details here 4.6.3.

4.6.2.1 Blockchain as Voting Mechanism

Blockchain technology was considered as a solution for providing an anonymous and decentralized solution to hold an election. Research and testing the ability to use etherum blockchain was conducted early in the project.

We deployed a development blockchain using TruffleSuite (3.5.17.4) on a local machine for testing smart contracts through a web interface. For testing, we used open source election/voting smart contracts found on GitHub. Submitting a smart contract - which would represent an election to the network required the submitter to spend some of its etherum as a fee on the network. A fee was also required to be paid by any node – i.e., a voter that would want to execute a computation (vote) on these election smart contracts, as a network fee.

4.6.3 Verifying Participants when Joining an Election

To enable voters to participate in an election, voters need to join and then verify their identity. For this, we have created an algorithm that we will try to explain in the following section. A full system sequence diagram can be seen in Appendix B.1.

4.6.3.1 Joining

When a voter enters the *Join election* page, a web socket connection is established with the back-end server. This web socket connection is used throughout the joining process as a real-time event emitter and listener. The socket is initialized with three event listeners on the server, *join, verification* and *token*, and a *voterVerified* event on the client side. The *join* event is emitted to the server when a voter submits the join form (see figure 4.10).

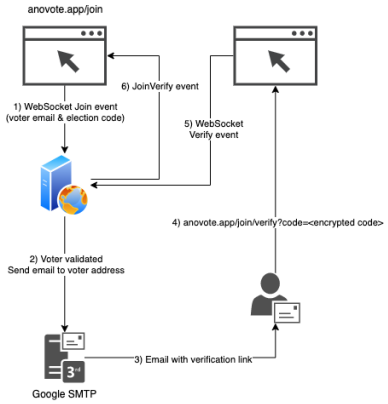


Figure 4.9: Simple illustration of join and verify

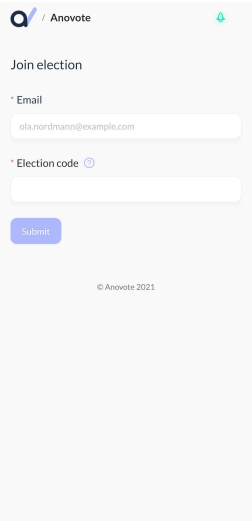


Figure 4.10: The join-election page. A voter needs to provide his/her email and an election code to join

In order for the server to notify the client that the join event is received, the emitted join event has an acknowledgment callback that the server can reply to. The server can reply with either success or error (see 4.5.4).

A state change occurs and the voter is immediately presented with a loading state and a recommendation to not close the browser window. User feedback is implemented in all stages to enhance user experience and comply with the design principles discussed in 2.12.1.

A series of validation routines are performed on the submitted voter credentials and the election which the voter is trying to join. The validation steps are:

- Verify that valid data is passed
- Check if election exists
- Check if the election is not finished or closed
- Check if the email exists in the list of eligible voters for the given election
- Check if the email has already been verified for the election

If any of the validation steps fails, an error is returned to the client through the acknowledgment callback (see 4.5.4). The error response is handled on the frontend with user feedback (see 4.17) and aborts the joining sequence.

4.6.3.2 Generating Verification Code

When all verification steps have passed, a verification code is generated. This verification code is an encrypted string containing the first part of the email, the voter id, the election id, and the socket id for the current websocket connection. To ensure that the code is not able to be generated outside our system, it makes use of a stored secret for performing the encryption. This secret is also used to decrypt the message later. This code is added to a verification URL (see Listing 4.3).

```
https://anovote.app/join/verify?code=U2FsdGVkX19RTZ%2Foz2xhEnavmYm
%2FBIIdO3PPAF2l5iJai1KS4lU1DsWv7nULGq6m7DgJS47BNEyP5ViqnvKssA%3D%3D
```

Listing 4.3: Example verification URL

A *MailService* is used to generate and send an email to the voters email address. We use Google's SMTP server for email transport. This email body contains a short description and the verification URL. The voter needs to click the verification URL to show that he/she is the owner of that particular email address.

When all steps have successfully completed, an *OK* acknowledgement is emitted back to the client. This will display a message that a verification link is sent to the particular email, and an instruction to not close the browser window, see illustration 4.6. The server side also subscribes to a new single event, *voterVerifiedReceived*. This event is used later in the verification step to perform notifications.

The browser window is instructed to stay open as part of the verification process, but it is not required, as fallback mechanisms are implemented to tackle this. This will be explained in more detail in the following section.

4.6.3.3 Verifying

When a voter opens the verification link, they are presented with a verification page that initializes a connection to the websocket server. The client side is subscribed to a *joinVerified* event, and the server is subscribed to the verify event. A verification timer is started on the frontend, which is part of the fallback mechanism in case of a closed join page. The verification code is gathered from the "code" query params. In case of a missing verification code, we abort the verification and a missing code message is displayed to the user, see 4.6. The verify event is emitted to the server with the verification code and an acknowledgment callback for success/error.

Once the server has received the verify event with the verification code, it decodes the verification code and gathers the data inside it. The verification process uses the same verification steps as described in 4.6.3.1, but also a code verification step. The data gathered from the verification

code is used in the verification process. In case of a missing code or failed decoding, we throw a *BadRequestError* with a message and error code. Any error is emitted back to the client via the acknowledgement callback.

Once the verification steps are successful, a JWT token is generated on the server, and the voter is marked as verified. The socket id for the join page in the verification code is then used to emit an event to the join page socket that the verification was successful. It includes a data payload with the verification socket's id and the JWT token. The server side of the verification socket will also subscribe to an *upgradeVerificationToJoin* event, which is part of the fallback mechanism. When the join page receives the *joinVerified* event, a *voterVerifiedReceived* event is emitted from the join page back to the server, with a data payload of the verification page's socket id in the received event payload. The frontend will then store the JWT token, set the application state to be logged in as a voter, and redirect to the voting view page.

The *voterVerifiedReceived* event received by the server uses the passed verification page's socket id to emit a *joinVerified* event to the verification socket frontend to notify the page that the joining was successful. The server stores the election ID and voter ID on the join socket for later reference. The join-socket is also joining the socket room for the given election and is subscribed to the vote event. The verification page disables the upgrade timer when the *joinVerified* event is received to prevent an upgrade.

In case of a voter closes the join page or there are issues with the acknowledmgents between the two sockets (join/verify). Then the fallback mechanism will be applied. The fallback mechanism uses the timer started on the verification page as an expected time to respond. If the *voterVerifiedReceived* event is not received on the verification page before the timer has expired, it will emit an *upgradeVerifyToJoin* event that signals the server that the verification page is to receive the JWT token. In the event of upgrading, the user will be notified with different state messages indicating this.

4.6.3.4 Re-join

The browser used to join has a stored JWT token. This token ensures that the voter can rejoin the election when he/she refreshes or closes the browser and opens the join page. Before they are able to rejoin, we fetch the election by the id stored in the JWT token. If the election is not *Closed/Finished*, they are able to return to the election. When they are redirected to the election page, a *joinWithToken* event is emitted to the server. This will perform validation steps and subscribe the voter to the election room and vote event.

4.6.4 Socket Room Service and Initilizing WebSocket Channels

As there can be multiple ongoing elections at any time, we need a way of controlling what voters should get what ballots. Socket.io (3.5.13.2) provides a way of creating "rooms" or channels that can filter events to clients assigned to the room. To keep track of the different users in each election, each election would get an assigned room. To keep track of what rooms are open, we have implemented a *SocketRoomService*-class. The service is responsible for managing the state of the socket rooms and adding voters to the correct room when they join. The *SocketRoomService* is a singleton (see 2.4.1) and therefore only one instance can exist to ensure the integrity of the elections.

4.6.5 Creating the Election Room

Upon the creation of an election, a socket room entity is automatically initiated together with an election (see Listing 4.4).

```
private async createElection(electionDTO: IElection): Promise<Election | undefined> {
    ...

    if (!election.socketRoom) {
        election.socketRoom = new SocketRoomEntity()
    }

    ...
    return await this.manager.save(election)
}
```

Listing 4.4: Code snippet to initialize an socket room entity in ElectionService.ts

The *SocketRoomEntity* stores information about the room state *(Open/Close)* and belongs to an election.

As Listing 4.5 shows, the *SocketRoom* will associate the election room with an *ElectionRoom* instance for a specific election. This *ElectionRoom* stores information about the total eligible voters count as well as the vote information for each ballot.

```
// routes/election/index.ts

router.post('/', async (request, response, next) => {
    try {
        const socketRoomService = SocketRoomService.getInstance()

        // ... election created

        if (election) {
            socketRoomService.createRoom(election)
        }
        response.status(StatusCodes.CREATED).json(election)
    } catch (error) {
        // error handling
    }
})

// SocketRoomService.ts

createRoom(election: Election) {
        const room = this._electionRooms.get(election.id)
        if (!room) {
            const ballotVoteInformation: BallotVoteInformation = new Map()
            for (const ballot of election.ballots) {
                ballotVoteInformation.set(ballot.id, {
                    stats: new BallotVoteStats(ballot),
                    voters: new Set()
                })
            }
            this._electionRooms.set(
                election.id,
                new ElectionRoom({
                    totalEligibleVoters: election.eligibleVoters.length,
                    ballotVoteInformation: ballotVoteInformation
                })
            )
        }
    }
```

Listing 4.5: Creation of socket room for election

4.6.6 Initializing socket room

When the *SocketRoom* entity is initialized, the server will now generate a websocket room upon first joining socket. The joining socket will be associated with an election as described in 4.6.3.3. The *SocketRoomService* are then able to affiliate the socket with the correct *electionRoom*. This room/channel is where the election events, i.e., *pushBallot* (4.6.7), *submitVote* (4.6.8), and others are broadcasted.

4.6.7 Pushing a Ballot

A ballot can have three different states, *In Queue, In Progress* and *Archived*. The first state, *In Queue*, indicates that the ballot has not yet been pushed.

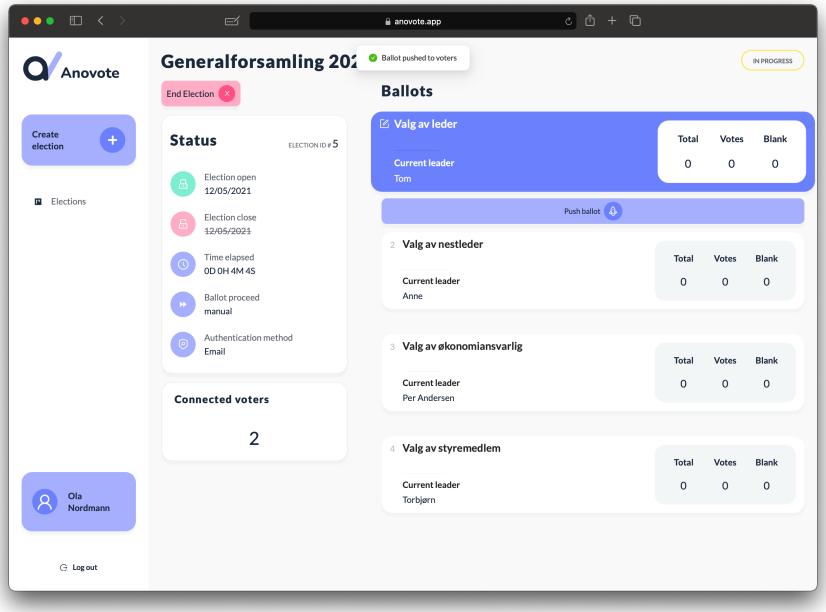


Figure 4.11: Panel for an election in progress. From here, the election organizer can push the selected ballot.

Once an organizer pushes a ballot to its eligible voters, an event is emitted from the administrator panel to the server. As described in 4.6.4, the socket room service is responsible for handling all elections that exist on the server. In order for an ballot to exist, an organizer must have an election stored on this service. If no room exists for the election, pushing a ballot is impossible.

A fundamental part of a voting system (2.1.4) is to ensure voters can only cast one vote for a given ballot. To ensure this, a ballot can only be pushed once to the voter. By utilizing the socket room service, we are able to get information about who has voted on the ballot. When a ballot is pushed for the first time, the ballot is indeed pushed to all voters. However, if a voter did not get the pushed ballot, the organizer could push the ballot again. The conditional on Listing 4.6 would ensure that other already voted voters would not get a ballot multiple times. This solution helps to ensure the integrity of the votes as it is more difficult to cast multiples votes on a ballot.

```
// Get voters on the ballot
const ballotVoters = room.getBallotVoters(ballotId)
if (ballotVoters.size == 0) {
    // Send to all as no one has voted on the ballot
    roomSocket.emit(Events.server.ballot.push, ballot)
} else {
    emitBallotToNotVotedSockets(event.server, roomId, ballotVoters, ballot)
}
```

Listing 4.6: Push ballot logic from pushBallot.ts

Only one ballot can be displayed to the voter at any time, i.e., if an organizer pushes multiple ballots before the voter has cast their vote, only the latest pushed ballot will get displayed. Since a voter only is allowed to cast one vote per ballot, the organizers do not have to worry about pushing the same ballot multiple times. If a voter has voted, the ballot will not be sent to that voter again, and if the voter has not voted, the ballot is sent, enabling the voter to cast a vote.

Once the organizer has pushed the ballot, the ballot state is changed to *In Progress*, and the voter will get the ballot displayed. From here, the voter can choose between the candidates or vote blank.

If all voters have voted on a ballot, the state is automatically changed to *Archived*, indicating to the organizer that there is no need to push the ballot again. If an organizer tries to push a ballot that is archived, the organizer is notified immediately, and the ballots are not sent to any voters.

4.6.8 Vote Handling

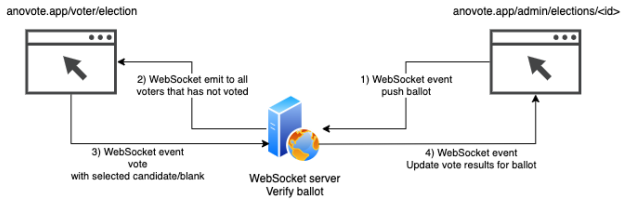


Figure 4.12: Simple voting sequence

When a vote gets submitted by an eligible voter, the WebSocket on their device will send the submitted vote to the WebSocket server on the backend. At the WebSocket server, a *submit vote* event is registered, which includes a *submitVote* method which is responsible for handling the backend side of the vote submission.

```
export const eventRegistration = ({ client, server}:
{ client: VoterSocket; server: Server}) => {
    client.on(Events.client.vote.submit, (data,  acknowledgement) =>
        submitVote({ client, server, data, acknowledgement })
    )
}
```

Listing 4.7: Event registration for submission of votes

From here, the vote will go through a series of validation steps:

1. The ballot being voted on exists in the database
2. The election being voted in exists in the database
3. If the election being voted in is ongoing
4. The candidate voted on exists and is connected to the ballot being voted on
5. That their is not an already existing vote for the ballot being voted on by the voter

If any of these validation steps fail, an error will be thrown and therefore the sequence of submitting a vote will be stopped. The thrown error will then be catched by the *submitVote* method, and the error will be responded to the client in the form of a WebSocket acknowledgment.

```
} catch (error) {
    if (error instanceof BaseError) {
        event.acknowledgement(EventErrorMessage(error))
    }

    // {...}
}
```

Listing 4.8: Error handling for vote submission

If all of these validation steps succeed, the vote submission sequence will continue. The submitted vote will then be sent to the *VoteService*-class which is responsible for working with the database regarding votes. Here, the vote data will be converted into Vote entity instance used by TypeORM.

TypeORM will then try to insert the *Vote* entity to the database on the backend, here there are also some validations being made:

- The voter id is a positive number.
- Their is a relation to the ballot being voted on.
- Their is a relation to the candidate being voted on.

If any of these validations fail, TypeORM will throw an error and it will be handled by the *submitVote* method described in listing 4.8.

4.6.9 Validation of election organizer

The validation of election organizers is done with the use of tokens, whenever an organizer is logged into the system, the organizer will carry an access token. When an election organizer wants to enter an election as an election organizer, the *tokenJoin*-method will send the token to the *verifyToken*-method for verification. If the token is invalid, I.E. the election organizer is not the owner of the election, the verifyToken method will throw an error. If the token is valid, the verifyToken method will return the decoded token to be used further by the tokenJoin method.

```
verifyToken(authorizationSchema: string | undefined): DecodedTokenValue {
    const token = this.getBearerToken(authorizationSchema)

    try {
        const decoded = verify(token, config.secret!) as DecodedTokenValue

        if (!decoded) {
            throw new UnauthorizedError()
        }

        return decoded
    } catch (error) {
        if (error instanceof JsonWebTokenError) {
            throw new UnauthorizedError()
        }
        throw error
    }
}
```

Listing 4.9: Simplified version of verifyToken method

The tokenJoin method will then tell the organizerJoin method to register all election organizer WebSocket events connected to the election organizer. The election organizer is now fully verified to control their owned elections.

4.7 Frontend architecture and technologies

The overall frontend architecture consists of three parts. React for UI, Ant Design for a range of base UI components, and Axios and Socket.io for backend communication.

4.7.1 React

The application structure is a component tree, with an App component at its stem. The app component encapsulates the core components for the entire application. It includes an application state component which holds the global application state, a router component that is responsible for switching the different views based on the URL path, and a websocket manager that exposes a socket.io provider to components that need a socket.io connection.

From the router component and down, the tree is switched out depending on the URL route. The router component is divided into public and private routes. Public routes are routes that do not require any form of authorization, e.g., home page, login, register. Protected routes require authorization and are also controlled by an authorization level. An authenticated voter can not access a protected route for an organiser, and vice versa. This is a frontend protection to prevent users from accessing routes which would not give them any data or functionality.

4.7.2 Ant Design as a React helper

Ant Design is used as a way to quickly create a robust, user-friendly, and universally uniform react UI. The biggest advantage of using Ant Design is how all components are uniformly designed and all components communicate with ease. In the register election organizer form, you

only need to define what rules each form input field should have, there is no need to implement the validation functionality since Ant Design provides this.

4.8 Backend architecture

The overall architecture of the backend consists of four parts, NodeJS as the web server, PostgreSQL as the database, TypeORM to communicate with the database, express to provide the HTTP endpoints, and Socket.io to enable web-socket communication. The database and web server were encapsulated in their own docker containers.

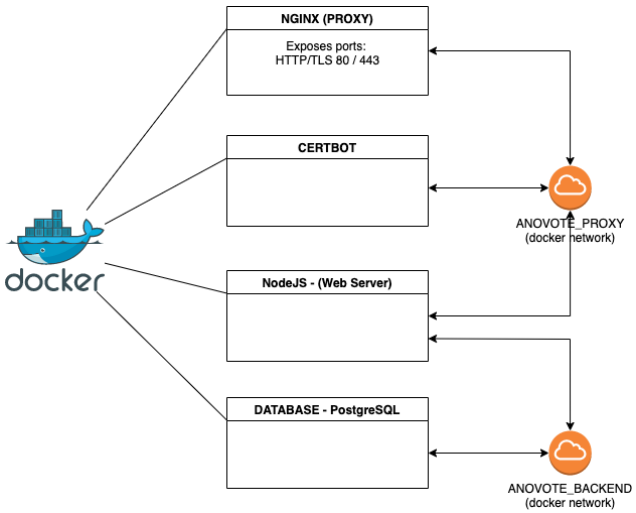


Figure 4.13: Container architecture

4.8.1 Node server

For running the backend server we used Node since we needed a runtime environment that builds scalable network applications. The node server sows together the express/socket.io communication layer with the TypeORM and PostgreSQL data layer. Node also lets us write the server in TypeScript, which enables us to only need one programming language for the whole voting system.

4.8.2 Database

Our system consists of different entities that are stored in a database. The Entity Relation (ER) diagram in figure 4.14 describes the relationships between the different entities. An election can have an one-to-many relationship with ballots, a ballot can have an one-to-many relationship with candidates. Each candidate can have many votes, but one vote can only have one candidate. An election organizer can have many elections and an election can have many eligible voters. Each voter can also participate in many elections. The relationship between an election and eligible voter is therefore many-to-many. Each election will also have their own socket room with a one-to-one relationship.

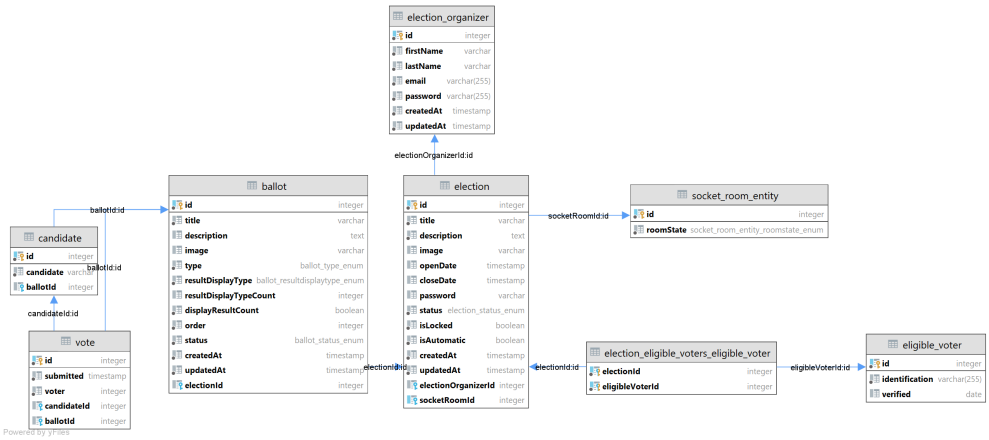


Figure 4.14: Entity relation diagram

4.8.3 TypeORM, Class-Transformer and Class-Validator

By utilizing TypeORM, we have been able to create entity classes that map to tables in the database. TypeORM uses annotations for specifying database column types. An example of an entity can be found in Listing 4.10.

For the application to work, it frequently Create Read Update Delete (CRUD) entities in the database. TypeORM provided an interface with intuitive CRUD commands. An example crud command can be seen in Listing 4.11.

```
/**
 * Represents the organizer of an election. An election organizer can organize many elec
 * An election organizer has a name, email, and password.
 */
@Entity()
export class ElectionOrganizer {
    @PrimaryGeneratedColumn()
    id!: number

    @Column({ type: 'varchar' })
    firstName!: string

    @Column({ type: 'varchar' })
    lastName!: string

    @Column({ type: 'varchar', length: 255 })
    email!: string

    @Column({ type: 'varchar', length: 255 })
    password!: string

    @CreateDateColumn()
    createdAt!: Date

    @UpdateDateColumn()
    updatedAt!: Date

    @OneToMany(() => Election, (election) => election.electionOrganizer)
    elections!: Election[]
}
```

Listing 4.10: Example definition of typeORM entity. The code does not include all fields and annotations

```
// getAllElections() in electionService.ts

// ...
return await this.manager.find({
    where: {
        electionOrganizer: this.owner
    }
})
// ...
```

Listing 4.11: Querying information with typeORM

Validations on entities To validate the data before storing it to the database, we have used the class-validator library. The validator works by annotation the entity with the right restrictions. A data object can then be run through a validator that will throw an error if any of the fields does not comply with their restrictions. For instance, validating the *firstName* for the election organizer can be done easily with class-validator using an annotation:

```
// ...

@Column({ type: 'varchar' })
@IsString()
firstName!: string

// ...
```

Adding more complicated validations were required in some cases. For instance, when creating a new election organizer, we had to make sure the election organizer was unique. To do this, we created our own validation annotation. The implementation and use can be seen in Listing 4.12 and Listing 4.13 respectively.

```
@ValidatorConstraint({ async: true })
export class IsElectionOrganizerUniqueConstraint implements ValidatorConstraintInterface
    async validate(email: string, args: ValidationArguments) {
        const [relatedPropertyName] = args.constraints
        const ownerId: number = (args.object as any)[relatedPropertyName]

        const electionOrganizer = await getRepository(ElectionOrganizer).findOne({ email

        // if an election organizer is found, and the owner is not this organizer, retur
        if (electionOrganizer && electionOrganizer.id != ownerId) return false
        return true
    }

    defaultMessage() {
        return ValidationErrorMessage.alreadyExists('Email')
    }
}
```

Listing 4.12: Definition of validation constraint for a unique election organizer

```
// using the constraint in ElectionOrganizerEntity.ts

@Column({ type: 'varchar', length: 255 })
@IsEmail()
@IsElectionOrganizerUnique('id')
email!: string
```

Listing 4.13: A field with both custom and predefined restriction annotations

As seen in Listing 4.12, the annotation can be used anywhere were an email is supplied. This type of validation can now be performed at every validation of an entity (see Listing 4.13). To

trigger the validation of an entity, the following code would run:

```
await validateEntity(organizer)
```

Class-transformer Some of the data stored is either sensitive (e.g., passwords) or not relevant to the user (e.g., creation date), and therefore should not be sent from the server. To strip fields from entities that we do not want to expose, we have used class-transformer (3.5.13.3). By annotating fields with the *@Exclude* and running it through the *classToClass* transformer, we can strip away all unwanted fields.

```
async getById(id: number): Promise<ElectionOrganizer | undefined> {
        return classToClass(await this.getElectionOrganizerById(id))
    }
```

Listing 4.14: Example of how we have stripped data that does not need to be exposed by API

For instance, the election organizer would have *@Exclude*-decorators for *createdAt* and *updatedAt*:

```
@Entity()
export class ElectionOrganizer {
    // ...

    @Exclude()
    @CreateDateColumn()
    createdAt!: Date

    @Exclude()
    @UpdateDateColumn()
    updatedAt!: Date
}
```

Listing 4.15: Class-transformer decorators used in election organizer

4.9 Using third-party library

Throughout the project we chose to use third-party libraries where we could. This has enabled us to create a more functional, more powerful, prettier and more secure application. We only had minor issues with some of the libraries. This was mostly fixed by tweeking the configurations of that particular library. E.g. both typeorm, eslint and prettier all have default settings that they will use if no override is found. For typeorm we had to provide details about the database in order for the connection to be established. Some of the default settings did not work and had to be changed to fit the project's needs.

4.10 Localizing the application

To localize the application, we used react-i18next (3.5.13.1). Every text string used in the web application has been prepared for localization. The default language for the application is English. A localized string looks like this:

```
export default function Home(): React.ReactElement {
    const [t] = useTranslation(['site'])
    // ...

    return (
        // ...
        <Title level={1}>{t('site:How Anovote works')}</Title>
        // ...
    )
}
```

Listing 4.16: A localized string in home/index.ts

The "site"-namespace refers to a locale file where all translation used for "site" is stored. This is a JSON (3.6.6) file with key-value pairs. Translating these files would allow localization to other languages.

4.10.1 Providing meaningful error messages

To communicate meaningful and localized error messages to the user, we created an error-code resolver:

```
/**
 * Error code resolver is a helper class resolving error codes to
        human readable error messages
 * translated current language using i18next translation as resolver.
 */
export class ErrorCodeResolver {
    private _translator: TFunction<string[]>

    private _codeMappings: CodeMapping = [
        { code: ErrorCode.alreadyVerified,
                message: (t) => t('error:Already verified') },
        // code mappings...
        { code: ErrorCode.alreadyVotedOnBallot,
                message: (t) => t('error:Already voted on ballot') },
        { code: ErrorCode.ballotArchived,
                message: (t) => t('error:Ballot is archived') },
    ]

    constructor(t: TFunction<string[]>) {
        this._translator = t
    }

    /**
     * Returns the error message for the provided code, if the code cant be
     * resolved to a message, a default unexpected error message is returned.
     * @param code error code to get error message for
     * @returns return error message for code, or default message
     */
    resolve(code: string): string {
        const codeMessage = this._codeMappings.find((e) => e.code === code)
        return codeMessage ? codeMessage.message(this._translator)
                        : this._translator('error.Unexpected')
    }
}
```

Listing 4.17: Full *ErrorCodeResolver*-class implementation

The *ErrorCodeResolver* in Listing 4.17 use the error code from a response to get the correct localized string. For instance, the error code *ALREADY VERIFIED* would be mapped to

```
t('error:Already verified')
```

which react-i18next would resolve to a localized string "You are already verified with a device" (see key-value pair in public/locales/en/error.json).

4.11 Continuous Integration and Delivery

We have incorporated Continuous Integration/Continuous delivery into out project by utilizing GitHub Actions (3.7.7). The frontend and backend repositories are configured with their own workflows which are run on pull requests, merges to the main branch, and on new tag releases. Both repositories had quality assurance and building workflows, our backend repository also contains a workflow for deployment. PRs would not get approved unless all GHA checks were completed and successful.

4.11.0.1 Workflows

Spellchecker workflow ran a spellchecker on the source code to find spelling mistakes in our code. The spellchecker understands variable and method names written with, e.g., camelCase or PascalCase. This workflow ran on both repositories.

Backend testing and building workflow was performing linting, building, compilation error checks, and running unit tests, as well as integration tests. The workflow also initializes a Postgres container to run implementation tests with a live database to ensure data integrity.

Frontend testing and building workflow was performing linting, building, compilation error checks, and running unit tests and UI tests (3.7.1).

Change log generation workflow was run when a new release tag was pushed to GitHub (in the event of a new release). This workflow generates a change log file in markdown (3.6.7) based on the latest pull requests since the last release (see 4.15.4).

Production Deployment workflow was run when a release with a new version is merged into the *main* branch. It performs an SSH login to the production server and executes our production start script described in (4.16.2.2) to automatically deploy new versions of the application.

4.11.1 CI/CD process overview

The diagram below illustrates the steps of the continuous integration cycle for our development branch. Pull requests for the *develop* branch will trigger this routine. The team receives notifications via Discord (3.5.8) during the process, and members must participate in the final step by performing a code review to approve the changes.

The diagram below illustrates the steps of the delivery routine to production. The developer uses the Git flow CLI to start a new release and publish this on GitHub. The release branch can then receive hot fixes (via a pull request which runs the CI steps shown in diagram 4.15) before the release is completed and merged into the main branch.

4.12 Environment files

As described in 3.8.0.1, we have used environment files to configure our different environment setups.

Environment files are not tracked by git as a security measure on our backend and are configured for the individual user/system. An example file is tracked by git to provide the necessary variables that are required. Our backend makes use of two .env files (.env.test, .env).

Environment files on the frontend are tracked by git as these contain no security-related con-

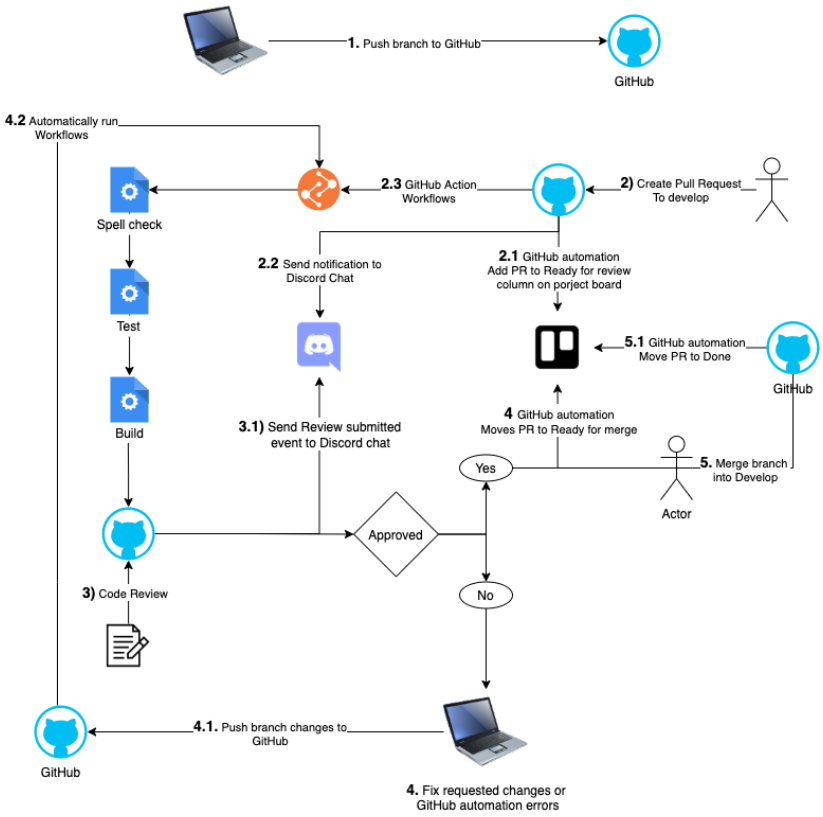


Figure 4.15: Continuous integration routine for development

figurations and are embedded into the output of a build to provide the data on the website. The frontend has three env files (.env.test, .env.development, .env.production). In local development, a developer can override these env files by creating a non-tracked version with a .evn.local.<test,development,production> file.

An env file is injected into the docker containers or red from file for the desired environment (testing, development, or production) which are used by the application to configure itself. For local development without docker, the file would be read from the source directory. When using docker the file is injected into the container configured in the docker-compose file.

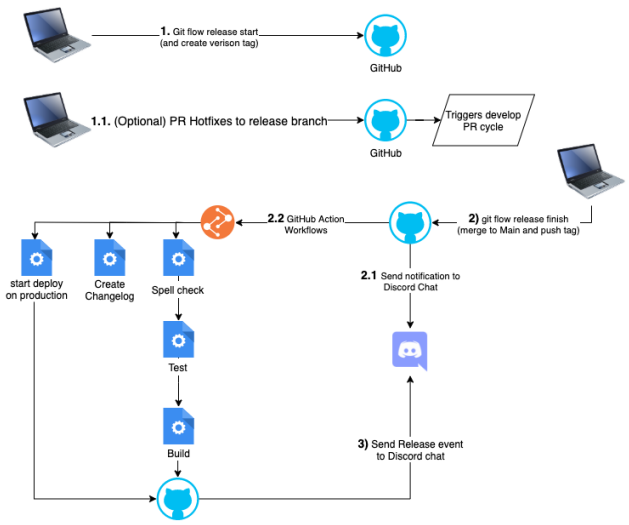


Figure 4.16: Continuous delivery routine for new releases

4.13 Tests and Quality Assurance

During the development of this project, we have practiced Test-Driven development. This section will examine the different results for tests.

4.13.1 Unit, UI and integration tests

A total of 180+ tests have been been written for the frontend and backend parts of the system. Writing tests led us to discover logical faults within our system early on. For instance, some tests made us discover that a voter was able to cast multiple votes on the same ballot and were able to modify the vote after it had been cast. The tests have also been a safeguard when refactoring the core components of the system. Most of the tests on the backend are unit (3.7.5) and integration (3.7.6) tests. For the frontend, UI tests (3.7.1) were written together with unit tests. Integration tests for our backend included both API testing and integration with the database.

Automating tests All test suites have been possible to run locally, but for ensuring the quality of the code, the test suites are run on every opened pull request with GitHub Actions (3.7.7), see section 4.11.

4.13.2 Usability testing

Two qualitative usability tests were performed during the project. The tests were performed on two different versions of the application. The usability test plan can be seen in Appendix D.1. The usability test plan had tasks (2.8.6.1) the participant (2.8.6) should perform. A summary of the feedback is given here:

- Participants were in two separate age group
- Most tasks were performed within expected time
- Not easy to see the ballot form, since it was at the right side.
- Difficult to remember/understand that you need to click OK after picking s. Did not understand eligible voter input. Bad spacing in election form.
- Knowing what "push ballot" does and is. What it means. Is not that intuitive. Maybe use a simpler language

4.13.3 Large-scale User Test

When the application was up and running on `anovote.app`, we wanted to perform a large-scale user test (3.7.4). The test occurred with 8 contestants. Out of this group, one inexperienced was selected as an election organizer. The rest (3 inexperienced, 4 experienced) were registered as eligible voters. The rest of this subsection will present some of the results from performing the large-scale user test.

4.13.3.1 Quantitative data

- 10/10 tasks by the election organizer performed within the expected time.
- 6/7 eligible voters were able to vote in the election, giving a success rate of ~ 86%.

4.13.3.2 Qualitative data

A summary of the qualitative data was gathered from answers from the large-scale user test questionnaire, see Appendix D.

Election organizer

- Design was easy and simple, Nice looking
- Easy to start election.
- Easy to push ballot.
- Did not notice tooltip/tips indicators
- Difficult to understand some form fields. "What are they there for?"
- Small title/inseparable title for ballot creation.
- Missed overview over all eligible voters for an election in progress. "I wish there was a count showing 6/7 connected eligible voters".
- Missed onboarding on pushing ballots.

Voter

- One eligible voter were not able to participate because of an unexpected problem.
- Design was small with larger screens.

4.13.3.3 Reported issue

One contestant had a "_" in his personal email address. This resulted in his autogenerated encrypted verification code (4.6.3.2) sent via mail failing.

4.13.4 Code formating and linting

We have used a formater and linter, that has been configured to run when files are saved. This has resulted in clean and nicely formated code base.

4.14 SCRUM

By using SCRUM we have been able to have a structure yet agile development cycle throughout the project. By following the framework, we have had regular meetings for discussing the progress. The tools that the framework provides, e.g., product backlog, have proven useful and effective when trying to organize work. We will continue by discussion some results of using SCRUM.

4.14.1 Work estimation

Work estimation was done in plenum during the SCRUM planning meeting. Story points provided a flexible and predictable approach for estimating workload. At the start of the project, we struggled with estimating tasks. This has changed during the project and we have been able to sharpen our ability to make estimations.

We have also been more able to predict our work velocity with higher accuracy as the project went on. At this point, we are able to have about 50 points at the beginning of each sprint, and still manage to finish all our tasks.

As described earlier (3.3.5) we have used two different approaches for estimating the story points. At the beginning of the project, task estimation would be given often before the task was even described or right after the description. Usually this was done by the person that had submitted the issue. This resulted in a lack of discussion regarding the task and the team would often move fast forward in this process. We will discuss some drawbacks of this approach in 5.13.1.

During a period of the pandemic outbreak, the team had to resolve to home office and internet calls during planning meetings. To make sure that all opinions were counted for, we introduced planning poker. This "forced" all team members to have an opinion on the topic. We have found this method to be the most effective and accurate.

4.14.2 Distribution of work

Throughout the whole project, we have distributed all work evenly between the members. For the most part all members have been able to solve tasks as assigned. Only occasionally, when some members where not able to fulfill all their tasks on time, have tasks been reassigned. There has also been a strong sense of collaboration and a culture of both asking for help when necessary, and giving support when needed or asked for.

4.14.3 Sprint

Sprints were run for one week, resulting in a high frequency of SCRUM meetings. Due to the short sprint's we have been able to calibrate our work load frequently. By calibrating often we have been able to quickly get a good estimate for our velocity. We have also been able to prioritize tasks more often, helping us reach a Minimum Viable Product (MVP) earlier.

4.14.4 Stand ups

Stand ups have been held standing. Each team member described what he had been working on, what may hinder further development and the continued work until the next sprint. We have managed to have these meetings every day. Due to COVID restrictions, these meetings have sometimes been held online. We have managed to keep to the time-boxing of 15 minutes. When the meeting has happened online, there has been a tendency for the meeting to also include socializing, which otherwise happens naturally in an office environment.

4.15 Version control and code collaboration

Below we present the results of using version control and associated technologies, and how it has enabled us to collaborate on a common code base.

4.15.1 Git, GitFlow and GitHub

By using git and GitHub we have been able to keep control of our code base and have a reliable platform for storing and managing the code. The GitFlow extension has provided a clean and predictable branching strategy that was easy to learn and follow.

For keeping control over the git branches, we have used the following branch model:

- **Main -** Holds the latest release.
- **Develop -** The branch that holds the recent changes and fixes.
- **Release -** Created for every new release, deleted when merged into main.
- **Feature -** Created for every feature that is being worked on, deleted when merged into develop.
- **Bugfix -** Created for every bug which is being fixed, deleted when merged into develop.

4.15.2 GitHub Issues

GitHub issues was used as the primary method for describing tasks and linking these to a repository. Some issues were related to several repositories. These issues were either split, with duplicate titles and descriptions, or associated PR where linked across repositories to the relevant issue.

4.15.3 Pull Request

Pull requests have provided a clean and controlled environment to give and receive feedback on code changes. GitHub has provided all the necessary features to do code review in a productive manor.

4.15.4 Semantic versioning

We have added versioning to our application by using the GitFlow release feature. When finishing a release branch, a new git tag is added to the project and pushed to GitHub. This in turn will trigger a GHA workflow that generates a release on GitHub, consisting of the source code and a change log, describing all changes between releases.

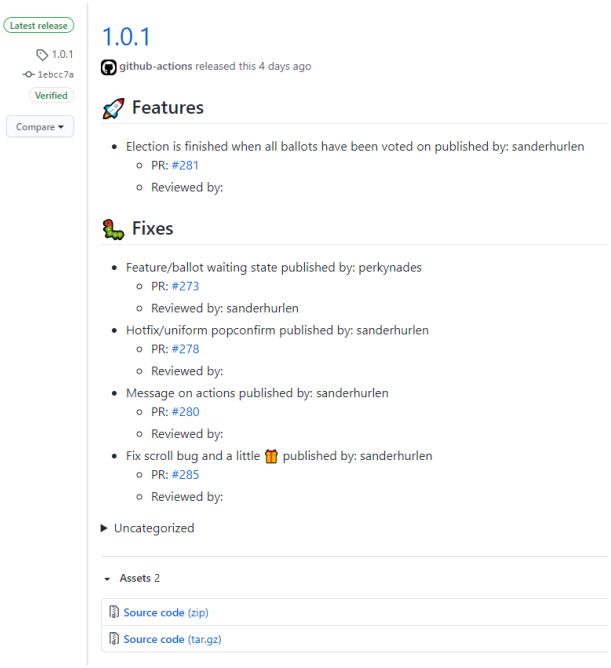


Figure 4.17: Autogenerated release

4.16 Project developed tools

As our project evolved, we saw that some aspects of the organizational work were repetitive, error prone and could be improved. This led us to develop a set of tools and scripts to make our organizational requirements easier and unify common tasks.

4.16.1 GitHub SCRUM chrome extension

We developed a Google Chrome extension for GitHub to provide extra functionality that GitHub Project does not provide and add some issue/pull request helpers. As GitHub Project does not provide any tool to have story points, we used GitHub's label system to provide labels for story points. These labels were attached to an issue/use story. As this is a creative way of providing story points for issues/user stories, we would lose control of distributed points. Thus, we developed an extension that parses the work board columns for story points (Sprint backlog, In progress and ready for review). Columns to parse is configurable in the extension menu. It inserts a total story points counter for all tasks in the sprint backlog(Marker 2 in Figure 4.18) along with the number of issues. It also creates cards for all persons that have an issue assigned to them in any of the columns, and displays how many points each person has in each of them (Marker 1 in Figure 4.18). As cards are moved to other columns, all points are recalculated automatically and each persons story point count and sprint backlog total count are updated.

Our extension will also provide reminder functionality when submitting a new pull or issue. For an issue or pull request to be added automatically to our workboard, we have to assign it to a project. We saw that we forgot this sometimes, so we implemented a large text warning when creating an issue, and disabled create issue/pull request button and keyboard enter button until a project is selected. The select project popup menu is automatically opened on these pages. When a project is selected, the create button and enter key are re-enabled 4.19.

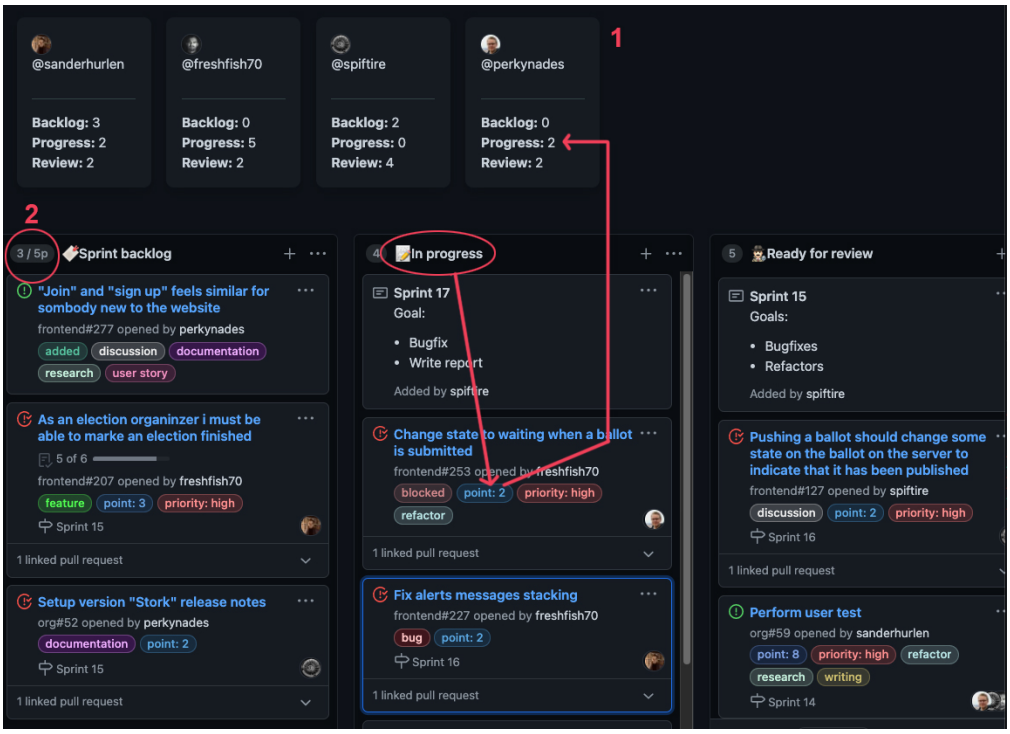


Figure 4.18: Scrum master work board preview

4.16.2 Environment setup

Our project uses Docker extensively in testing, development, and production. We therefore saw the need for creating a utility script that would let us start, stop, and rebuild these environments without having to remember the required commands, arguments, and options for each.

4.16.2.1 Anovote CLI

anovote is a shell script that provides commands to start, stop, and build our frontend or backend environment. It combines docker-compose commands with a set of arguments and links different docker-compose.yml files for a given environments. Development environment is started with:

```
anovote dev
```

Listing 4.18: Starting of development environment

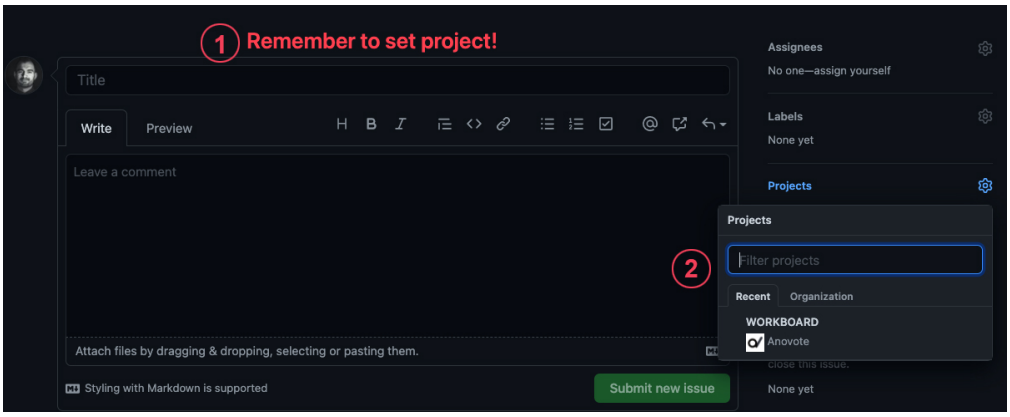


Figure 4.19: Scrum master issue/pull request preview

Development environment is re built with:

```
anovote dev --build
```

Listing 4.19: Re-building of development environment

4.16.2.2 start-production

start-production is a shell script that is responsible for starting the whole production setup on a production server. It generates SSL certificates through LetsEncrypt for our reverse proxy, creates all necessary folders, moves config files, clones the frontend and builds it, pulls and builds the backend and starts all docker services. The tool also comes with some argument options.

To start production from scratch with new TLS certificates generation and building:

```
start-production.sh --cert --build
```

Listing 4.20: Start and build production with certificates

To rebuild only backend and frontend:

To restart production:

```
start-production.sh --build
```

Listing 4.21: Start and build production

```
start-production.sh
```

Listing 4.22: Restart production

4.16.3 Document generator

For our weekly sprint planning, review and retrospective meetings, we were using the same document templates for documenting the meetings. We therefore created a JavaScript utility to generate these files for us for a given sprint, and organize them in a folder structure. It produces markdown files for the respective sprint and populates each document with a template. This script can also generate meeting agenda and summary documents with week and date named documents with a fixed template.

To create documents for a new sprint 4.20

```
ano -s
```

Listing 4.23: Generate documents for sprint

To create documents for a general meeting 4.21

```
ano -m
```

Listing 4.24: Generate documents for meetings

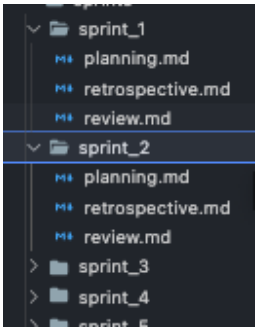


Figure 4.20: Generated sprint documets

4.16.4 Docsify sidebar generator

Our organizational documentation is hosted on GitHub page `docs.anovote.app` and makes use of a tool called Docsify to generate a presentable website from markdown files. It has support for navigation display through files called _sidebar.md in folders where this is wanted, which must be linked to through a sidebar.md file in the root directory. We developed a JavaScript utility to generate and populate these sidebar files for our documentation tree of markdown files. Folders that do not contain markdown documentation are ignored by having a _ignore file inside them, which ignores the content of that folder and sub-folders.

To generate files (organization repository) 4.22

```
node navgen.js
```

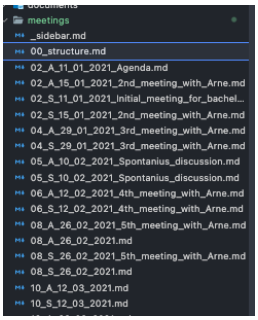


Figure 4.21: Generated meeting documents

Figure 4.22: Output from navigation generation

4.17 Design

At the core of the problem definition of Anovote, lies the definition of being able to organize elections with a simple and elegant user interface. Early in the development process, sketches of how the user interface for organizing and participating in elections were created.

4.17.1 Wireframing

After brainstorming ideas, we could start by creating wireframes for the different stakeholders (see 4.2) of Anovote.

4.17.1.1 Voter perspective

We developed wireframes from the voter perspective first. The main flow of a voter's perspective includes; *joining, authenticating, waiting, voting, results/logout.* As figure 4.23 displays, this flow

is the starting point for developing the experience for the voter. The voter perspective was developed with a "mobile first" mentality (2.12.2), as we predicted that most voters would participate from their smartphones.

Figure 4.23: The main flow for a voter, designed with wireframes.

Alternatives An important part of designing wireframes is the ability to quickly generate structural elements that suggests how the design is working. As a part of this, different wireframe designs and alternatives were considered and developed. Figure 4.23 showcases all the alternatives that were created for the voter.

Candidate list A major consideration for the wireframes went into finding out how the candidates should be displayed in such a way that it (1) suggests to the voter that you may not see all candidates, (2) the order of the candidates should not influence the voters consideration (3) the order of the candidates are equally worth. As a part of this, two main designs for displaying candidates were presented:

Figure 4.24: Variants proposed for choosing how candidates should be displayed. "Row design" to the left.

After considering both alternatives, we decided to go forward with the row alternative.

Ballot information Getting all information for a ballot is essential. Knowing what you are casting your vote on is vital, hence the title should be very visible for the voter. Initially, ballots could have a title, description, and image at maximum. With description and image being optional. We created wireframe mockups for some of the cases, to display how it would look for the voter in different scenarios.

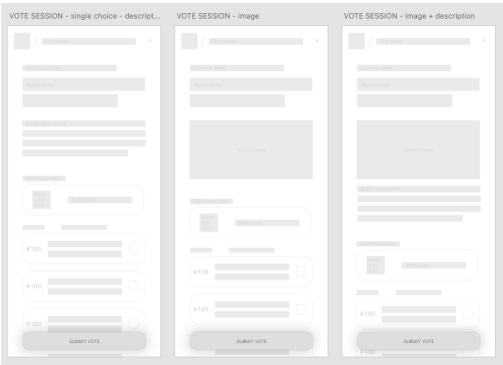


Figure 4.25: All cases of displaying information of the ballot

4.17.1.2 Organizer perspective

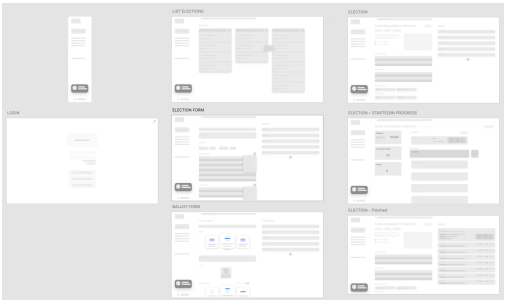


Figure 4.26: Wireframes of all use cases for the organizer

Wireframe was also developed for the organizer panel. UI design mockups have been created for the organizer panel. A full view of the UI design of the organizer panel can be seen in Appendix A.3. The design of the organizer panel have been crafted to comply with Don Normans Design Principles (see 2.12.1.

4.17.1.3 Landing and about page

Wireframes and UI design mockups were produced for the landing and about page, see Appendix A.

Chapter 5

Discussion

In this chapter we will examine the work and decisions that has been done during the project. We will look at how the project have been developed, the results we have made and the complications we have met along the way. We will also point out how the ongoing COVID-19 pandemic has affected our work, and the adaptations we have had to make as a part of this.

5.1 General result

Through a methodical and structured approach, applied knowledge and hard work, we have been able to deliver on the goals of our application (1.3). In our preliminary project plan we made a list of features for our application (C). Some of these features we have not been able to implement (see 5.2). This has mostly been due to time constrains. We had agreed to spend about six hours a day, five days a week working on the project, excluding holidays and exam preparation. Early on in the project we expanded the working hours in order to meet our project exceptions. On average we have been working on the project eight hours a day. We clearly overestimated the amount of work we would be able to in order to complete the project. Still, we have manage to create an application that is production ready and available to the public.

5.2 Functionality

While we already have implemented a lot of core logic, there are still features we have started work on but not finished. Below we discuss some of these missing features. We will also briefly discuss some benefits of the logic we have implemented.

5.2.1 Election

We wanted ballots to have options both for how they could be voted on and how the results should be displayed.

We had plans to implement "select multiple" and "ranked" voting options. "Select multiple" would allow voters to vote on more than one candidate, "ranked" would allow the voter to rank all candidates. As "result display"-options we wanted the organizer to be able to display "none" or "ranked". The "none" option would not display who won, and "ranked" would show the top three candidates. As of now, only a single candidate is activated, and the ballot results can only be displayed with a single winner. Only the organizer can view the results. We had plans to enable publishing the results to all voters.

Automatic push ballot It is possible to set a start date for an election. This would change the state of the election automatically from *Not Started* to *Started*. In reality this feature has no real benefit to the organizer as he/she would still need to manually push ballots to the voters. We have plans to elevate this functionality where ballots would be pushed automatically to voters.

5.2.2 Election Life cycle

The election life cycle states provide an intuitive state management for each election. The states make it easy to validate if voters can enter an election or not, what components should be rendered by the organizer and what can be edited.

There are still some edge cases that we have not addressed. E.g., it is possible for an election to be started without any voters or ballots. This does not make any sense in the real world as there is no use in voting if there are no ballots, and no use in holding an election if there are no voters.

5.2.2.1 CRON jobs

We used CRON jobs to check if an election was supposed to be open or closed. The CRON jobs check each election every minute, which lead to only one job running at a time and was fast due to a minimal amount of SQL queries, the only cost being elections not opening or closing on time. We could also implement the CRON job to only run when needed, which would lead to elections opening or closing on time. However, this could lead to the possibility of many CRON jobs waiting to do work, allocating unnecessary resources. Since the open and closing times of an election are not time-sensitive, we kept using the first approach.

5.3 Voting Implementation

As Haines et. al [114] have concluded several times, developing a secure and bug-free implementation of an electronic voting system is a difficult and time-consuming task. There are also wide varieties of possible algorithms and protocols that are viable candidates for such a system. Choosing an established and mature solution is also a time-consuming task. With recommendations from our supervisor, we were advised to move forward with the project and make the basic application work without a proper secure and anonymous protocol implemented.

We had from the very start looked at different design patterns to be able to easily swap out the voting protocol if we needed to. Since the protocols work in different ways, easily swapping one for another has proven difficult. We decided that it would be easiest and possibly wisest to only rely on one voting protocol and build the solution around this decision.

To make sure that there is only one vote per voter, the system needs to keep track of who has voted. The way the system works today it is possible to determine who cast what vote. This

would take some effort, but the system is not truly anonymous. There had to be made additional work to make sure that the votes would not be traceable but at the same time make sure only one vote per voter and still make the voters able to verify that their vote was cast.

There is also a security hole in the system that we will briefly discuss. Lets say that a person A is eligible to vote in an election. If person B enters the credentials of person A into the "join election"-form, an email will be sent to person A. If person A is not cautious and clicks the verification link, person B will be able to join the election with the identity of person A. This will happen if and only if A has not joined the election already.

The system can not be said to be truly anonymous with the current implemented solution. A vote can be traced back to the server side by cross-linking votes to election ballots. This is only possible if you have full control of the system, i.e., the database and server logs. This requires the voters to trust the platform. A vote is however protected during transmitting since all communication happen over secure connections. Thus, any vote sent to the server can not be altered or eavesdropped during transmission. The election organizer has no access or authority to view any submitted vote data, except for the candidate vote count for a given ballot. Vote counts are only accessible by the owner of the ballot, i.e., the election organizer.

Another issue is that a voter has no option to verify or validate that their vote was submitted correctly or even part of the tally.

5.3.1 Blockchain as an Election Mechanism

Blockchain would be a viable option to hold immutable and anonymous elections. The requirements of running applications on the etherum blockchain proved to be too high. The main issues are due to network fees, but also processing speed. Voters and election organizers would be required to own Etherum and spend portions of it for any action on the platform. Processing speed could vary on the network, which could provide issues when using the public ethermum chain. The etherum network can only perform 15-30 transactions each second, which can result in large delays for our real-time needs.

5.4 Client-server communication

As described in section 4.5, we have implemented two different modes of communication between the clients and the server. While it would be possible to run all communication through just one of these two modes, both have their drawbacks. HTTP requests to a Rest API is really good when the result of a request only affects two parties, the client and the server. When there is a need to notify several clients of a change, the Rest API cannot handle this, as it has no way of reaching out to clients. Through basic http requests the server is only ever able to respond to incoming request.

Web Socket on the other hand is able to conduct bi-directional communication, were both the server and the client can initiate data transfer. WebSockets are also able to communicate with multiple clients at ones, making updating data across all clients in real time. To get the same functionality using HTTP request the client has to resolve to HTTP Polling. There are significantly less overhead when communication over a WebSocket compared to HTTP request. One issue with websockets is that it is hard to scale them horizontally [105]. If there would be a lot of users using the system simultaneously, a WebSocket server might struggle to deliver. Spinning up an extra WebSocket server means that the developer has to syncronize the events across multiple servers in order to reach all clients. In the same scenario adding an additional HTTP server is easy, and comes at very little additional cost to the developer.

In our application, we probably would suffice with using just WebSockets as our mode of communication. We knew from the get go that we had to use WebSockets in order to get the immediate response callback functionality that we wanted. Our choice to use both HTTP and WebSockets, came as less thought out process. Implementing the HTTP request endpoints has probably added some extra overhead to our application. On the other hand, HTTP calls are easy to test, build and manage. With the added benefit of being able to scale well, we don't see the added work as a waste, but rather as a possibility to learn common technology and an investment in future redundancy for the application.

5.5 Framework decision

We started by using Deno as the runtime environment for both frontend and backend as it improved upon Node and were developed by the same person. However, when using Deno, we kept running into issues with third party libraries that were premature for professional development. The main issue was the Deno port of TypeORM, as it did not have full support for a PostgreSQL database. This led us to discuss the choice of runtime environment.

To stay with Deno, we would have to switch from PostgreSQL to MySQL for the database, which might lead to further issues. Porting to Node would take a full sprint to perform, taking time away from implementing new features.

As we had already met issues with third-party libraries being poorly ported, we did not have confidence in the Deno eco-system being ready for professional adoption. We decided to port from Deno to Node, as the time spent porting would be gained by the time spent dealing with third party issues [57].

When it came to frameworks and libraries that ran on the runtime environment, we felt like we made the right decision. React and Ant Design let us build a solid and good looking web UI without much hassle. On the server, we also felt like we had made the right decision, TypeORM made the management of data effortless, Node, express, and Socket.io made the communication with the frontend easy.

5.5.1 Database handling

Deciding to use an ORM has proven to be invaluable. The alternative would have been to setup the database by writing and running all SQL statements manually. This means writing all table creation and crud queries from scratch. Writing complicated SQL statements is both time consuming and prone to errors, possibly resulting in bugs. When any of our entity classes would change, all relevant queries would have needed to be updated, adding additional work to any refactoring, possibly introducing new bugs. By using an ORM, we could simply forget about this

problem and let the ORM handle all SQL statements.

Another key benefit of using an ORM is that it uses prepared statements by default, mitigating any SQL injection attacks on our system.

5.6 Third-party Library

As described earlier, by using third-party libraries, we were able to create a richer and more secure application in less time. Other libraries we would not have been able to create with our current knowledge, and the application would have been impossible to create without these libraries. Especially Socket.io [102] was crucial for the application. This library has taken years to develop and requires knowledge of internet protocols that we do not possess.

Bcrypt (3.5.13.4) [9] was used for hashing passwords before saving them to the database. Even though we understand the basics of the hashing algorithm and with enough time and effort would have been able to write code with similar functionality, there is a high probability that we would have implemented bugs and errors during development. We therefore found it wisest to use tested and approved third-party libraries when doing operations that revolve around securing the applications. Another example of this is helmet [37]. It's a simple library that changes the http headers of server responses. While we could have set all these manually, and thus did not depend on yet another library, having a middleware that is already tested and widely used, made us more confident that the settings would be appropriate. By using such libraries, it enabled us to move ahead quicker and focus our attention on other key mechanics that could not be solved by external libraries.

While using external libraries, there are some potential drawbacks that we needed to consider. These drawbacks have been described in 2.6.8. While in early development, we tried to use Deno as our main runtime environment on the backend. A lot of third-party libraries for Deno, i.e., typeORM, were not properly developed and had several bugs or missing features. This dependency therefore made our development halt.

This kind of dependency block is not sustainable for a project like ours, where the limited time for development is critical. We could simply not wait for the library to be patched and fixed sometime in the future. After moving away from Deno and over to Node, we had not experienced any of the drawbacks described earlier. We see this is one of the main advantages of working on established, well supported and widely used frameworks and technologies.

5.7 Localizing our application

Currently, the application does not support other languages than English. In our requirement specification (see Appendix C), we wanted to localize our application to a minimum of two languages, English and Norwegian. This requirement has not been fulfilled. Localizing an application takes time to implement and has sometimes been a pain to take into consideration. We do however, believe that the time and effort of localizing the application from the start, prove to be a better option than possibly refactoring all text at a later point. At the state the application is in now, adding additional languages just requires us to add JSON files with translations for all keys in the application. This task can also be outsourced on different platforms online, and requires very little technical skills to resolve.

5.8 Code Style Aiding

Early in the project, we decided to use a common code style and linting to aid our coding. Agreeing on a common code style can be a root of discussion and argument within software teams. We also had minor discussions of how we wanted to format our files. This dispute was settled by majority rule. Making prettier enforce these rules on save has helped us to comply.

The linter will show information in the IDE according to the rules applied. The warnings has for the most part been of great help. We have, however, experienced that some warnings has not been relevant, or unfixable. E.g., we have had a rule saying that we can not declare a variable as the any type. When working with third-party libraries this has sometimes been avoidable, as we

could not predict what datatype it would receive and use. In these cases we have had to disable the linter for that specific part of the code.

5.9 Version Control and Code Management

Version control has proven to be a crucial tool for software development as a team. Git has provided the necessary functionality to provide a secure way of merging work from several sources. With the framework that git flow has provided, managing and controlling different branches has made the process a lot smoother.

The integration between GitHub and git has also made working with the project from different places easier. We have been able to develop on our computers, create pull requests on GitHub, and then review the changes on our phone. All this has been a simple solution.

5.9.1 Semantic versioning

We chose to use the GitFlow toolkit, as it gave us a structured strategy for creating and deleting branches while building out the application. When the GitFlow commands were run correctly, they made working on multiple branches elegant and easy.

By adding the versioning to our application, we have made a foundation for keeping track of releases. We found the workflow to be simple to follow and easy to execute. Combined with GitHub Actions that produce changelog statements, versioning has become a quite powerful tool.

The author and designer of the GitFlow workflow does not recommend to use the release feature for building web apps, as web apps should be CI. He recommends using the GitHub flow method. The GitHub flow method is a simplified version of the GitFlow workflow [108].

We found that by using the release workflow, it could mark clear milestones. When combined with auto-generated changelog, it gives a quick indication of the changes that have been made

since the last release.

We have found that, even when developing a web app, using GitFlow can organize branching in a structured and predictable way. Adding GitFlow does not really add any drawbacks, it only adds tools that might not be used as often, or used for other purposes than intended, e.g., trigger changelog workflows.

5.10 Self Developed Tools

During our work, we developed a few tools and scripts primarily to automate tasks. We will discuss the reasoning for this extra work and if the effort was worth it.

5.10.1 GitHub Scrum Master

While GitHub has several built-in tools to help aid agile development processes, it has not been developed specifically to facilitate the SCRUM approach. There are other platforms that are better suited for SCRUM, like Atlassian Jira [103].

GitHub was chosen as we liked how code, issues, comments, pull-requests, and workboards could coexist in the same ecosystem. We did not know of any other (free) platform that made this possible. Because we had chosen to use the SCRUM, we saw the need to adapt GitHub to our needs.

The adaption was done through a creative use of the label feature in GitHub, and building an extension for the Chrome Browser (4.16.1). This has made our GitHub project board more SCRUM friendly and management easier. In sprint planning, and throughout the sprint, it provided a good overview of how many story points had been estimated and points distributed. We feel that the extra work developing the extension was well worth it, since it has resulted in less overhead when conducting sprint planning, and have provided an overview of how each team member was doing story point wise.

5.10.2 Anovote CLI

During development, we used Docker containers to run the different parts of the application (see section 3.5.15 for more details). Controlling containers can be done through a CLI. The commands can often become long and complicated, with several parameters that have to be set in order for the containers to start with the right configuration. These commands can also vary based on the environment they are run in (development, production, testing), what they should do (starting, stopping, cleaning, building), and are often run multiple times a day. To mitigate the possibility of error and to provide an easier way of starting and stopping containers, a simple bash script was written.

The team would use this script to a varying degree. Some might only use the script in certain situations, e.g., when starting services on the server, while some use the CLI extensively. We found the script to be useful and a skill that is good to master. However, the Node package file also comes with the ability to create custom commands, leaving the anovote cli redundant.

5.10.3 start-production script

To automate our deployment on the server, we utilized GHA workflows to log onto the server over ssh and run commands. Putting commands into a workflow file and then testing them can be quite cumbersome and difficult. To avoid this, we wrote a simple script that the workflow could run when it got onto the server. This script could then be tested on its own, making it easier to see if the results were what we expected. The tool for starting a full production environment makes it very easy to setup a production environment on a new server, as well as providing continuous delivery to an already running system. The start-production script has proven very effective and has made our CI/CD workflow easy and reliable.

5.10.4 Document generator

During all meetings we have made meeting summaries. To make documents have the same structure, we have written a document generation script. This script has been used every week and has made sure that all documents uses the same layout, inserting the correct dates and naming convention for files.

Overall, we found that the extra tools we have created have made our day simpler and repetitive tasks automated. While these tools take time to develop, the time saved by the tools quickly adds up, leaving more time to focus on resolving sprint goals. We think that the additional time spent on developing these tools has come to good use for all members in the project, in the form of providing uniformity, error prevention, and reducing time spent on wrongly used commands or configurations.

5.11 Testing

During the development, spending time on writing tests was without a good time investment. The time spent on writing tests was gained in time that would have been spent on fixing bugs that spiraled out of control since they were not spotted early enough. When writing tests, the team would also be able to develop the features more confidently, as we felt that any new features added would be added safely and without breaking the whole system.

However, there were also a few times we assumed we had good test coverage, which could lead to the team feeling a false sense of security. If we worked on a new feature, we were under the impression that it was safe to implement since the test coverage was good and none of the tests failed. This could result in us forgetting to write tests for the feature, and the faults and/or bugs of that feature would be spotted later than we wanted.

5.11.1 Usability Tests

Due to the restrictions to combat the COVID-19 pandemic, we found it difficult to perform a series of qualitative usability tests. There were only a few times during the project that we had the opportunity to perform one-to-one usability tests. Most of the time, the questionnaire and test plan was not finalized. When we got the chance to complete a qualitative usability test, we found it to be helpful. Some of the feedback we gathered has already been fixed or reworked. In an ideal situation, we would have arranged more usability tests to get an even better understanding of what we should have improved, and how realistic users use the application.

5.11.2 Large-scale User Test

In the one large-scale user test we performed, the most notable feedback was on the graphical design of the application. Some of the titles were small, no onboarding and/or in the form of hints, no overview over all eligible voters, voters in a started election and a lack of intuition on what some form fields are for. This led to changing the graphical design of the solution based on the feedback that was received (see 5.12). The fact that we got a large amount of feedback on the graphical design might indicate that more usability tests should have been performed earlier in the project, before actually implementing the graphical design.

The large-scale user test discovered one critical issue with our implementation of the verification code generation (see 4.6.3.2). We would not be able to discover this issue so fast, if it would not have been for arranging a large-scale user test. The implementation of our verification code generation that is sent via mail used a "_" underscore as a delimiter for the code before it gets encrypted. Since this is an acceptable character for use in mail, the verification of the decoded code would fail because it would delimit the wrong parts of the code. Luckily, this critical issue was fixed shortly after performing the user test.

5.12 Modifying the design

The original design of the Anovote website does not differ drastically from what has been implemented. There are some adjustments that have been made as a part of using Ant Design UI components (see 4.7.2), however such adjustments are to be expected. For designing the voter perspective, a mobile first approach was used. This is because mobile phones are a widespread device and more accessible. When designing the organizer panel, we did not follow the mobile first approach. This led to a lot of unnecessary refactoring of the UI later in the project. As we improved the support for mobile devices, we experienced the overall design and flow of the webpage to improve.

After performing usability tests and large-scale user testing, we got a better understanding of how a realistic user of our product used it. The feedback from these usability tests were especially important. A common pattern in the feedback was that form elements were difficult to understand, and generally that the domain-specific words (election, ballot, eligible voter) were difficult to understand for a Norwegian speaker. This feedback has been added as issues on the workboard. From then have we tried to improve the layout and design of the web site.

5.13 SCRUM

Scrum is a clearly structured and well-defined framework for developing software. During our development, we had to deviate a bit from the core model to make it work for our needs. Still, the framework was adaptable enough to be constructive to our process.

During the whole project, we were missing two key roles in the model, the product owner and the scrum master. The lack of a product owner caused the product to move forward in a less organized manner. The team would agree on what they wanted to work on, even though this might not help the project move forward in an incremental way. As an example, we have built the frontend logic for having several different ballot display options, long before we were even able to display a ballot at all. It is hard to say that the lack of missing roles would prevent this

development from happening. On the other hand, we have experienced that it is easy to lose track of what is essential when only focusing on the end goal and not on the incremental process.

The SCRUM masters main objective is to make the team follow the agile workflow and guide the SCRUM process. During the development of the project, the teamwork has never halted or had any major difficulties, even though we were missing a SCRUM master. There has been times where our focus has drifted during meetings etc, where a SCRUM master might have been helpful to keep the work structured. However, for the most part we are happy that we could manage without this role involved in the team. We can only speculate, but part of this success might be related to our experience working with SCRUM, and our knowledge of the SCRUM masters responsibilities. Because of this knowledge, the team has managed to adapt these responsibilities and use them when necessary.

5.13.1 Work estimation

As already described (see 3.3.5), we have used two different approaches when trying to estimate work. After moving to planning poker, we have experienced that task description and discussion has been deeper and more meaningful. The team members have thereby gained a deeper understanding of the task prior to estimating.

Planning poker is effective at showing how different members evaluate a task. If the team disagreed when presenting their estimation, new questions and discussions about the task could be had. This would often further deepen our understanding of the task and the team would be more able to agree on an estimation.

Using a baseline task as a reference has often helped us being able to predict more accurately. But estimations are always based on the teams collective knowledge. When task are distributed to team members with higher or lower knowledge of the task, estimations might be inaccurate. As said by Radigan: 'Agile estimation is just that: an estimate. Not a blood-oath [132].' When we have not been able to complete tasks they have for the most part been planned for the next sprint, and in some cases been moved to the product backlog.

5.13.2 Motivation, psychology, and team work

The team started the semester with a high expectation and motivation for the project. We established a good group dynamic and openness very early on. This made grounds for us to have open discussions and statements regarding our well-being and acceptance of explaining concerns, and do private errands without any issues. As the project progressed, we felt that time was our biggest hurdle. The time pressure to complete issues within a sprint and the constantly growing backlog of planned features and reported bugs. This led to longer work days week after week, which resulted in strain on some of the team members. One member was starting to feel burnt out midway through the project. The consequence was feeling less motivated and slight irritation. This was communicated and brought up during sprint retrospectives, and the workload was adjusted accordingly to ensure that we could reestablish motivation and engagement.

5.13.3 Remote vs on-location work

Due to current COVID-19 pandemic restrictions, we were working from both the school and remote/home. The effects this had on each team member varied, some team members already enjoyed working from home and had no problems, while others had a strong need to work in an office environment in order to work efficiently.

Stand-up meetings were also effected by this change in the work environment. By recommendation from our supervisor, we performed stand-up meetings standing. This resulted in the meetings being efficient and short. Performing the stand-up from different locations meant doing it online and sitting instead of standing. This made the stand-ups longer, less efficient, and imprecise.

This effect might have a two-part explanation. When standing, the team would finish more quickly since it was uncomfortable to stand, and less likely to be distracted by the computer. When sitting, the team was more comfortable and therefore did not have the urge to finish quickly, leading to discussions lasting longer and sometimes getting distracted. The other reason is that stand-ups where less effective can be related to the need for social interaction. During

work at school, we would often have conversations while developing. When working in isolation from home, stand-up would be a natural time to discuss topics not related to the sprint, sometimes resulting in occupying some of the allocated time.

Chapter 6

Conclusions

This project's goal has been to develop and implement a full-stack digital voting system that allows election organizers to arrange digital elections which evade the need for manual counting and registration of voters. In this report, we have aimed to describe how we went about solving this task.

The process has been demanding, but our work has proven that it is possible to build such an application using TypeScript and Node as the primary technology stack. At the same time, we have experience that relying heavily on cutting edge technology might cause problems. Being able to find alternatives early and replacing the stack when needed is a key take away from the process.

6.1 Problem solving

To resolve all project goals, we have adapted an agile process with the SCRUM framework. We have been working on established frameworks and technologies based on TypeScript and Node. The development team has used the same software for doing work with coordinated linter and formatter. We have used Git and GitHub to manage our code base and we have elevated GitHub to arrange, report, discuss, and plan the upcoming task.

6.2 Recommendations

SCRUM provides a clear and structured framework for developing a software product. Even when deviating from the model, the framework has proven flexible enough to be helpful in moving the project forward. The SCRUM agile approach has been very useful to the team and is recommended for anyone wishing to conduct similar projects.

Git and GitHub have been of great help for organizing code and making collaboration easy. GitHub has also worked great for organizing work. We would recommend the platform for anyone seeking to develop an application.

GitHub has not proven so useful when working with SCRUM, but by exploiting some platform features and developing our own extension, we have been able to make it work for our needs.

6.3 Our Contribution

By the end of the project period, we have been able to deliver a product ready application, which now is available to the greater public.

We have also developed a chrome extension that can make GitHub seem like a SCRUM management tool. The extension is open for anyone to use.

6.4 Further Work

The application still has unimplemented functionality. The most critical part is to make the voting system comply with the criterion described in 2.1.4, and make the voting actually anonymous. There are also several features remaining that we would have liked to be added. Some of them are password protected elections, the possibility to choose more than one candidate and ranking candidates when voting, have different forms of displaying the results, export the results as CSV or other data format, and implement a search bar.

We would also like to upgrade some dependencies to the latest version, especially Socket.io v4 has added new functionality that would benefit the application. The upgrade has not been done as it could break the application.

The application could also use more testing and auditing to reveal bugs and edge cases which we have yet to find and fix.

Appendices

A Wireframes and UI Design

B UML Diagram

C Requirement Specification

D Large-scale user test & Usability test documents

Appendix A

Wireframes and UI design

A.1 Voter wireframe

Figure A.1: All cases of displaying information of the ballot

A.2 Organizer wireframe

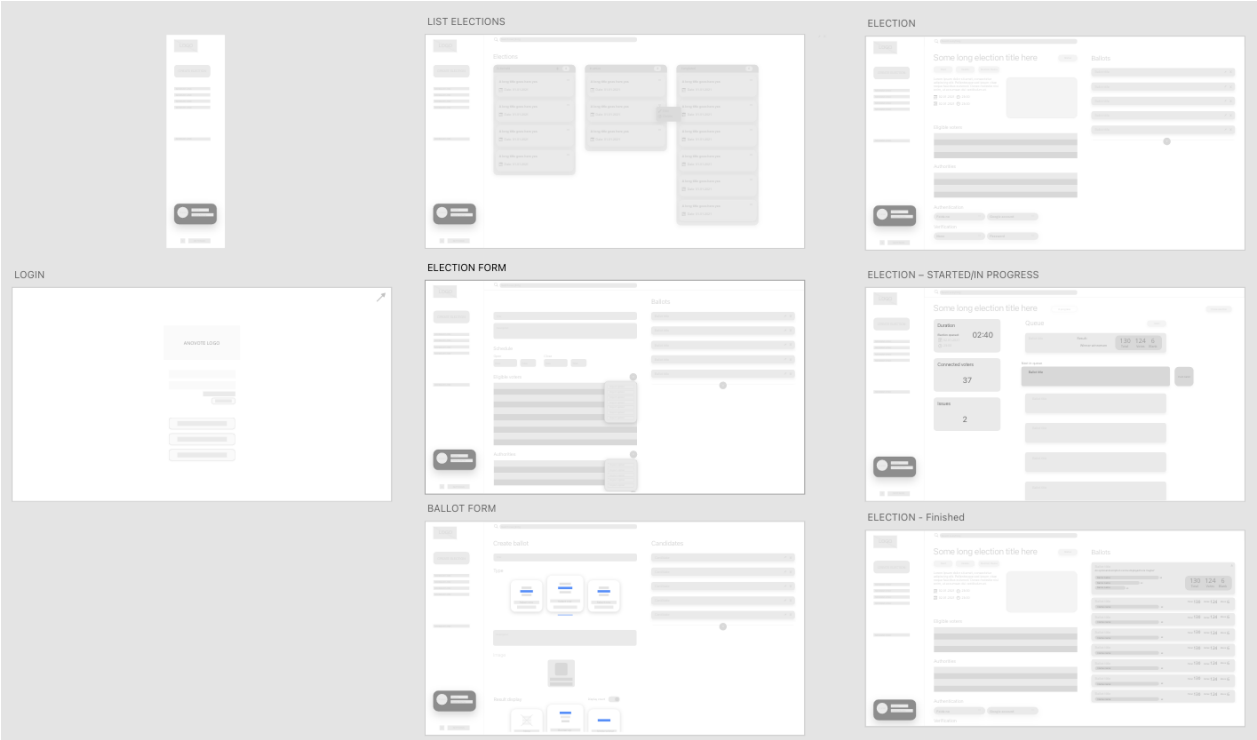


Figure A.2: Wireframes of all use cases for the organizer

A.3 UI design organizer panel

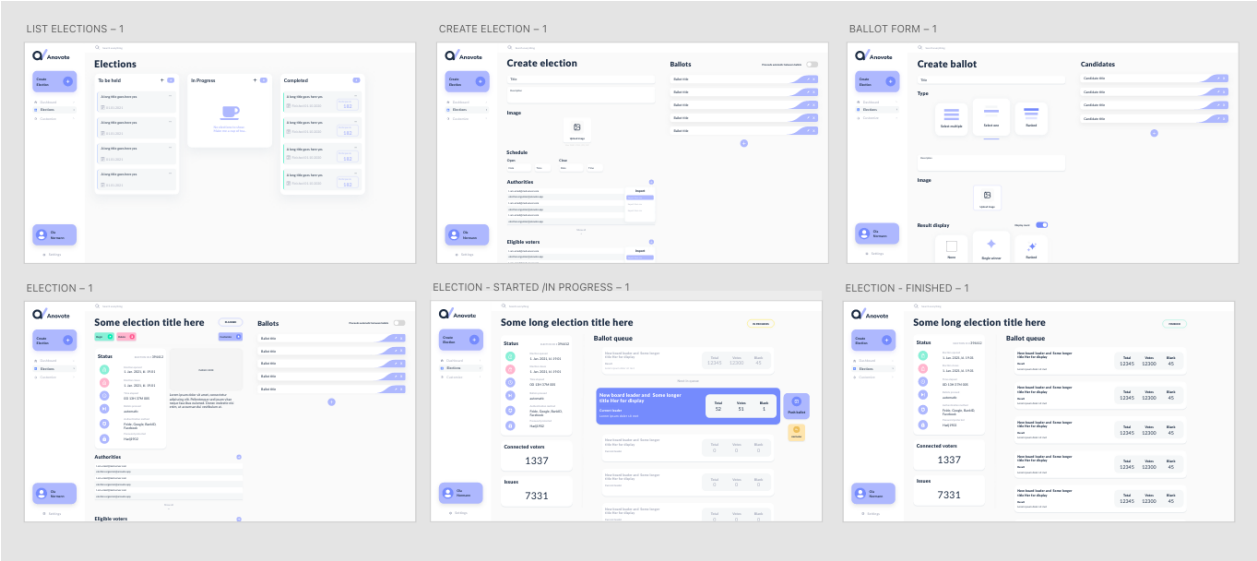


Figure A.3: UI design for Anovote organizer panel

A.4 Landing page wireframe

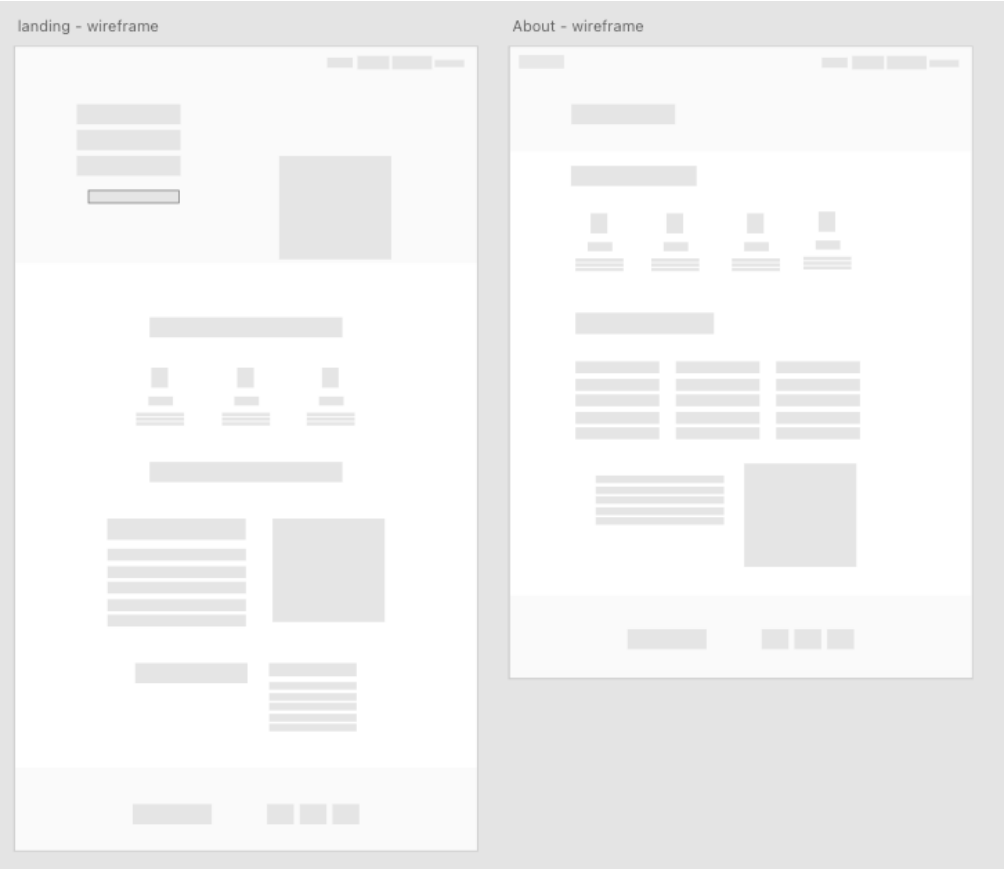


Figure A.4: Wireframes for landing and about page

A.5 Landing page design

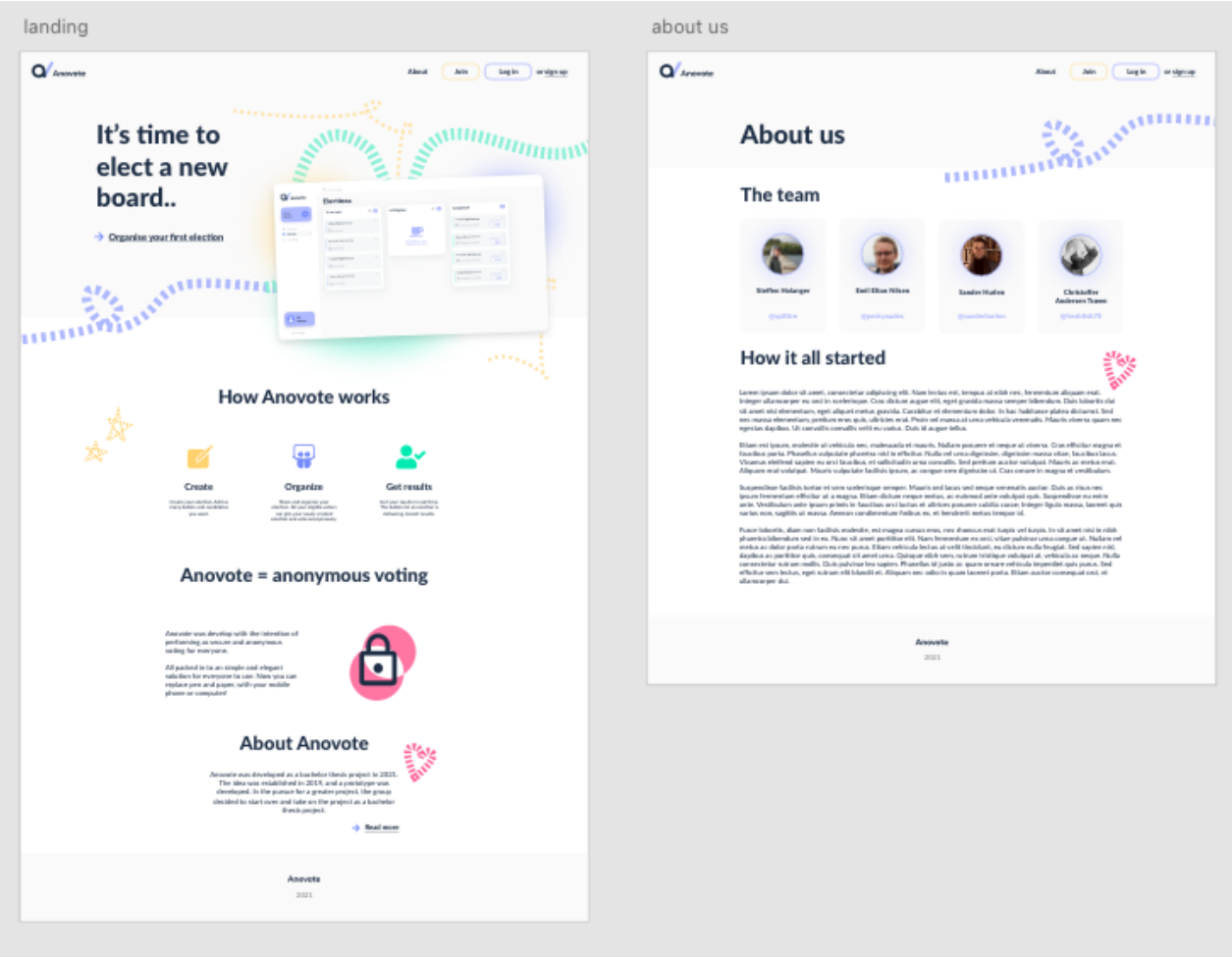


Figure A.5: Landing page design for https://anovote.app

Appendix B

UML diagrams

B.1 Sequence diagrams

The following diagram illustrates the join/verification mechanism. It shows the events, decisions, and calls that are executed down the application chain from when a voter submits the join button until they are verified. A full scale image can be seen in (/attachments/figures/uml/join-sequence.png)

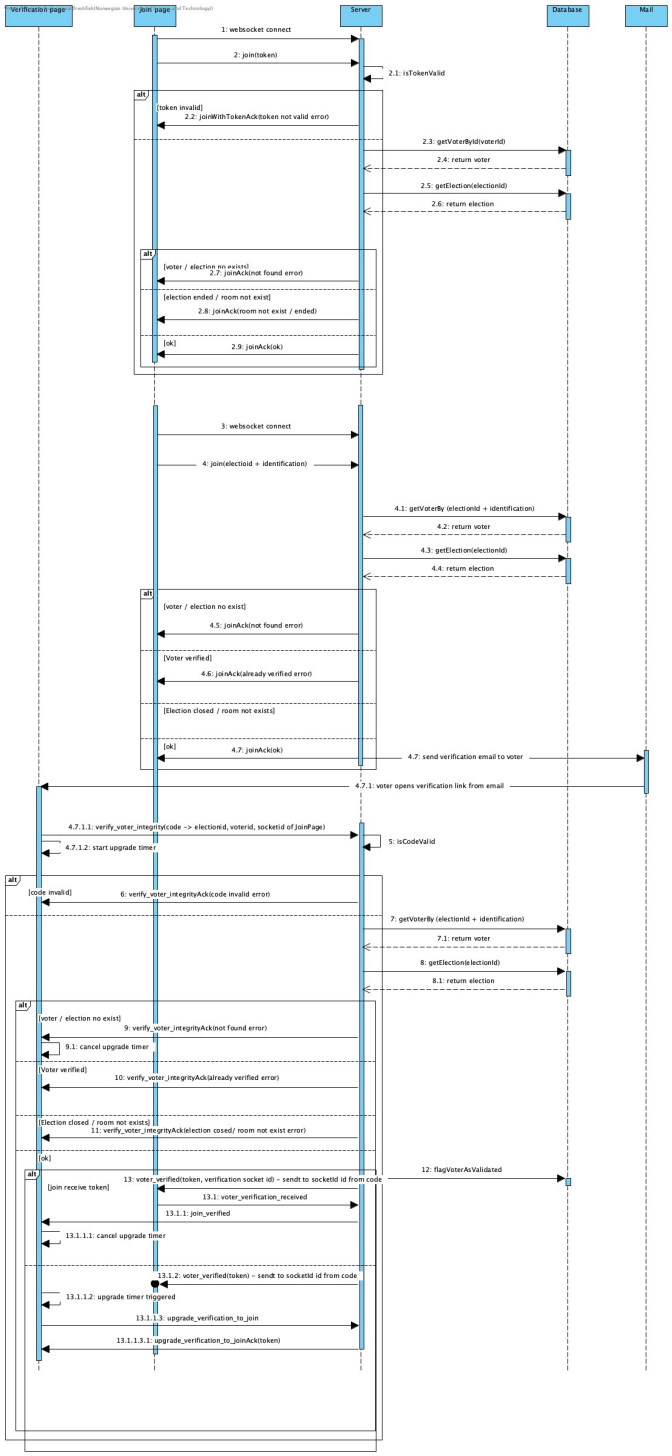


Figure B.1: Join and verification sequence diagram

The following diagram illustrates the vote sequence from when an election organizer pushes a ballot to a tallied ballot. A full scale image can be seen in (/attachments/figures/uml/vote-sequence.png)

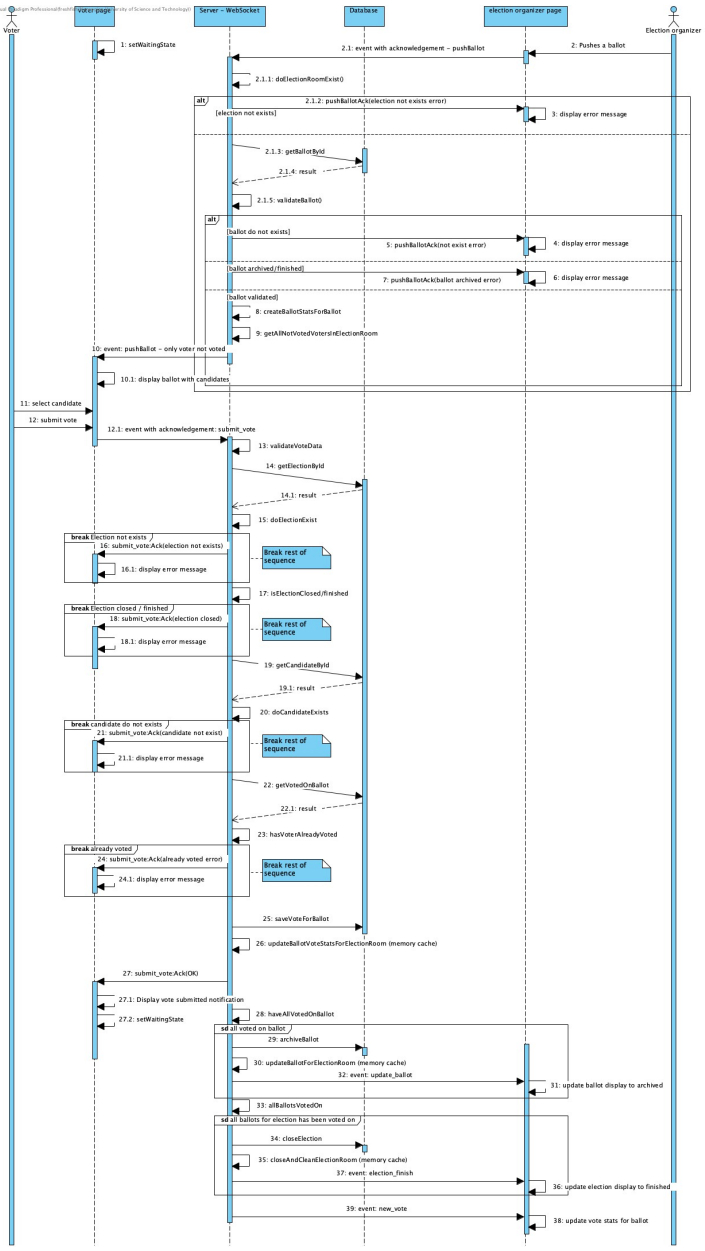


Figure B.2: Vote sequence diagram

Appendix C

Requirement Specification

A Voter features

Specification	*Importance*
A voter must be able to vote	Must have
A voter must be able to login	Must have
A voter should be able to view the result of each vote if enabled by the vote caster	Nice to have
A voter must be able to close and open the voting service and persist sessions	Must have
A voter must be able to logout	Must have
A voter should be able to change language	Nice to have

B Election administration panel features

Specification	*Importance*
An election organizer must be able to login	Must have
An election organizer must be able to logout	Must have
An election organizer must be able to register	Must have
An election organizer must be able to create a new election	Must have
An election organizer must be able to invite users to an election	Must have
An election organizer must be able to see casted vote results	Must have
An election organizer must be able to create an election template	Must have
An election organizer should be able to change email and password	Nice to have
An election organizer should be able to export election results	Should have
An election organizer should be able to see the current ballot results in real time	Should have
An election organizer should be able to delete an election	Should have
An election organizer should be able to change the theme of an election	Should have
An election organizer should be able to change the language	Should have

C Ballot features

Specification	*Importance*
A ballot must have a title	Must have
A ballot should have a description	Should have
A ballot can have an image	Nice to have
A ballot could be cast as blank	Must have
A ballot can be of different types	Must have
A ballot must be easy to understand what is being voted on	Must have

D Election features

Specification	*Importance*
An election must have a title	Must have
An election should have a description	Should have
An election must have a list of ballots	Must have
An election should have a progress bar	Should have
An election must be able to push ballots to voters	Must have
An election should have a due time	Should have

E System features

Specification	*Importance*
The votes must be untraceable	Must have
The voter must be validated	Must have
The system should support multiple login options	Should have

Appendix D

Test Documents

D.1 Usability Test Documents

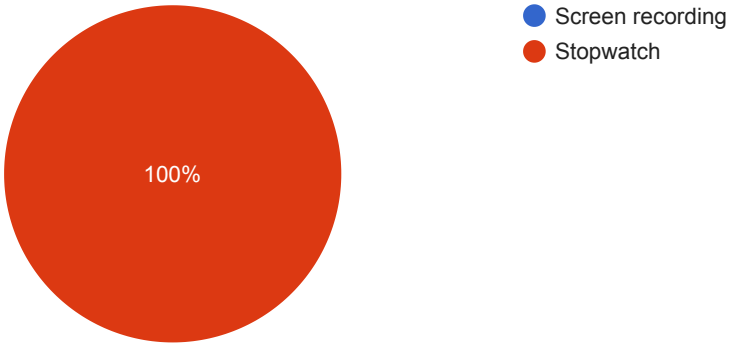
Useability testing

2 responses

Publish analytics

FOR TESTER: How are you recording time?

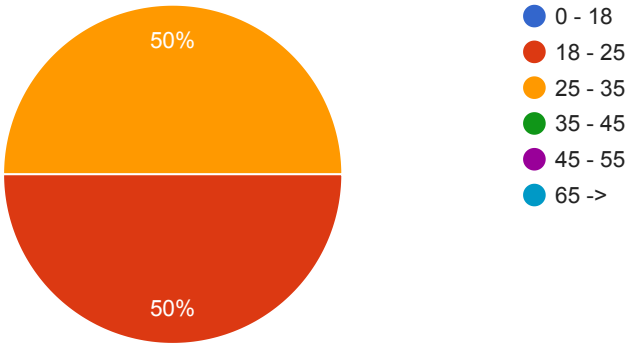
2 responses



Screening

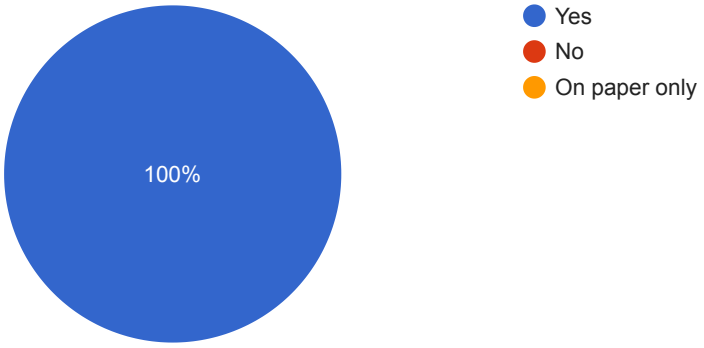
Which age group do you belong to?

2 responses



Have you ever arranged a digital election or a quiz of any kind before? (by using menti, strawpoll or kahoot for instance?)

2 responses

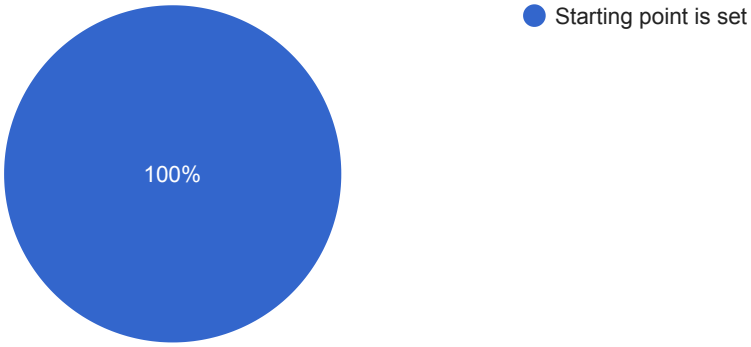


Testing

Information for tester

Starting point for test: http://anovote.uials.no:3001/login

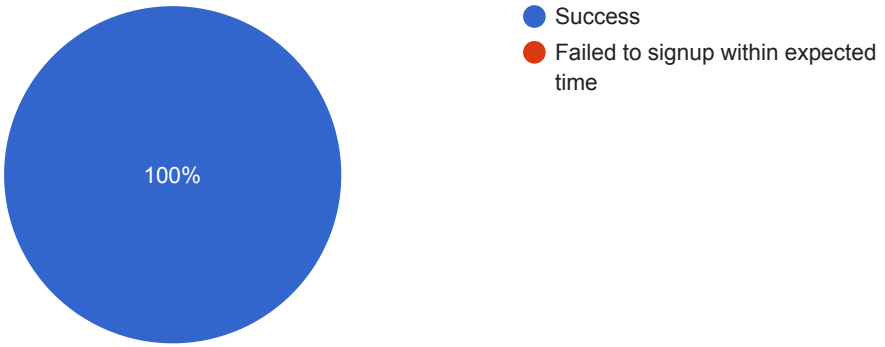
2 responses



#1 Register account - Exp. time: 2 min

#1 Register account (2 min)

2 responses



#1 - If signup errors occured, what did you think of the form input feedback

2 responses

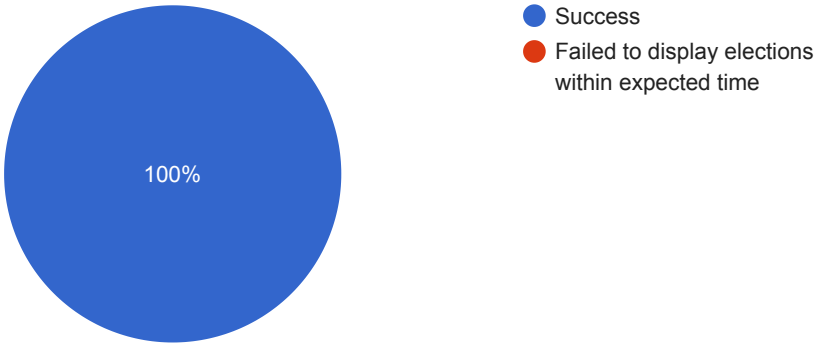
Did get "must be equal..". Thought it was precise and fixed it quickly

Too strict rules on password

#2 Show elections - Exp. time: 20 sec

#2 Show elections (20 sec)

2 responses



#3 Create new election - Exp. time: 4 min

FOR TEST PERSON: Did any error occur? (fill in if something unexpected happend)

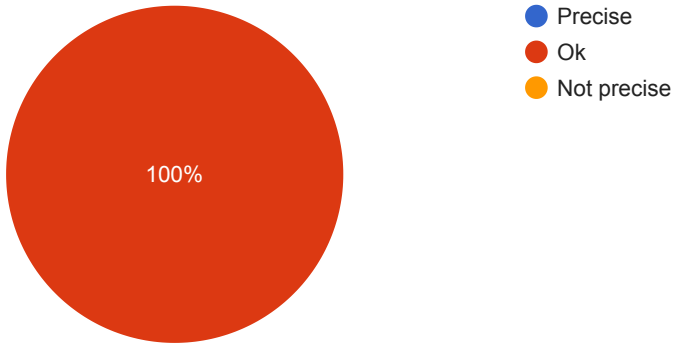
2 responses

Difficult to remember/understand that you need to click OK after dates. Did not understand Eligible voters input. Bad spacing in election form.

none

What did you think of the form input feedback?

2 responses



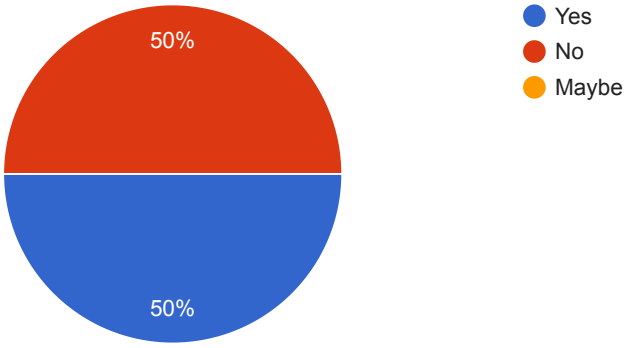
If not precise, why do you could improve?

1 response

Dato skapte problemer

Did you notice whether there was any other way to get to the create election screen?

2 responses



#3.5 Create Ballot - Exp. time: 3 min

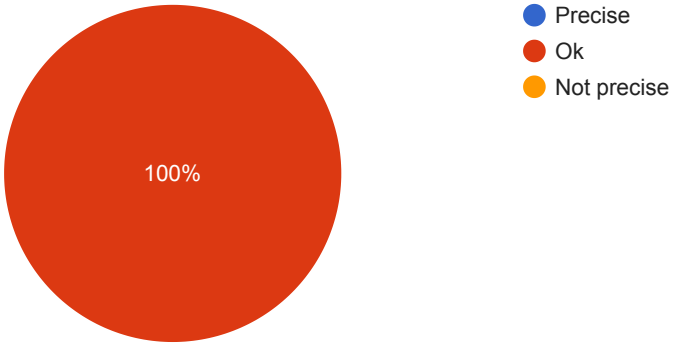
FOR TEST PERSON: Did any error occur? (fill in if something unexpected happend)

1 response

none

What did you think of the form input feedback?

2 responses



If not precise, why do you could improve?

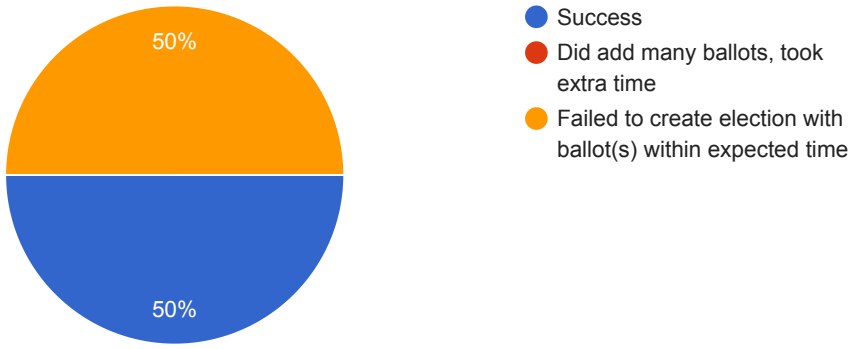
1 response

Ønske om at feltene kom opp automatisk, ikke "Add" per gang. Har mest lyst til å fortsette å skrive... Enter integrasjon var veldig bra

Try to summarize the steps: 3. and 3.5:

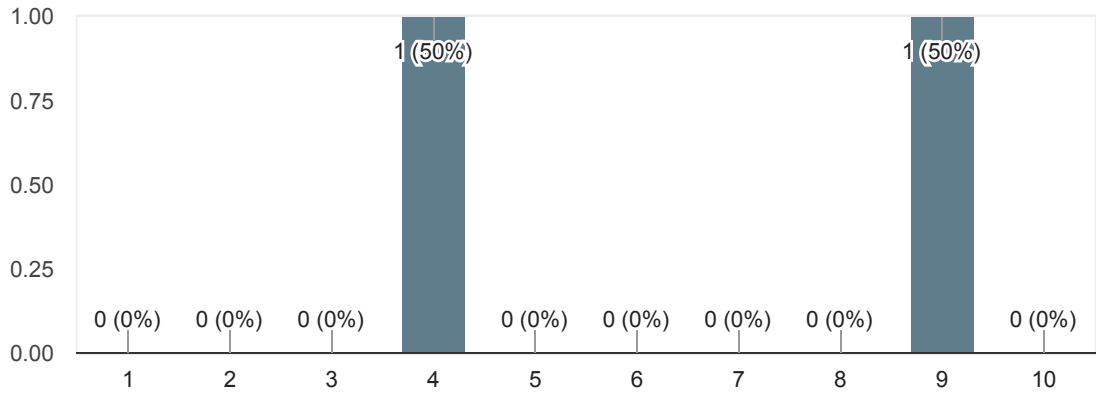
#3 Create election and #3.5 add ballots (7 min)

2 responses



What did you think of the layout and process of creating an election with ballots?

2 responses

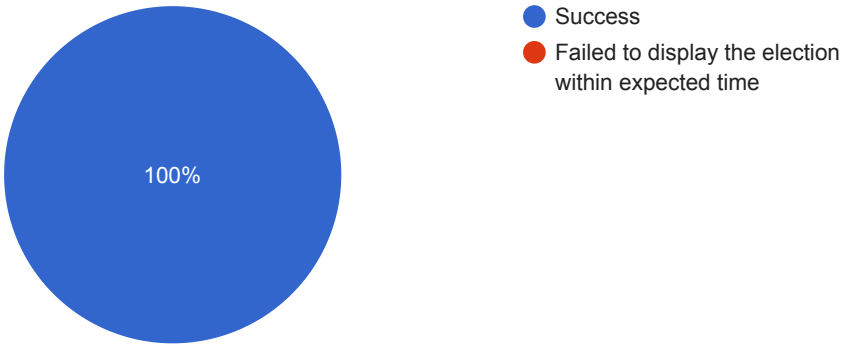


Testing Cont.

#4 Display newly created election - Exp. time: 10 sec

#4 Display the newly created election in its election view (10 sec)

2 responses



FOR TESTER: any remarks?

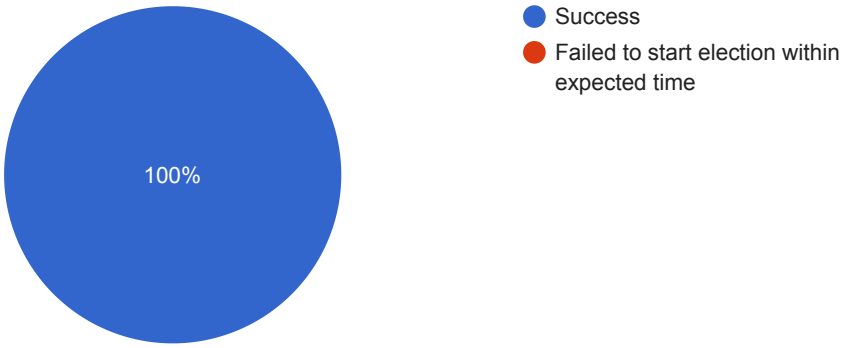
0 responses

No responses yet for this question.

#5 Start the election - Exp. time: 20 sec

#5 Start the election (20 sec)

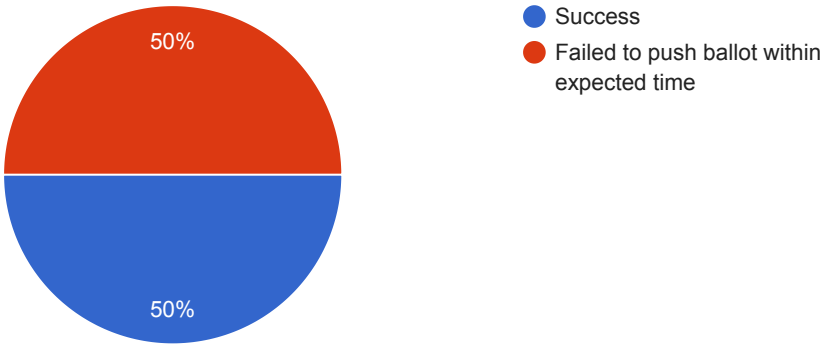
2 responses



#6 Push a ballot - Exp. time: 20 sec

#6 Push ballot (20 sec)

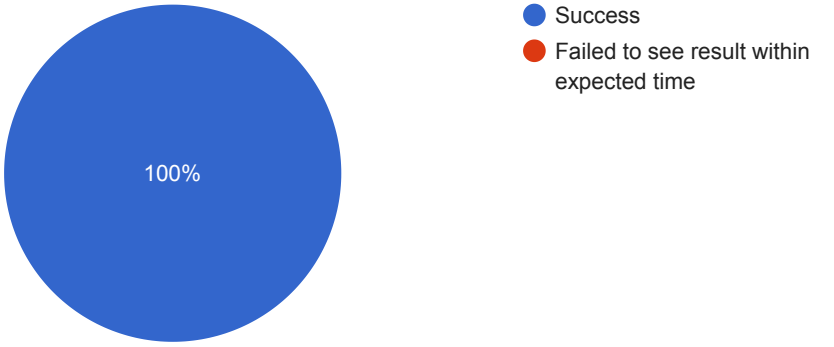
2 responses



#6.5 See result of ballot - Exp. time: 10 sec

#6.5 See result of ballot (10 sec)

2 responses



What happend if he/she did not manage

0 responses

No responses yet for this question.

I noticed you did click on ---. Can you tell me why? Follow up on any interesting behavior you observe during the test to get a better idea of the thought process behind the user's actions.

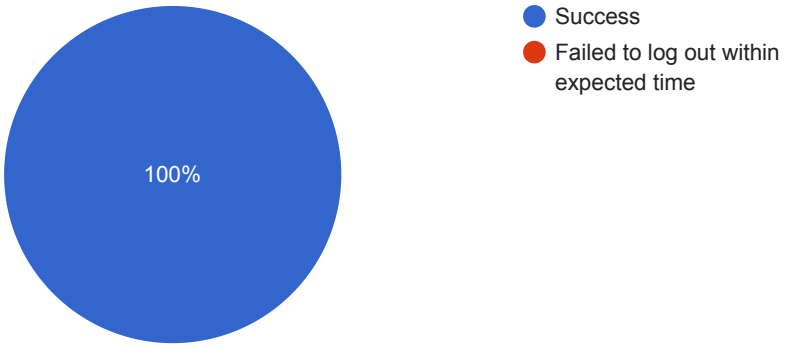
0 responses

No responses yet for this question.

#7 Log out - Exp. time: 10 sec

#7 Log out (10 sec)

2 responses

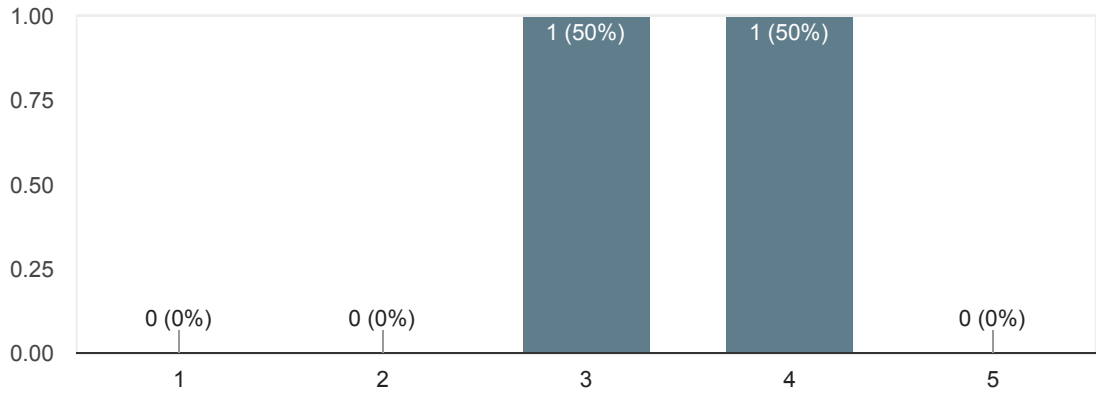


Post test

General questions

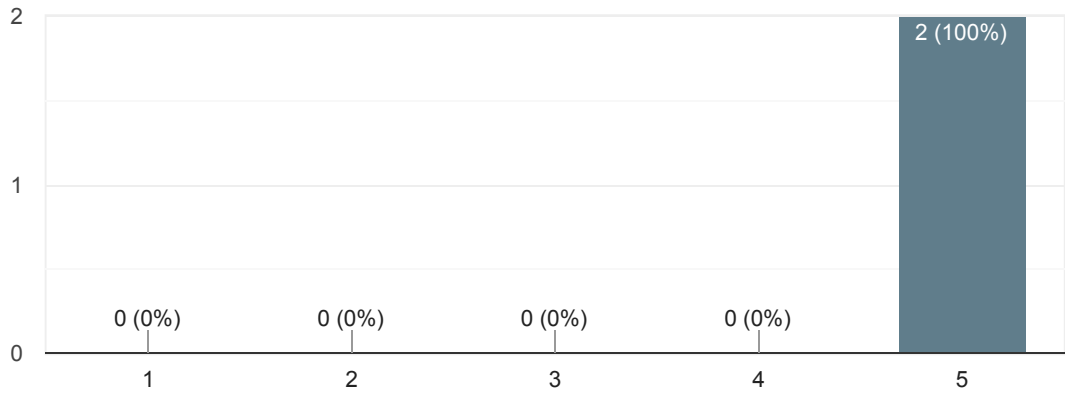
How did you rate the experience of using the website to complete the tasks?

2 responses



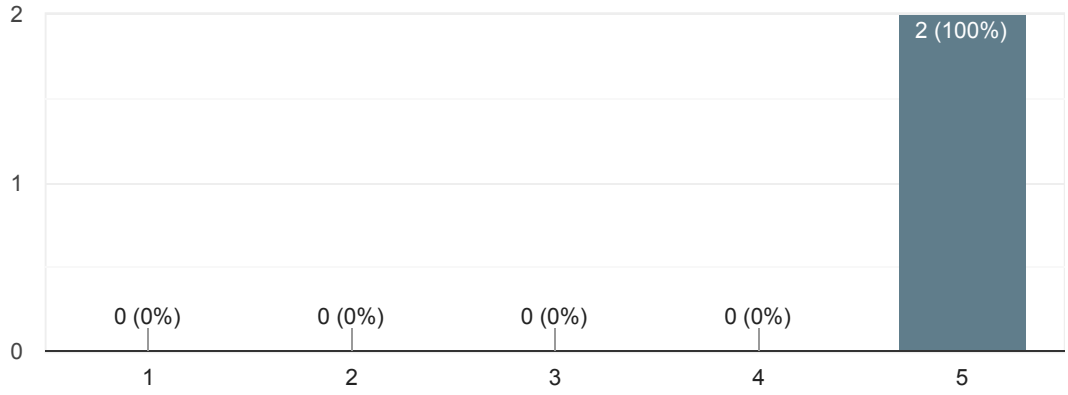
Was it easy to navigate around the site?

2 responses



Was it easy to read the text on the site?

2 responses



Any other comments?

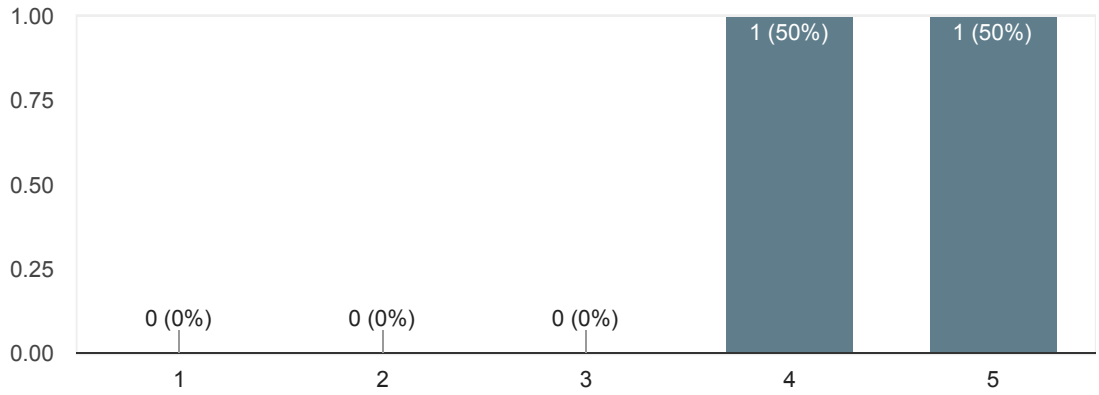
0 responses

No responses yet for this question.

What was your overall impression of ___?

What was your overall impression of creating an election?

2 responses



What could improve?

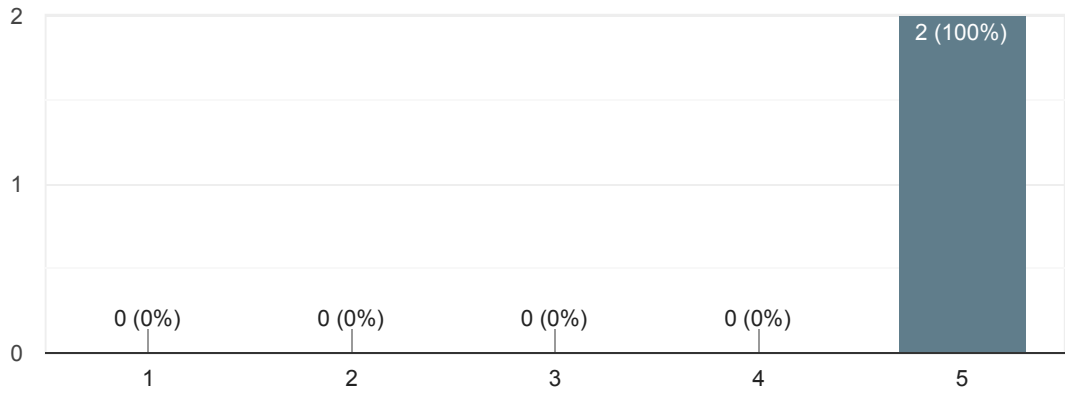
1 response

Under create election:
Det var ikke iøyenfallende at man burde fylle inn ballots. Siden denne var på høyre side og ikke ville gitt noen varsel.

For å skrive inn passord til election: Den lå for tett opp til tabellen

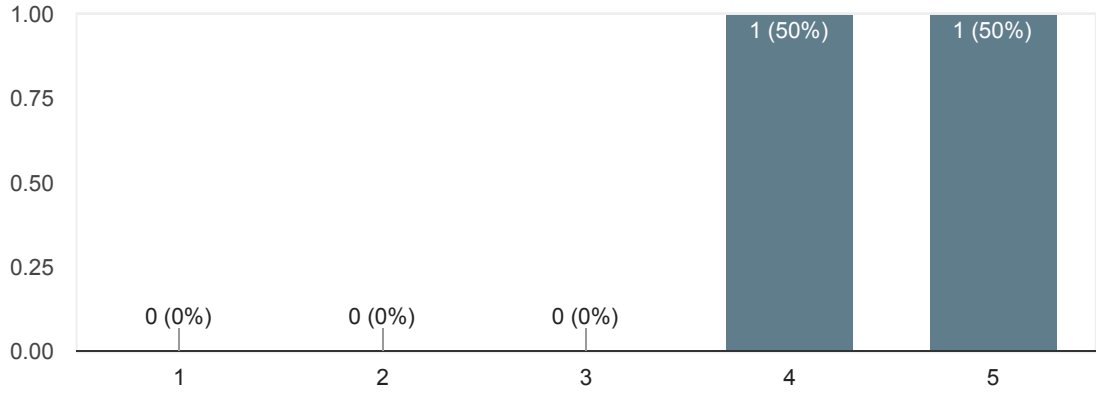
What was your overall impression of registering an account?

2 responses



What was your overall impression of the navigation?

2 responses



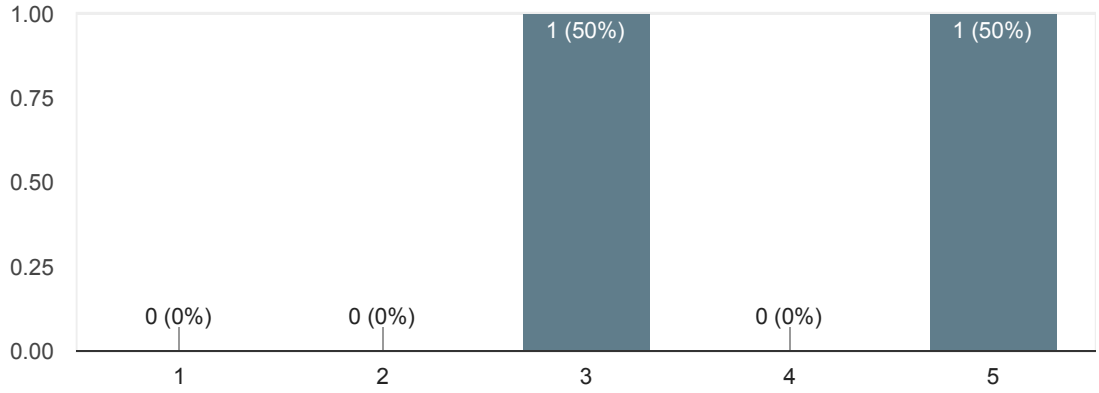
How would you improve it?

0 responses

No responses yet for this question.

What was your overall impression of pushing a ballot?

2 responses



How would improve it?

2 responses

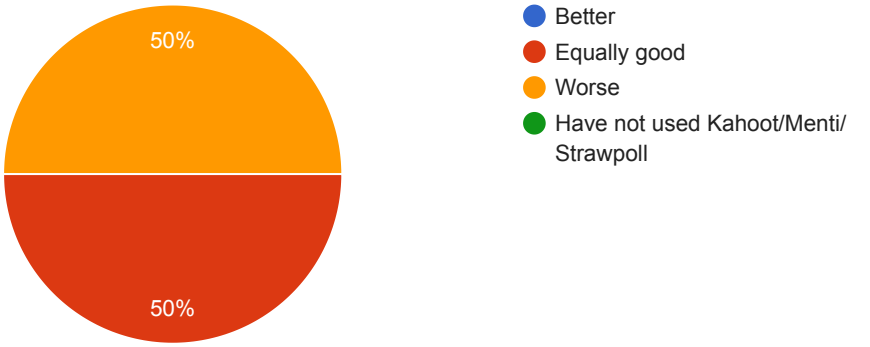
Knowing what "push ballot" does is and does, is not that intuitive. Maybe use a simpler language

"Push"

Other questions

If you have used Kahoot/Menti/Strawpoll to create a quiz: How would you compare Anovote to Kahoot/Menti/Strawpoll?

2 responses



OTHER (Have the tester noted down something unexpected?)

0 responses

No responses yet for this question.

D.2 Large-scale User Test Document

Answers for large scale user test

Who should participate

4 unexperienced

3 experienced users with prior knowledge. These are going to be voters

Screening & pre-test

1. Which age group do you belong to? Students
2. Have you ever arranged a election or quiz of any kind? For ex, by using menti, strawpoll or kahoot? Yes

Test

Plan

#	Task	Description	Expected time	Success
0	Go to page	Should find its way to the landing page	20sec	X
1	Register account	Allows the user to arrange an election	2 min	X
2	Show elections	Display the elections view. It should be empty	20 sec	X
3	Create new election	Create an election to be able to arrange it later	4 min	X Litt uklarhet om election/ballot
3.5	Add ballot(s)	Create a ballot for an election	3 min	X Burde vært lettere å kunne trykke den lilla knappen for å adde, istedenfor kun add
4	Display newly created election	See the newly created election	10 sec	X
5	Start an election	Arrange an election for the newly created election	20 sec	X
6	Push ballot	The election org should push a ballot out	20 sec	X Oversiktlig å se stats kanskje ikke increment. Heller vertikal visning?
6.5	See result	The election org should display the results of the ballot in graph view	10 sec	X
7	Log out	Log out of application	10 sec	X

Arrange an election for:

Styremøte 2021

With description

Valg av representanter til styret 2021

Eligible voters

All in the room

Should include 3 ballots:

- Valg av leder
    - Kandidater
        - Steffen
        - Sander
        - Christoffer
- Valg av nestleder
    - Kandidater
        - Ole Thomas
        - Vilde
- Valg av styremedlem
    - Kandidater
        - Per
        - Pål

bemerkninger

- "Passord vil man ha, fordi man ikke vil at alle skal kunne stemme"
- "Åpningsdato er kjekt å ha"
- Kan ikke bla nedover på election siden når voters tar mye plass
- Election finished or closed on test. Probably because of

Questions to ask AFTER completing a task

- **I noticed you did . Can you tell me why?** Follow up on any interesting behavior you observe during the test to get a better idea of the thought process behind the user's actions.
- **Did you notice whether there was any other way to ___?** You are trying to determine why the user did one thing instead of another.
- **Which of these two approaches/options do you find best? Why?** This is useful if you're trying to determine the more appealing of multiple options.
- **What did you think of the layout of the content?**
- **What did you think of the form input feedback?** (Not precise/Ok/Precise)

Post test

General questions

- How did you find the experience of using the website to complete this task? Happy

  God oversikt over hvor mange som var inne

  Fin oversikt over valget live

- What do you think of the look of the design?

  Enkelt og fint

  Greit å bruke. Lett å starte et valg. Lett å velge ballots

- Was it easy to navigate around the site? Gikk fint
- Was it easy to read the text on the site? Gikk fint
- other comments? Savnet hint og onboarding Savnet oversikt over antall stemmeberettige. I dette tilfellet 6 som stemte som skulle vært 7.

What was your overall impression of [x]?

- Creating an election
  - What could improve?

    Liten tittel for ballot

    Banskelig å forstå seg på noen av feltene: Open, close, passord. Hva er de for?
- Registering
  - Gikk fint
- The navigation
  - How would you improve it? Good
- Pushing a ballot
  - How would you improve it? manglet en onboarding.

If you have used kahoot to create a quiz: **How would you compare Anovote to Kahoot/Menti/Strawpoll?**

- Hvordan gikk det å delta?

Bare fint. Ingen problemer.

- Hvordan gikk det å avgi stemme

Intuitivt. Ingenting man lurte på.

- Design

  Litt "smått" på diger/større skjermer
- Informasjonsflyt mellom avstemningene La ikke merke til noe spesielt.
- Utlogging
  - Blir kasta ut automatisk. Det gikk fint!

Bibliography

[1] API Platform: API Tools & Solutions. URL https://www.postman.com/api-platform/.

[2] Access token. URL https://en.wikipedia.org/wiki/Access_token.

[3] Adobe XD. URL https://en.wikipedia.org/w/index.php?title=Adobe_XD&oldid=1018677082.

[4] Ant Design. URL https://ant.design/.

[5] Atlassian/react-beautiful-dnd. URL https://github.com/atlassian/react-beautiful-dnd.

[6] Auth0/node-jsonwebtoken. URL https://github.com/auth0/node-jsonwebtoken.

[7] Axios. URL https://axios-http.com/.

[8] Ballot. URL https://en.wikipedia.org/w/index.php?title=Ballot&oldid=1018926883.

[9] Bcrypt.js. URL https://github.com/kelektiv/node.bcrypt.js.

[10] Blockchain, . URL https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=1019805422.

[11] Blockchain Definition, . URL https://techterms.com/definition/blockchain.

[12] Bug #1732482 "[snap] doesn't properly save desktop files for "cr..." : Bugs : Chromium-browser package : Ubuntu. URL https://bugs.launchpad.net/bugs/1732482.

[13] Code formatting. URL https://torquemag.io/2019/07/code-formatting-guide/.

[14] Create-react-app. URL https://github.com/facebook/create-react-app.

[15] Cross-Origin Resource Sharing (CORS) - HTTP | MDN, . URL https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[16] Cross-site scripting - MDN Web Docs Glossary: Definitions of Web-related terms | MDN, . URL https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting.

[17] Cryptocurrency, . URL https://en.wikipedia.org/w/index.php?title=Cryptocurrency&oldid=1020227186.

[18] Cryptography - Electronic Voting, . URL https://crypto.stanford.edu/pbc/notes/crypto/voting.html.

[19] Cryptology ePrint Archive, . URL https://eprint.iacr.org/complete/.

[20] DataGrip: The Cross-Platform IDE for Databases & SQL by JetBrains, . URL https://www.jetbrains.com/datagrip/.

[21] Date-fns, . URL https://date-fns.org/.

[22] Definition of BALLOT. URL https://www.merriam-webster.com/dictionary/ballot.

[23] Design principles. URL https://folk.ntnu.no/baldurk/skolearbeid/MMI/Forelesninger%20MMI/30-Designprinsipper.pdf.

[24] Discord (software). URL https://en.wikipedia.org/w/index.php?title=Discord_(software)&oldid=1021245834.

[25] Don normans design principles. URL http://www.csun.edu/science/courses/671/bibliography/preece.html.

[26] ESLint. URL https://eslint.org/.

[27] Election, . URL https://en.wikipedia.org/w/index.php?title=Election&oldid=1016660308.

[28] Election | History, Polls, Results, Date, & Facts, . URL https://www.britannica.com/topic/election-political-science.

[29] Electoral system, . URL https://en.wikipedia.org/w/index.php?title=Electoral_system&oldid=1017267926.

[30] Electoral systems (BP-334E), . URL http://publications.gc.ca/Collection-R/LoPBdP/BP/bp334-e.htm#INTRODUCTIONtxt.

[31] Express - Node.js web application framework. URL https://expressjs.com/.

[32] Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger - Lovdata. URL https://lovdata.no/dokument/SF/forskrift/2013-06-21-732?q=forskrift%20om%20universell%20utforming.

[33] Git, . URL https://git-scm.com/.

[34] Git, . URL https://en.wikipedia.org/w/index.php?title=Git&oldid=1021092067.

[35] GitHub Actions, . URL https://github.com/features/actions.

[36] Hello World · GitHub Guides, . URL https://guides.github.com/activities/hello-world/.

[37] Helmetjs/helmet, . URL https://github.com/helmetjs/helmet.

[38] Hypertext Transfer Protocol – HTTP/1.1, . URL https://www.w3.org/Protocols/rfc2616/rfc2616.html.

[39] Hypertext Transfer Protocol, . URL https://en.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&oldid=1021103766.

[40] Integrated development environment, . URL https://en.wikipedia.org/wiki/Integrated_development_environment.

[41] Intelligent code completion, . URL https://en.wikipedia.org/wiki/Intelligent_code_completion.

[42] Intro to Ethereum, . URL https://ethereum.org.

[43] Introducing GitFlow, . URL https://datasift.github.io/gitflow/IntroducingGitFlow.html.

[44] Introduction to dapps, . URL https://ethereum.org.

[45] Introduction to Test Driven Development (TDD), . URL http://agiledata.org/essays/tdd.html.

[46] JavaScript. URL https://en.wikipedia.org/wiki/JavaScript.

[47] Jest (JavaScript framework). URL https://en.wikipedia.org/w/index.php?title=Jest_(JavaScript_framework)&oldid=1002056817.

[48] Linter. URL https://en.wikipedia.org/wiki/Lint_(software).

[49] Minimum viable product. URL https://en.wikipedia.org/w/index.php?title=Minimum_viable_product&oldid=1023320733.

[50] Module pattern. URL https://addyosmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript.

[51] Nginx. URL https://nginx.org/en/.

[52] ORM. URL https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping.

[53] Object–relational mapping. URL https://en.wikipedia.org/w/index.php?title=Object%E2%80%93relational_mapping&oldid=1021719270.

[54] Observer. URL https://refactoring.guru/design-patterns/observer.

[55] Open ballot system. URL https://en.wikipedia.org/w/index.php?title=Open_ballot_system&oldid=1001325053.

[56] Papa Parse - Powerful CSV Parser for JavaScript. URL https://www.papaparse.com/.

[57] Port from deno to node. URL https://github.com/anovote/org/blob/main/decisions/port-from-deno-to-node.md.

[58] PostgreSQL. URL https://en.wikipedia.org/w/index.php?title=PostgreSQL&oldid=1023021237.

[59] Prettier. URL https://prettier.io/.

[60] Progressive web apps (PWAs) | MDN, . URL https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

[61] Prototype Testing / Wireframe Testing|Professionalqa.com, . URL https://www.professionalqa.com/wireframe-testing-or-prototype-testing.

[62] REST Principles and Architectural Constraints. URL https://restfulapi.net/rest-architectural-constraints/.

[63] RTE (Runtime Environment) Definition. URL https://techterms.com/definition/rte.

[64] React, . URL https://reactjs.org/.

[65] React i18next, . URL https://react.i18next.com/.

[66] React Router: Declarative Routing for React, . URL https://reacttraining.com/react-router.

[67] React Testing Library | Testing Library, . URL https://testing-library.com/docs/react-testing-library/intro.

[68] Retningslinjer for tilgjengelig webinnhold (WCAG) 2.0. URL https://www.w3.org/Translations/WCAG20-no/.

[69] SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. URL https://developer.mozilla.org/en-US/docs/Glossary/SPA.

[70] SQL, . URL https://en.wikipedia.org/wiki/SQL.

[71] SQL Injection Prevention - OWASP Cheat Sheet Series, . URL https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

[72] Scrum. URL https://felixgrayson.wordpress.com/tag/scrum/.

[73] Single-page application, . URL https://en.wikipedia.org/w/index.php?title=Single-page_application&oldid=1017843157.

[74] Singleton, . URL https://refactoring.guru/design-patterns/singleton.

[75] Source-code editor. URL https://en.wikipedia.org/wiki/Source-code_editor.

[76] Transport Layer Security. URL https://en.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=1021012606.

[77] TypeScript, . URL https://www.typescriptlang.org/.

[78] Typestack/class-transformer, . URL https://github.com/typestack/class-transformer.

[79] Typestack/class-validator, . URL https://github.com/typestack/class-validator.

[80] Universal design. URL https://en.wikipedia.org/w/index.php?title=Universal_design&oldid=1017733026.

[81] User interface design. URL https://en.wikipedia.org/w/index.php?title=User_interface_design&oldid=1020653595.

[82] Version control. URL https://en.wikipedia.org/w/index.php?title=Version_control&oldid=1020382212.

[83] Virtual machine. URL https://en.wikipedia.org/wiki/Virtual_machine.

[84] Visual Studio Code. URL https://en.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=1020885576.

[85] Voting, . URL https://en.wikipedia.org/w/index.php?title=Voting&oldid=1020953943.

[86] Voting machine, . URL https://en.wikipedia.org/w/index.php?title=Voting_machine&oldid=1016435078.

[87] Web application, . URL https://en.wikipedia.org/w/index.php?title=Web_application&oldid=1018709214.

[88] WebSocket, . URL https://en.wikipedia.org/wiki/WebSocket.

[89] Website wireframe, . URL https://en.wikipedia.org/w/index.php?title=Website_wireframe&oldid=1019435291.

[90] What are Scrum Artifacts?, . URL https://www.visual-paradigm.com/scrum/what-are-scrum-artifacts/.

[91] What is Continuous Delivery? – Amazon Web Services, . URL https://aws.amazon.com/devops/continuous-delivery/.

[92] What is Continuous Integration? – Amazon Web Services, . URL https://aws.amazon.com/devops/continuous-integration/.

[93] What is Mobile First?, . URL https://smartbear.com/learn/performance-monitoring/what-is-mobile-first/.

[94] What is REST, . URL https://restfulapi.net/.

[95] What is a Scrum Master?, . URL https://www.agilealliance.org/glossary/scrum-master/.

[96] What is Unified Modeling Language, . URL https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language.

[97] Windows Subsystem for Linux. URL https://en.wikipedia.org/w/index.php?title=Windows_Subsystem_for_Linux&oldid=1022616864.

[98] Adobe Illustrator. URL https://en.wikipedia.org/wiki/Adobe_Illustrator.

[99] Integration Testing: What is, Types, Top Down & Bottom Up Example. URL https://www.guru99.com/integration-testing.html.

[100] Localization. URL https://en.wikipedia.org/wiki/Language_localisation.

[101] Overleaf. URL https://github.com/overleaf/overleaf.

[102] Damien Arrachequesne. Socket.IO. URL https://socket.io/index.html.

[103] Atlassian. Jira. URL https://www.atlassian.com/software/jira.

[104] L. M. Bach, B. Mihaljevic, and M. Zagar. Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1545–1550. doi: 10.23919/MIPRO.2018.8400278.

[105] Kumar Ch and rakant. REST vs WebSockets | Baeldung. URL https://www.baeldung.com/rest-vs-websockets.

[106] Thomas M. Connolly and Carolyn E. Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Always Learning. Pearson, 6. ed., global ed edition. ISBN 978-1-292-06118-4 978-0-13-294326-0.

[107] Torgeir Daler, Roar Gulbrandse, Tore Audun Høie, and Torbjørn Sjølstad. *Håndbok i datasikkerhet informasjonsteknologi og risikostyring*. Tapir akademisk. ISBN 978-82-519-2538-9.

[108] Vincent Driessen. A successful Git branching model. URL http://nvie.com/posts/a-successful-git-branching-model/.

[109] Sfirnaciuc Emilia, Vasilescu Miruna-Elena, and Simion Emil. E-voting protocols in context of COVID19. URL http://eprint.iacr.org/2021/027.

[110] World Leaders in Research-Based User Experience. From Research Goals to Usability-Testing Scenarios: A 7-Step Method, . URL https://www.nngroup.com/articles/ux-research-goals-to-scenarios/.

[111] World Leaders in Research-Based User Experience. Usability Testing 101, . URL https://www.nngroup.com/articles/usability-testing-101/.

[112] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 3rd ed edition. ISBN 978-0-321-19368-1.

[113] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, editors. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley. ISBN 978-0-201-63361-0.

[114] Thomas Haines and Peter Roenne. New Standards for E-Voting Systems: Reflections on Source Code Examinations. URL http://eprint.iacr.org/2021/391.

[115] Thomas Haines, Rajeev Gore, and Bhavesh Sharma. Did you mix me? Formally Verifying Verifiable Mix Nets in Electronic Voting. URL http://eprint.iacr.org/2020/1114.

[116] James Hall. MrRio/jsPDF. URL https://github.com/MrRio/jsPDF.

[117] Chandler Harris. Agile scrum artifacts. URL https://www.atlassian.com/agile/scrum/artifacts.

[118] Philip Hutchison. PDFObject: A JavaScript utility for embedding PDFs. URL https://pdfobject.com/.

[119] Gergely Kalman. 10 Web Security Vulnerabilities You Can Prevent. URL https://www.toptal.com/security/10-most-common-web-security-vulnerabilities.

[120] Kieran Kilbride-Singh. WebSockets vs Long Polling. URL https://ghost.ably.com/blog/websockets-vs-long-polling/.

[121] kingthorin. SQL Injection. URL https://owasp.org/www-community/attacks/SQL_Injection.

[122] James Kurose and Keith Ross. *Computer Networking*. Pearson Education Limited. ISBN 978-1-292-15360-5. URL https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=5573712.

[123] Anders Smedstuen Lund and Martin Strand. Decryption phase in Norwegian electronic voting. URL http://eprint.iacr.org/2016/1002.

[124] Warren Lynch. The Brief of History of Scrum. URL https://warren2lynch.medium.com/the-brief-of-history-of-scrum-15efb73b4701.

[125] Rachaelle Lynn. History of agile | planview leankit. URL https://www.planview.com/no/resources/guide/agile-methodologies-a-beginners-guide/history-of-agile/.

[126] Slava Moskalenko. What Scrum Says About Estimates. URL https://www.scrum.org/resources/blog/what-scrum-says-about-estimates.

[127] Bryce Neal. Prettymuchbryce/http-status-codes. URL https://github.com/prettymuchbryce/http-status-codes.

[128] Jennifer Niederst Robbins. *Learning Web Design: A Beginner's Guide to HTML, CSS, Javascript, and Web Graphics*. O'Reilly, fifth edition edition. ISBN 978-1-4919-6020-2.

[129] Steven Telio of Build on Purpose 02/23/2016. Customer Feedback Best Practices: Usability Testing. URL https://community.uservoice.com/blog/usability-testing-best-practices/.

[130] Aris Papadopoulos. Should Developers Use Third-Party Libraries? URL https://www.scalablepath.com/blog/third-party-libraries/.

[131] Tom Preston-Werner. Semantic Versioning 2.0.0. URL https://semver.org/.

[132] Dan Radigan. What are story points and how do you estimate them? URL https://www.atlassian.com/agile/project-management/estimation.

[133] Ken Schwaber and Jeff Sutherland. Scrum Guide. URL https://scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100.

[134] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley, 9. ed., internat. student version edition. ISBN 978-1-118-09375-7.

[135] Viktoria Stray, Nils Brede Moe, and Dag I.K. Sjoberg. Daily Stand-Up Meetings: Start Breaking the Rules. 37(3):70–77. ISSN 0740-7459, 1937-4194. doi: 10.1109/MS.2018.2875988. URL https://ieeexplore.ieee.org/document/8501962/.

[136] Peter Varhol. The complete history of agile software development. URL https://techbeacon.com/app-dev-testing/agility-beyond-history-legacy-agile-development.

[137] Niklas von Hertzen. Html2canvas - Screenshots with JavaScript. URL https://html2canvas.hertzen.com/.

[138] Michael E. Whitman and Herbert J. Mattord. *Principles of Information Security*. Cengage Learning, sixth edition edition. ISBN 978-1-337-10206-3.