System documentation CutWear v1.0 - Group 29

Davidsen, Mats Elias Lund, Kristian Nybakken Wiig, Knut

20th of May 2019

Innhold

1 Introduction

Last updated 20th May 2019
This document is an attachement to a bachelor's thesis at NTNU IDI, and functions as system documentation for the CutWear-system (version 1), which consists of a client and a server (API). The purpose of this document is to document all demands and dependencies to be able to install and maintain the system. The document contains all relevant information for maintaining, and further developing the system, by documenting how the system works and how to use it. For documentation of why certain choices, regarding technology, were made, please see the chapter 4 in the bachelor's thesis. This document is written for people with some knowledge of running applications and has some knowledge of programming with databases. If a tutorial or article has been used for developing some of the solutions, a reference is provided. One does not need knowledge of Node.js or npm to run the application for production, but some knowledge of Docker and Docker-compose is needed to start or stop the application.

All diagrams and models were made using draw.io.

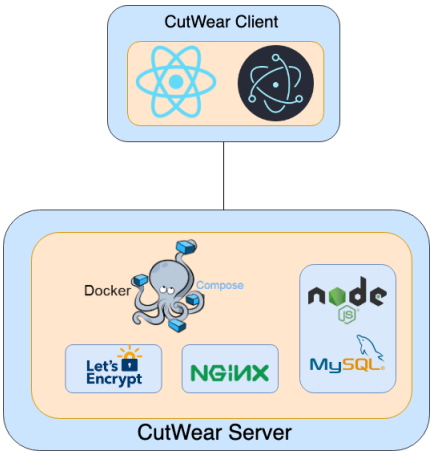
The client and server is separated in two private Github repositories:
Server: Github - cutWearServer
Client: Github - CutWearClient

Each repository contains a README.md file, where the latest information will be available. If there is a reference to one of the README files in this document, the files are located on the repository root, see 3.
For access to repositories, please refer to the customer in the thesis.
For future reference, any mentions of 'server' or 'client' in this document, will be referencing CutWearServer or CutWearClient, respectively.

The server in which the API is hosted (at the time this documentation is being written) and tested on, is a virtual machine running Ubuntu 18.04, 127 GB storage, 2 GB RAM and dual-core CPU and is available at https://cutware.ibm.ntnu.no (yes, there is a typo). Se chapter 5 for more information about how to use the API.

2 Architecture and Design Patterns

2.1 Technology Stack

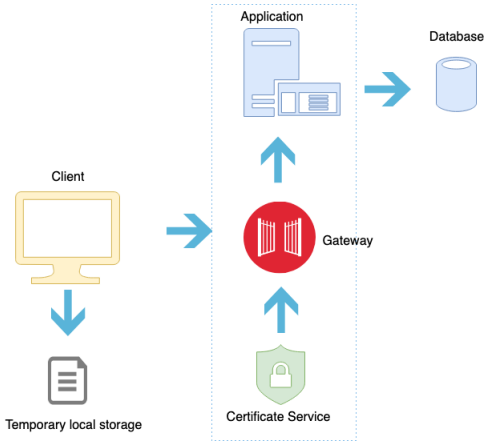


Figur 1: Overview of the technology stack of CutWear

Figure 1 shows a simple overview of the CutWear system. The client uses the JavaScript framework

ReactJS v16.8.1 and Electron v4.1.0 as platform. The server stack is bit more complicated, and uses Node.js v10.15.1 and Express v4.16.4. CutWear uses Docker-compose to build, start and stop server using multiple Dockerimages. The Docker images (represented by blue rectangles inside the golden rectangle) consists of one Docker image to obtain and renew Let's Encrypt (TLS) certificates, the second Docker image (nginx) works as a load balancer/proxy (in case you want to run more instances of CutWear) and functions as the gateway to the CutWear API, by receiving a (HTTPS) request and passing it through to CutWear. [1] The third Docker image is the server application, which uses NodeJS with Express framework and connects to a external MySQL-database. [2]

2.2 Architecture



Figur 2: Architecture of CutWear. (not shown in image, the response flow)

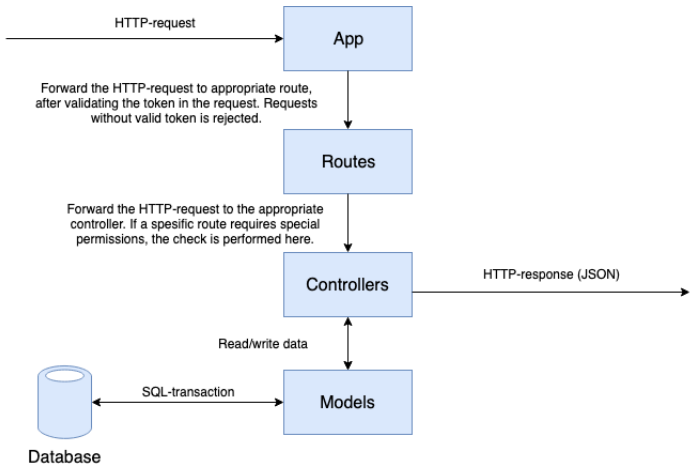
Figure 2 shows the architecture of CutWear, which is a REST-architecture. See 5 for REST-documentation. The client sends a HTTPS reguests to the gateway, and the gateway processes the HTTPS the request and forwards it to the application server as a HTTP-request, where the request will be processed. If the request requires data, the application server makes a request to the database.
The certificate service provides the gateway with valid certificates from Let's Encrypt. Upon renewal of certificates the certificate service restarts the gateway. [1]
If the client is offline, it will store changes in a temporary local file until reconnected to internet, where it will try to re-upload.

2.3 Model Route Controller

CutWear server uses a Model View Controller(MVC) design pattern, but since the View is only accessible through the desktop application, it has transformed into a Model Route Controller (MRC) pattern, where the Route has been included for easier understanding. Figure 3 shows the how a request is handled by the server and how different components interact with eachother. The response is always in JSON format. [3]

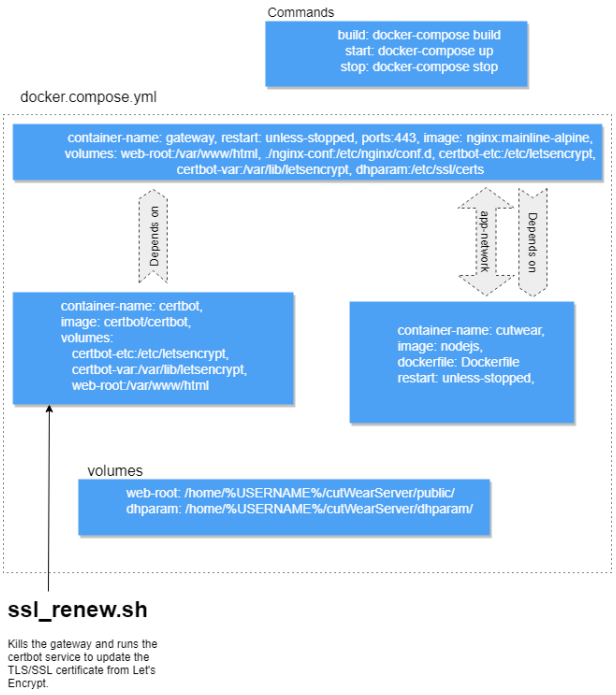
2.4 Docker-Compose

Figure 4 shows the Docker-compose setup for CutWear. The gateway is running the latest nginx-image from Dockerhub, and handles all incoming requests and passes them to cutwear. The gateway depends on cutwear running for performings its tasks. The cutwear service uses the Dockerfile in the server repository. Gateway and Cutwear uses a app-network, which is a virtual network, to communicate between each other. Cutwear and gateway can only be stopped by using the 'stop' command, since 'restart' is configured to 'unless-stopped', meaning the services will be restarted only if you use the stop command. Certbot runs the certbot image from Let's Encrypt's Dockerhub. Certbot sends a request to Let's Encrypt to get or renew



Figur 3: Design pattern of the CutWear server application. Shows the flow of request and responses.

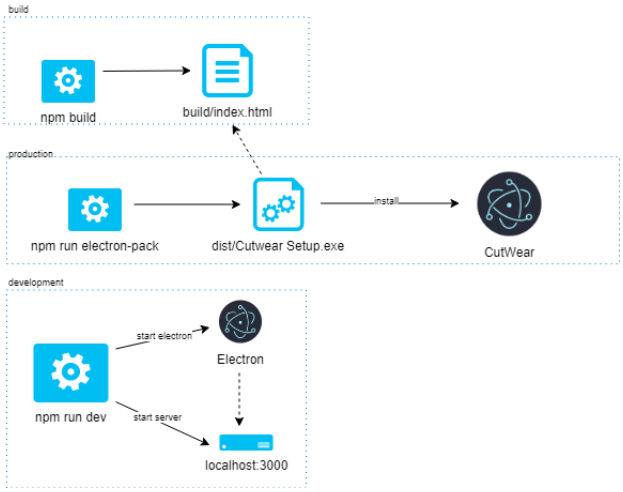
TLS/SSL certificates. Certbot should exit upon finishing its tasks. The ssl_renew.sh is a executable and kills the gateway and starts the certbot image and should be run in intervals using crontab. See chapter 7 for more information. [2] [1]



Figur 4: Docker-Compose setup for CutWear

2.5 Client

The client for this project was bootstrapped with Create React App, this gave the development team a starting point for developing the frontend. This is also explained in the README file of the project. If you do not have access to the README file, the documentation for Create React App can be found by following this link: **Create React App Documentation**

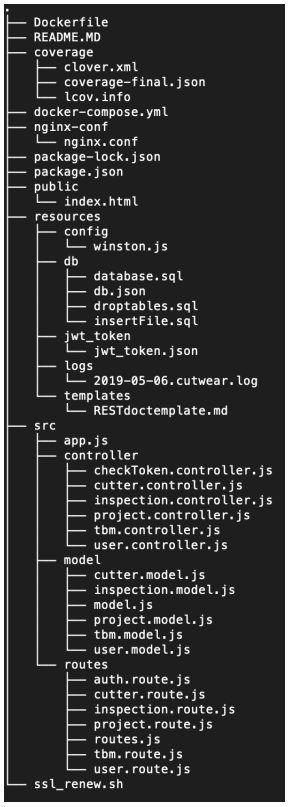


Figur 5: Overview of client in different environments

Figure 5 shows what files are being used by the client when running in different environments. To generate the build files, the node package manager (npm) calls the build script in the package.json, and then creates the build files which is located in the build directory. When packaging the application for production (NODE_ENV=production), using the dependency (electron-builder) and the command 'npm run electron-pack', it uses the build files to package and generate a installer for the application. [4] When running in a development environment (NODE_ENV=development), using 'npm run dev', it will start a local server, and start electron which points to localhost:3000 and opens the developer tools. When running electron with localhost instead of build files, you enable hot-reloading which means the application updates upon changes to code.

3 Projectstructure

3.1 Server



Figur 6: File structure of the server (generated using tree command)

Figure 6 shows the file structure of the server. The source code is located under src/ folder, with app.js being the entry file. The main route file is route.js which then passes the request to the correct route. Source files are usually build up with *.role.js. The source code is organized in MCR, as shown in 2.3.
The resources directory contains configuration files,templates and logs. The Dockerfile, docker.compose.yml and ssl_renew.sh contains the scripts for building and running the server. See 2.4 for more information.
Coverage folder containts result of tests. See 7.4.2 for more details. The public folder only contains index.html, which works as a API homepage. Package.json and package-lock.json contains information about the server, such as dependencies, versions, name, and npm scripts.
Not shown in 6 is the tests (7.4.2) folder, out, pack_with_docker.sh (7.3) folder and node_modules.

3.1.1 npm scripts

npm install

Installs all the dependencies. Needs to be run first time and upon changes to dependencies.

npm run test

Script for running the tests in the server. See 7.4.2 for more information.

npm start

Starts the server outside of Docker. The script points the './src/app.js' file and starts it using 'node'. Usually used in development environments.

npm run doc

Generates the code documentation. See 7.3 for more details.

3.1.2 Docker commands

Commands might require 'sudo'.

docker build -t cutwear node .

Builds the image described in Dockerfile. The '-t' is added to give a name to the containter, in this case 'cutwear'. The '.' describes the context of the build.

docker run -p 80:5000 -d cutwear

Starts the image 'cutwear'. '-p' forwards port 80 to port 5000 in the image. Use '-d' to run image in the background. Port is hardcoded in app.js in server, so if you wish to use another port update the command and port in app.js or set the PORT environment variable. The application is now available at http://localhost:5000

docker stop cutwear

Stops the image cutwear

docker stop

Stops all images running

docker ps

List of all running images.

3.1.3 Docker-Compose commands

docker-compose build

Builds all the images described in the docker.compose.yml file.

docker-compose build cutwear

Builds the image named cutwear.

docker-compose up -d

Start all the images. '-d' for running in the background.

docker-compose stop

Stops all the images. Only valid way to stop gateway and cutwear image.

docker-compose stop cutwear

Stops one image. In this case, 'cutwear' is the image to stop.

docker-compose up -d –force-recreate –no-deps nodejs

If you have stopped one image, use this command to start the image you stopped without affecting other images running.

docker-compose logs

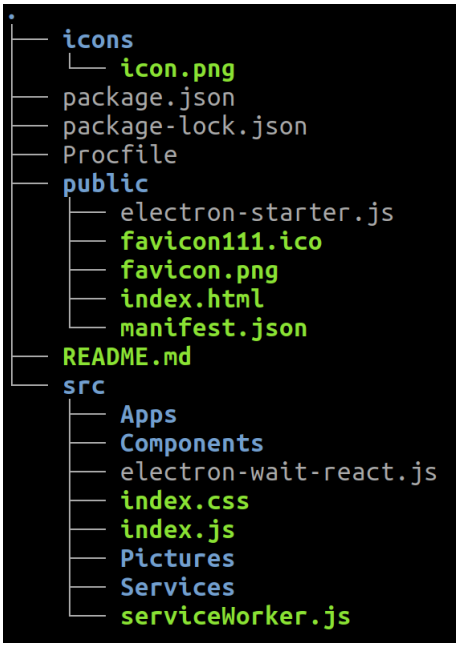
Gets all the logs from the running images.

docker-compose logs cutwear

Gets the logs from cutwear.

3.2 Client

The Client has the following simplified structure, shown in figure 7. In addition to this there is also a node modules folder, but there is not relevant to go into detail about this folder.

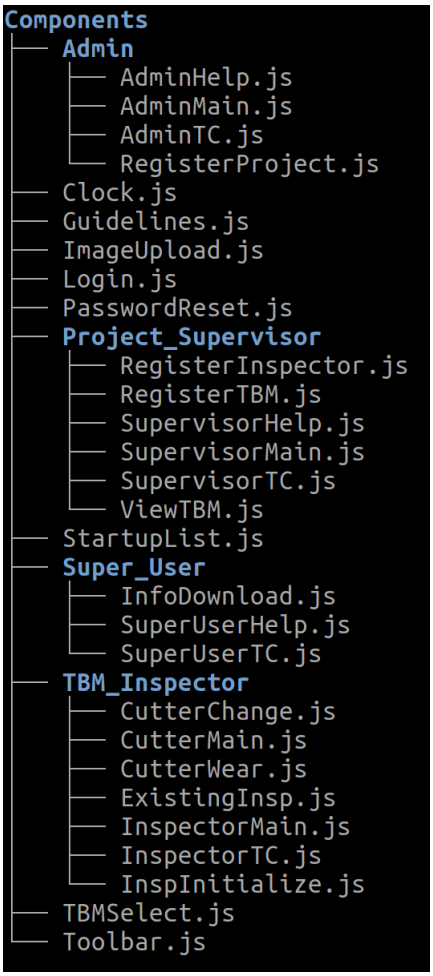


Figur 7: File structure of the client (generated using tree command)

The source code is located under src/ folder, with index.js being the entry file and index.css styling most of the app. Each user group has a main file located in the Apps/ folder that handles the state and saves the fundamental data throughout the session.

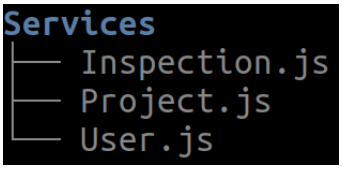
Figur 8: File structure of the Apps/ folder (generated using tree command)

Since the client is written in React the application is composed of Components, these components is located in the Components/ folder. In the Components/ folder the components belonging to one specific user group is located in a folder with the same name as the user group.



Figur 9: File structure of the Components/ folder (generated using tree command)

The last folder that contains source code is the Services/ folder, this contains files that handle the communication with the server.



Figur 10: File structure of the Services/ folder (generated using tree command)

3.2.1 npm scripts

npm install

Installs all the dependencies. Needs to be run first time and upon changes to dependencies.

npm run dev

Starts the client in development mode. If running app on Windows, remove 'BROWSER=none' from package.json. NODE_ENV must be set to 'development'.

npm run electron-pack

Builds and packages the application to .exe. NODE_ENV must be set to 'production'. Could take some time and consume some memory.

npm test

Calls the test script from create-react-app.

npm run electron

Starts electron without running server.

npm run preelectron-pack

Calls the build script before electron-pack.

npm run doc

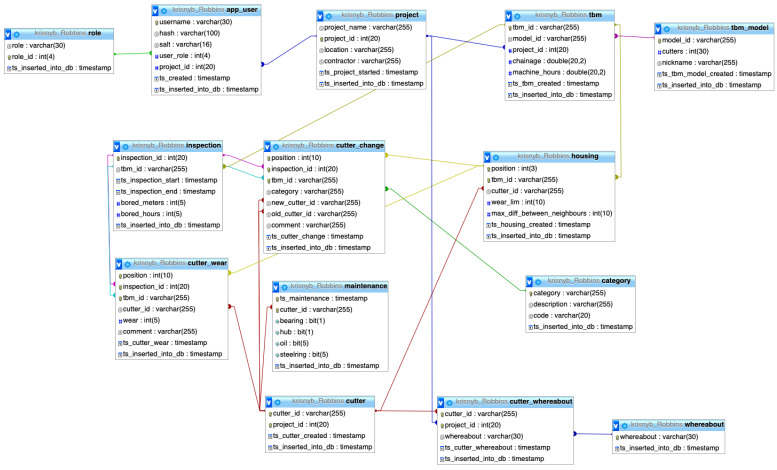
Generates the code documentation. See 7.3 for more details.

3.2.2 Docker scripts

pack_with_docker.sh

Starts up a Dockerimage and generates a Installer for platforms described in package.json. Results are stored in dist folder.

4 Database Model



Figur 11: Database model of CutWear, foreign keys are included

5 REST Services

This section has been written alongside development in markdown and is meant for the README in the repository. The file is therefore converted from markdown to pdf, which explains the inconsistent format of this section.

AUTH means that an valid Bearer token must be provided in the request
Tokens can be request by sending a login request
Authorization: Bearer TOKEN
If auth is required, and an invalid token is provided, you will receive this message:
Code: 400 BAD REQUEST

```
{
    "status":false,
    "message": "Invalid Token"
}
```

Logging in

POST for logging in and receiving a token
POST /api/v1/auth/login
Auth required: NO
Permissions required: User must be created
Data constraints:
The body the http-request must contain username and password

```
{
    "username": "String of valid username",
    "password": "String of valid password"
}
```

Success Responses

Condition: User is authenticated and has received a jwt token.
Code: 200 OK
Content Example:
The response from server

```
{
    "success": true,
    "message": "Authorization successful",
    "token": "eyJhbGciOiJIUzI1NiIsInwc3244fVCJ9.eyJ1c2VybmFtZSI6IkJpbGx5QXJuZSIs2edWDlhdCIwrwrwMzE2Mzr2r32ZXhwIjoxNTUzMjA2ODc0fQ.VEYWXtrwer3EFmwtR2sxefDej452-j5zPqEbXzalAyQ
}
```

To use the token, put it in the Authorization or x-access-token header as a Bearer token. Example.
Authorization: Bearer token
Remember the whitespace between Bearer and the token.

Error Responses

Condition: Username or password is missing from request body
Code:422 UNPROCESSABLE ENTITY
Content Example:
The response from server

```
{
     "success": false,
     "message": "missing username or password"
   }
```

OR

Condition: Username or password is incorrect and user is not authenticated
Code: 400 BAD REQUEST
Content Example:
The response from server

```
{
     "success": false,
     "message": "Authorization unsuccessful"
   }
```

OR

Condition: Internal server error
Code: 500 Internal Server Error
Content Example:
The response from server

```
{
     "success": false,
     "message": "Error in authenticating user"
   }
```


Registering Users

POST for registering new users.
POST api/v1/users/register
URL parameters: NONE

Auth required: YES
Permissions required: Administrator or a project supervisor in the project which the new user is to added.
Data Constraints:

```
{
    "username":"yourusernamehers",
    "password": "yourpasswordhere",
    "user_role": "role of user (integer)",
    "project_id":"id of project(integer)"
}
```

Success Responses

Condition: User is created.
Code: `200 OK`
Content Example:

```
{
    "status":true,
    "message": "User created successfully"
}
```

Error Response

Condition: The user trying to register a new user does not have permission to create users.
Code:`400 BAD REQUEST`
Content Example:

```
{
    "status": false,
    "message": "Permission Denied"
}
```

OR

Condition: Missing or incorrect parameters.
Code: `422 Unprocessable Entity`
Content Example:

```
{
    "status": false,
    "message": "Missing or incorrect parameters"
}
```

OR

Condition: Something went wrong in registering user.
Code: Ex: `500 INTERNAL SERVER ERROR`
Content Example: What the response content is to look like. Example:

```
{
    "status": false,
    "message": "Internal Server Error, user not created"
}
```

Get a User

GET for getting an user.
`GET api/v1/users/:name`
URL parameters: Username is provided in the `:name` param.
Auth required: YES
Permissions required: NO

Success Responses

Condition: A user object is provided in the response.
Code: `200 OK`
Content Example:

```
{
    "status": true,
    "message": "success",
    "user": {
        "username": "UsyNamy",
        "user_role": 1,
        "project_id": 3,
        "ts_created": "2019-03-21T09:14:49.000Z"
    }
}
```

Error Response

Condition: User is not found or internal server error.
Code:`500 Internal Server Error`
Content Example:

```
{
    "status": false,
```

```
    "message": "Internal Server Error"
}
```

Create a project

Creates a new project.
`POST api/v1/project/create`
URL parameters: NONE
Auth required: YES
Permissions required: User has to be admin.
Data Constraints:

```
{
    "name":"name of project",
    "location": "location of the project",
    "contractor": "contractor of the project",
    "ts_project_created": "Unix Timestamp of when the project was created"
}
```

Other constraints: The timestamp should be a unix timestamp.

Success Responses

Condition: Project is created and a project id is returned.
Code: `200 OK`
Content Example:

```
{
    "status": true,
    "message": "Project Created",
    "project_id": 1034
}
```

Error Response

Condition: User is not admin
Code: `400 BAD REQUEST`
Content Example:

```
{
    "status": false,
    "message": "Could not create project"
}
```

OR

Condition: Something went wrong server side.
Code: `500 Internal Server Error`
Content Example:

```
{
    "status": false,
    "message": "Internal Server Error"
}
```

Get a project

Gets a project with its information
`GET api/v1/project/:projectid`
URL parameters: YES, `project_id` is provided in `:projectId` param.
Auth required: YES
Permissions required: User has to member of the project, or be an admin or super user.

Success Responses

Condition: Project is fetched from database and returned.
Code: `200 OK`
Content Example:

```
{
    "status" :true,
    "message": "Returning project 98",
    "project": [
        {
            "project_name": "ProjectsWIthTBM",
            "project_id": 98,
            "location": "Korea",
            "contractor": "Arnes Entrepenarurrurur",
            "ts_project_started": "2019-03-20T09:12:01.000Z",
            "ts_inserted_into_db": "2019-03-20T14:46:51.000Z"
        }
    ]
}
```


Error Response

Condition: Could not find project
Code: `400 Internal Server Error`
Content Example:

```
{
   "status":false,
   "message": "Could not find project"
}
```

Get all projects

Gets all the projects in database.
`GET api/v1/project`
URL parameters: NONE
Auth required: YES
Permissions required: User has to be admin, workshop supervisor or super user.

Success Responses

Condition: A list of projects.
Code: `200 OK`
Content Example:

```
{
   "status": true,
   "message": "Returning projects ",
   "projects": [
      {
         "project_id": 1,
         "name": "Arnes Prosjekt",
         "location": "Valhall",
         "contractor": "LNS",
         "project_started": "2019-03-20T09:15:01.000Z"
      },
      {
         "project_id": 2,
         "name": "KÅshagen",
         "location": "Ytre Enebakk",
         "contractor": "Arne",
         "project_started": "2019-03-20T04:15:01.000Z"
      }
   ]
}
```

Error Response

Condition: No projects
Code: `400 BAD REQUEST`
Content Example:

```
{
   "status": false,
   "message": "Could not find projects"
}
```

OR

Condition: User does not have permission to view this data
Code:`401 UNAUTORIZED`
Content Example:

```
{
   "status": false,
   "message": "Permission Denied"
}
```

OR

Condition: Internal Server Error
Code: `500 Internal Server Error`
Content Example:

```
{
   "status": false,
   "message": "Internal server error"
}
```

Get all TBMs in project

Gets all the TBMs connected to a project.
`GET api/v1/project/:projectId/tbm/all`
URL parameters: :projectId
Auth required: YES
Permissions required: User has to be admin,superuser or member of project

Success Responses

Condition: List of TBMs is returned in body.
Code: `200 OK`
Content Example:


```
{
   "success":true,
   "message": "Retrieving TBMs successful",
```

```
    "tbm": "tbms[]"
}
```

Error Response

Condition: Project doesnt exist or project has no tbms.
Code: `500 Internal Server Error`
Content Example:

```
{
    "status": false,
    "message": "Fetching TBM unsuccesful"
}
```

Create a new TBM in project

Creates a new TBM in a project
`POST api/v1/project/:projectId/tbm/create`
URL parameters: :projectId
Auth required: YES
Permissions required: User has to be admin or project supervisor
Data Constraints:

```
{
  "tbmId":"string",
  "ts_tbm_created": "unix timestamp",
  "housing": [
     {
        "position":"int",
        "wear": "int",
        "max_between_neighbours": "int"
     }
  ]
}
```

Success Responses

Condition: TBM is created
Code: `200 OK`
Content Example: What the response content is to look like. Example:

```
{
    "success":true,
    "message":"TBM tbm203 created"
}
```

Error Response

Condition: Missing parameters in request or user does not have permission to create tbm
Code:`400 BAD REQUEST`
Content Example:

```
{
    "status": false,
    "message": "One or more missing attributes" || "user does not have permission"
}
```

OR

Condition: Error in checking user permission or error in creating the tbm.
Code:`500 Internal Server Error`
Content Example:

```
{
    "status": false,
    "message": "Internal Server Error"
}
```

Get cutters in a TBM

Fetches all the cutters in a TBM
`GET api/v1/project/:projectId/tbm/:tbmId/cutters/`
URL parameters: projectId and tbmId
Auth required: YES
Permissions required: User has to be admin, super user or member of project.

Success Responses

Condition: Found cutters
Code: `200 OK`
Content Example:


```
{
  "success":true,
  "message": "Cutterdata from tbm tbm203",
  "data": "cutters[]"
}
```

Error Response

Condition: Did not find any cutters
Code: 404 NOT FOUND
Content Example:

```
{
   "status": false,
   "message": "Error in fetching cutterdata for tbm tbm203"
}
```

All cutters in project

Gets all the cutters in a project
GET api/v1/project/:projectId/cutters/all
URL parameters: :projectId
Auth required: YES
Permissions required: User has to be admin,super user or member of project

Success Responses

Condition: Found cutters
Code: 200 OK
Content Example:

```
{
  "success":true,
  "message": "Cutterdata from project 3",
  "data": "cutters[]"
}
```

Error Response

Condition: Did not find any cutters
Code: 404 NOT FOUND
Content Example:

```
{
   "status": false,
   "message": "Error in fetching cutterdata for project 3"
}
```

Get a cutter

Fetch a cutter from a project
GET api/v1/project/:projectId/cutter/:cutterId
URL parameters: :projectId, :cutterId
Auth required: YES
Permissions required: User has to be admin,super user or member of project

Success Responses

Condition: Found cutter
Code: 200 OK
Content Example:

```
{
  "status":true,
  "message": "returning cutter 234",
  "data": "{cutter}"
}
```

Error Response

Condition: Could not find cutter
Code:404 NOT FOUND
Content Example:

```
{
   "status": false,
   "message": "No data or wrong project"
}
```

OR

Condition: Error in fetching cutters
Code:500 Internal Server Error
Content Example:

```
{
   "status": false,
   "message": "Internal Server Error"
}
```


Cutter categories

Gets the cutter categories for changing cutters.
GET /api/v1/project/cutters/categories
URL parameters: NONE
Auth required: YES
Permissions required: NONE

Success Responses

Condition: Categories are found
Code: 200 OK
Content Example: What the response content is to look like. Example:

```
{
   "success":true,
   "message":"found categories",
   "categories": "categories[]"
}
```

Error Response

Condition: Error in fetching categories
Code: Ex: 500 Internal Server Error
Content Example:

```
{
   "status": false,
   "message": "Internal Server Error"
}
```

Post Inspection

Creates a new inspection
POST api/v1/project/:projectId/tbm/:tbmId/inspection/post
URL parameters: :projectId, tbmId
Auth required: YES
Permissions required: User has to admin, superuser or member of project
Data Constraints:

```
{
   "ts_start": "unix timestamp",
   "ts_end": "unix timestamp",
   "boredhours": "double",
   "boredmeters": "double",
   "newCutters": [
      {
         "cutterId":"string",
         "ts_cutter_created":"unix timestamp",
         "position": "int"
      }
   ],
   "wear": [
      {
         "position": "int",
         "cutter_id": "string",
         "wear": "int",
         "comment": "string",
         "ts_cutter_wear": "unix timestamp",
      }
   ],
   "changes": [
      {
         "position": "int",
         "comment": "string",
         "category":"string",
         "new_cutter_id":"string",
         "old_cutter_id":"string",
         "ts_cutter_change":"unix timestamp"
      }
   ]
}
```

Success Responses

Condition: Inspection is created
Code: 200 OK
Content Example:

```
{
   "status":true,
   "message": "inspection successful"
}
```

Error Response

Condition: Missing parameters
Code:400 BAD REQUEST
Content Example:


```
{
   "status": false,
   "message": "Missing one or more inputs"
}
```

OR

Condition: Error in creating inspection
Code: `500 Internal Server Error`
Content Example:

```
{
   "status": false,
   "message": "Error in creating a inspection"
}
```

Get inspections on tbm

Gets all the inspections done on a tbm
`GET api/v1/project/:projectId/tbm/:tbmId/inspection/all`
URL parameters: :projectId, :tbmId
Auth required: YES
Permissions required: User has to be admin, super user or member of project

Success Responses

Condition: Inspections found
Code: `200 OK`
Content Example: What the response content is to look like. Example:

```
{
   "status":true,
   "message": "inspectiondata from tbm tbm203",
   "data": "inspections[]"
}
```

Error Response

Condition: Could not find inspections
Code: `404 NOT FOUND`
Content Example:

```
{
   "status": false,
   "message": "error in fetching inspectiondata for tbm203"
}
```

Get 3 latest inspections on tbm

Gets the three latest inspections done on a tbm
`GET api/v1/project/:projectId/tbm/:tbmId/inspection`
URL parameters: :projectId, :tbmId
Auth required: YES
Permissions required: User has to be admin, super user or member of project

Success Responses

Condition: Inspections found
Code: `200 OK`
Content Example: What the response content is to look like. Example:

```
{
   "status":true,
   "message": "inspectiondata from tbm tbm203",
   "data": "inspections[]"
}
```

Error Response

Condition: Could not find inspections
Code: `404 NOT FOUND`
Content Example:

```
{
   "status": false,
   "message": "error in fetching inspectiondata for tbm203"
}
```

6 Security

6.1 Server

6.1.1 Hashing and storing passwords

The server uses bcrypt as hashing algorithm and all passwords are stored as hashes with the same length, alongside a salt in the database.[5]
There are only two ways to create or change a hash using the API, and that is through the registerUser and updatePassword methods in user.controller, which only a administrator or a project supervisor (of the same project as the user to be created/updated) has permission to do.

username	hash	salt
test2	$2b$10$PRdKd8P7oIE/g2kpBsBK/.B7VrMROmL5.UTX5Tvbwic...	b2cdc860bb1913c9

Figur 12: A users credentials stored in the database

6.1.2 JSON Web Tokens

JSON Web Token (JWT) is a digital signed token. For more information about creating a secret and setting expiry date for CutWear server, see 7
CutWear uses jwt for access to the system. Without a token in the request, the request will be denied. To receive a token, see 5
To use the token in a request, put the token as a Bearer token in the request header under x-access-token or authorization.
Example:

$$authorization: \quad Bearer \quad myToken1234$$

$$x-access-token: \quad Bearer \quad myToken1234$$

For more details about JWT: jwt.io or [6]

6.1.3 Let's Encrypt

Communication between client and server uses HTTPS. Let's Encrypt(LE) provides the CutWear server application with free TLS/SSL certificates to ensure secure connections between client and server.[7] The certificate is fetched using the certbot Docker-image. LE certificates expire after 90 days, but with a simple script, the renewal process can be automated. Add the ssl.renew.sh to crontab or an equivalent program to renew the certificates before expiry date. For more detailed information on how to setup auto renewal, see chapter 7
For more details about Let's Encrypt: letsencrypt.org

6.1.4 SQL security

For connecting to the database, the node.js mysql driver is used. Mysql provides a transaction method, which is used for all queries. To prevent SQL-injections, all queries are escaped using parameters. For easier use for developers, model.js has a method "query", which takes in a sql statement and parameters. "query"is transactional and rolls back the changes if the statement fails [8]. All *.model classes should inherit the Model class, and by using the super method "query", making sure that all queries are executed correctly [9].
Example of usage from src/controllers/CutterModel.js:

```
/**
 * Model for reading and writing user data.
 * @extends Model
 */
class UserModel extends Model {
  /**
   * Takes a username and returns the matching row in app_user
   * @param {string} name
   */
  getUser(name) {
    let sql = 'SELECT * FROM app_user WHERE username=?';
    let params = [name];
    return new Promise((resolve, reject) => {
      super.query(sql,params)
      .then(rows => {
        resolve(rows);
      })
      .catch(err => {
        reject(err);
      });
    })

  }
```

Figur 13: Example of SQL-seurity from CutterModel.js

6.1.5 Protecting routes with middleware

Express provides simple ways of integrating middleware in the code, by using the 'use' and 'param' method, one can run security checks before executing the target code [10]. The 'use' method excecutes when a request route matches the route in the "use"method. This makes it possible to check everytime someone requests data which requires special permissions [11]. Example from the route '/api/v1/users/:name' which requires a valid token to be able to request. Figure 14 shows the middleware used for checking tokens. If the token is

```
/**
 * Checks is the token in the request is valid.
 * If valid, send to next middleware,
 * if not valid, return http-400.
 * @param {*} req Request
 * @param {*} res Response
 * @param {*} next Next, function to be called next.
 */
let authUser = (req,res,next) => {
  check.checkToken(req, res)
  // if token is valid, go to next function.
  .then(() => next())
  .catch(err => {
      // if token is invalid or not available, return error message.
      res.status(400).json({
          status: false,
          message: err.message
      });
  })
}
```

Figur 14: The middleware for checking valid token

invalid, it will return a 400, but if the token is valid, it calls the next method, which tells the 'use' method to call the next method in the chain. If the next method is called in authUser, 15 shows that addPool (another

```
// baseroute, auth user and adds pool, then next()
app.use(urlRoot,authUser,addPool,routes);
```

Figur 15: Middleware for all requests starting with urlRoot("api/v1")

middleware) is run, and the request is then sent through to the next handler, which in this case is the route.js, which then again will forward the request to the user.route and then the user.controller.
The param"method is exceuted everytime a certain parameter is present in a route.

6.1.6 Heavy load scenarios

All requests are initially received by the gateway (see 2.4), and therefore works as a load balancer if you wish to run more instances of the server and configure it correctly. If the gateway crashes under heavy load, the cutwear server will not be affected other than it will not receive the requests meant for it.

6.2 Client

6.2.1 Cross Site Scripting(XSS)

ReactJS handles Cross Site Scripting for you by escaping all user input [12].

6.2.2 Client-security

To have a minimal amount of time where the users password is stored as clear text (in variables etc), the password is also hashed (and salted) in the client, before removed from variables. When the client sends a login request it uses the fore mentioned hash to authenticate itself. The hashing adds no further security other than that the password is not in clear text for others to see (backend or frontend).

6.2.3 Timestamps

Since the client has the possibilty of being used offline, there are some problems with knowing what is the correct and newest data. This is handled by timestamping every event on the frontend at the time of the event, and is then sent with the post-request to the database and stored. Then you can have uncommitted events offline on a device, and still not have any conflicts when it is committed. To prevent other errors with using timestamps over different timezones, the timestamp is a UNIX timestamp which is in milliseconds since 1 jan 1970 UTC [13].

7 Installation and running the application

7.1 Server

To install and run the application, you need this software installed:

- node.js (needed for development and production)
- Docker (not necessary for development)
- Docker-compose (needed for production)
- mysql (needed for development and production, if you do not have external database)
- git (useful for easy version control)
- if the documentation is highlighted by %, then it means to replace the text with your own information.

7.1.1 First time installation

1. Clone or pull the latest version from Github
2. If you do not have a pre-configured cutwear database, you need to install the database.
    - Create a database in mysql.
    - Run the ./resources/db/insertFile.sql to setup the database structure.
3. Create the file ./resources/db/db.json.
4. Add the database settings to db.json, like figure 16
5. Create the file ./resources/jwt_token/jwt_token.json
6. Add the jwt secret and expiry date to jwt_token.json, like figure 17

```
{
  "connectionLimit":XXX,
  "host": "hostname",
  "database":"databasename",
  "user":"user",
  "password":"password of db",
  "debug": false,
  "multipleStatements":true
}
```

Figur 16: Example of database connections settings

```
{
    "token":"Secret token, make your own",
    "expiresIn": "when the token expires, can be written like '12h'"
}
```

Figur 17: JWT token settings

7.1.2 Running server outside Docker

1. run 'npm install'
2. run 'npm start'
3. Server is now available on http://localhost:5000

7.1.3 Running server with Docker

1. Build the image
2. Start the image
3. application is now available on http://localhost/

7.1.4 Running server in production with Docker-Compose [1]

1. This configuration is tested on Ubuntu 18.04.
2. Make sure the project is cloned to the '/home/%USERNAME%/' directory, otherwise you need to update the directories in the nginx-conf/nginx.conf file and docker-compose.yml.
3. Update the nginx-conf/nginx.conf with the correct domains. Replace %DOMAIN% with the actual domain.
   Example:
   server_name cutware.ibm.ntnu.no;
   ssl_certificate /etc/letsencrypt/live/cutware.ibm.ntnu.no/fullchain.pem;
   ssl_certificate_key /etc/letsencrypt/live/cutware.ibm.ntnu.no/privkey.pem;
   To add more domains: server_name cutware.ibm.ntnu.no www.cutware.ntnu.no
4. Update docker-compose.yml. Replace the %EMAIL% with an email and replace %DOMAIN% with the same domain from nginx.conf. If you have more than one domain, append -d other.domain.com at the end of command.
   Example:
   command: certonly –webroot –webroot-path=/var/www/html –email krisnyb@stud.ntnu.no –agree-tos –no-eff-email –force-renewal -d cutware.ibm.ntnu.no -d www.cutware.ibm.ntnu.no
5. Creating the keys.
   On the project root: 'mkdir dhparam'.
   Run 'sudo openssl dhparam -out /home/%USERNAME%/cutWearServer/dhparam/dhparam-2048.pem 2048'

6. Setting up auto-renewal of Let's Encrypt certificate. Update the ssl_renew.sh with the correct directory (update %USERNAME%).

7. Open crontab using 'sudo crontab -e'. Add:
'0 12 * * * /home/%USERNAME%/cutWearServer/ssl_renew.sh >> /var/log/cron.log 2>&1'
to the bottom of the file. This will try to renew every 12th hour. A certificate lasts 90 days, so the interval could be bigger.

8. To see logs for certification renewal: 'tail -f /var/log/cron.log'.

9. Build the images with 'docker-compose build'

10. Starting the servers. Make sure Docker-Compose is installed. Run 'docker-compose up -d'. The '-d' tells it to run in the background. Check status of servers: 'docker-compose ps'. Certbot should have exited with 0, and cutwear and gateway should have state=UP.

NOTE: Even though server is running, it might not be possible to log on using tools like curl or postman with the same user and password one uses to log in to the client, since the passwords are hashed on client-side. The production environment setup(with Let's Encrypt, Docker-compose) was created using this.

7.1.5 Handling updates when in production

1. Clone or pull the latest version of the image.

2. Stop the server in question:
sudo docker-compose stop %IMAGENAME%

3. Rebuild the image:
sudo docker-compose build %IMAGENAME%

4. Restart the server:
sudo docker-compose up -d –force-recreate –no-deps %IMAGENAME%.

5. Check the server status (state should be UP):
sudo docker-compose ps

7.1.6 Libraries

This section will contain libraries used in the server with where they are used and what version. Links or other resources to the libraries will be documented here. If the resource is a guide or tutorial, then it has been used in the project as guidance or been used as a learning resource.

bcrypt

- Version 3.0.4
- Usage
  - user.controller.js - used to hash, and to compare two hashes
- Documentation
  - bcrypt-npm

crypto

- Version 1.0.1
- Usage
  - user.controller.js - used to create a random salt of 16 bytes
- Documentation
  - The site containing the algorithm to create salt
  - Built-in to NodeJS

winston

- Version 3.2.1
- Usage
  - ./resources/config/winston.js - configuration file
  - Replaces console.log, and writes to file and console.
- Documentation
  - Github
  - winston-npm
  - Tutorial used for setting winston up for server

jsonwebtoken

- Version 8.5.0
- Usage
  - user.controller.js - Generates a token
  - checkToken.controller.js - Verifies a token
- Documentation
  - Guide for adding JWT to application
  - jsonwebtoken-npm

morgan

- Version 1.9.1
- Usage
  - ./src/app.js - Logs all incoming requests and responses
- Documentation
  - Tutorial used for setting winston up for server
  - morgan-npm

express

- Version 4.16.4
- Usage
  - Web-framework for NodeJS
  - ./src/app.js - Instanced to create a HTTP server
  - ./src/routes/ - Forwards all routes from app.js to the correct routes in route folder.
- Documentation
  - express-npm
  - Express Tutorial Part 4: Routes and controllers

jsDoc

- Version 24.1.0
- Usage
  - npm run doc - used for generating code documentation.
- Documentation
  - jsDoc Dev Docs
  - jsdoc-npm
  - jsDoc cheatsheet

jest

- Version 24.1.0
- Usage
  - npm test - used for running unittests
- Documentation
  - Jest Docs

moment

- Version 2.24.0
- Usage
  - Used to convert unix timestamps (milliseconds since 1 jan 1970) to YYYY-MM-DD H:mm:ss (mysql format)
- Documentation
  - MomentJS

mysql

- Version 2.16.0
- Usage
  - model.js - Used to connect to mysql database using a pool.
  - Uses transactions and roles back on error.
- Documentation
  - mysql-npm
  - Example of creating the model from TDAT3019
  - Creating transactions in mysql-npm

7.2 Client

7.2.1 Creating a installer without Docker

1. Clone or pull the latest version of the client from Github.
2. Set 'NODE_ENV=development'
3. 'electron-builder' has dependencies who rely on python27. Easiest way to avoid the problem is to download and install build essentials.
   For Windows:
   (a) Open CMD as administrator.
   (b) run 'npm install –global –production windows-build-tools'

   For OSX: Python27 should come with OSX.
   For Linux:
   (a) run 'sudo apt-get update'
   (b) run 'sudo apt-get install build-essential'
4. run 'npm install' to install dependencies.
5. Set 'NODE_ENV=production'
6. run 'npm run electron-pack'. This might take a while.
7. The result of the build is located in the 'dist' directory. Run the CutWear Setup version.exe to install on client-pc.

NOTE: if error under npm run electron-pack caused by fsevents, try downgrading the node version to 8.15. This has shown to work if node 10 or 11 errors out. npm i –save-dev node@8.15.0. If error with .sh not finding the build script, set 'NODE_ENV=development' to make sure it downloads the electron-builder dependency, as it is a devDependencie in package.json. To see if that worked, see if './node_modules/.bin/build' exists

For further development, it would be wise to make a Docker image who builds and packages the application, to prevent any error caused by different dependencies, and also make it a part of the continous integration pipeline.

7.2.2 Creating a installer with Docker

1. Make sure Docker is installed, and bash is installed (check if able to run bash scripts)
2. Run './pack_with_docker.sh'. Run with sudo if no permission.
3. The result of the build is located in the 'dist' directory. Run the CutWear Setup version.exe to install on client-pc.

7.2.3 Installing on Windows x64 PC

1. Download or get the installer.
2. Double click the installer and it will install automatically.
3. If there is warning about installing the program, press 'More info' or equivalent, and press 'Run/install anyway'.
4. The application opens upon finishing the installation.
5. A shortcut will be saved on the desktop, called 'CutWear'.

7.2.4 Uninstalling the desktop application

1. Navigate to 'C:/users/%username%/appdata/local/programs/cutwear'
2. Double click the uninstaller and it will uninstall automatically.

7.2.5 Development environment

1. Clone or pull the latest version of the client from Github.
2. run 'npm install'
3. Set 'NODE_ENV=development'.
4. if developer machine is a Windows machine, remove 'BROWSER=none' from 'package.json'.
5. Run 'npm run dev'. A local server and electron starts up.
6. Ctrl+C to exit server and electron.

NOTE: If 'npm run dev' fails because of too many connections error, try setting 'NODE_ENV=production'. This will make electron point to the build files instead of the server. This disables hot-reloading and 'npm run dev' has to be run again. The error message is inconsistent, and does not show up everytime, so it might work after a reboot.
References for all tutorials, guides or other resources used to help with electron and react: [4], [14], [15], [16], [17] and [18].

7.2.6 Libraries

This section will contain libraries used in the client with where they are used and what version. Links or other resources to the libraries will be documented here. If the resource is a guide or tutorial, then it has been used in the project as guidance or been used as a learning resource.

onsen UI

- Version 2.10.6
- Usage
  - Used for styling the app and providing components.
- Documentation
  - `https://onsen.io/v2/guide/react/`
- Parts of the following examples have been used in the application
  - Toolbar: `https://onsen.io/playground/?framework=react&category=reference&module=splitter`
  - Login: `https://onsen.io/playground/?framework=react&category=reference&module=input`

react-router

- Version 4.3.1
- Usage
  - Used for routing between Login and the different Apps
- Documentation
  - `https://www.npmjs.com/package/react-router`
  - Guide: `https://reacttraining.com/react-router/`

react-numpad

- Version 4.1.2
- Usage
  - Used for displaying a numpad for input in the application
- Documentation
  - `https://www.npmjs.com/package/react-numpad`

create-react-app

-

bcrypt

- Version 3.0.5
- Usage
  - src/Services/User.js - used to hash, and to compare two hashes
- Documentation
  - bcrypt-npm

electron

- Version 4.1.0
- Usage
  - Cross-platform desktop app to be used with JavaScript, HTML and CSS.
  - Client is running in a electron application
- Documentation
  - Homepage of electron

electron-builder

- Version 20.39.0
- Usage
  - Node library for building and packaging electron apps.
  - Creates a .exe installer for the client.
- Documentation
  - Github of electron-builder
  - Electron-builder documentation
  - How to build an Electron app using create-react-app. No webpack configuration or 'ejecting' necessary.
  - Using Electron with React: The Basics
  - Turn The Famous React Boilerplate Into Electron Desktop Application
  - Build Electron App using Docker on a Local Machine

7.3 Documentation of source code

For code documentation, jsCode has been used. To generate the docs, first ‘npm install’, then ‘npm run doc’. This will generate a directory called ‘./out’, and in the directory, open ‘index.html’ in a browser of your own choosing.

7.4 Continous Integration and testing

7.4.1 Continous Integration

The continous integration (CI) of CutWear uses TravisCI for running the tests in an Linux Ubuntu Trusty environment and Codecov.io for publishing test results.
The server's continous integration pipeline works as following:

Push to Github –> TravisCI: Run tests –> Codecov.io: Test results and statistics on code coverage is published on codecov.

The configuration is described in 'travis.yml'. The configuration file is a boilerplate travis-node_js configuration [19] with added functionality to push test results to codecov.io after success. [20]

NOTE: as of 15th of May, the project has reached it's maximum builds for private Github repositories in TravisCI. The replacement for TravisCI could be CircleCi, but is not implemented at this time. Running the test locally (see under) will produce the same results.

7.4.2 Tests

The server uses jest for unit testing. To run('npm install' first) the tests, use the script 'npm test' or 'npm test file.js' to test a spesific file. The results of the testing, can be seen in the file './coverage/lcov-report/index.html'. Here one can see the code coverage, which is also available on codecov.io.
The controller methods have been tested for logic, using stubs and mocks for all I/O from the models. The models are not unit tested since they mainly only returns the result from the sql queries, but the main query from Model class has been tested (used by all child models to query database). The controllers have a code coverage of about 97%.
Each SQL-query is tested manually using NTNU Php Myadmin. For testing each REST-endpoint (routes+controllers+models), Postman has been used, but is a manual job, and hard to document in a reusable way.

Referanser

[1] *How To Secure a Containerized Node.js Application with Nginx, Let's Encrypt, and Docker Compose.* side: https://www.digitalocean.com/community/tutorials/how-to-secure-a-containerized-node-js-application-with-nginx-let-s-encrypt-and-docker-compose.

[2] «How To Build a Node.js Application with Docker», side: https://www.digitalocean.com/community/tutorials/how-to-build-a-node-js-application-with-docker.

[3] «Express Tutorial Part 4: Routes and controllers», side: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes.

[4] «Using Electron with React: The Basics», side: https://medium.com/@brockhoff/using-electron-with-react-the-basics-e93f9761f86f.

[5] «Hashing in Action: Understanding bcrypt», side: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/.

[6] «A guide for adding JWT token-based authentication to your single page Node.js applications», side: https://medium.com/dev-bits/a-guide-for-adding-jwt-token-based-authentication-to-your-single-page-nodejs-applications-c403f7cf04f4.

[7] «How to Use SSL/TLS with Node.js», side: https://www.sitepoint.com/how-to-use-ssltls-with-node-js/.

[8] «MySQL transactions in NodeJS», side: https://www.codediesel.com/nodejs/mysql-transactions-in-nodejs/.

[9] «TDAT3019», side: https://gitlab.stud.iie.ntnu.no/nilstesd/DatabaseTest.

[10] «A Simple Explanation Of Express Middleware», side: https://medium.com/@agoiabeladeyemi/a-simple-explanation-of-express-middleware-c68ea839f498.

[11] «Learn how to handle authentication with Node using Passport.js», side: https://medium.freecodecamp.org/learn-how-to-handle-authentication-with-node-using-passport-js-4a56ed18e81e.

[12] «JSX Prevents Injection Attacks», side: https://reactjs.org/docs/introducing-jsx.html#jsx-prevents-injection-attacks.

[13] «What is the unix time stamp?», side: https://www.unixtimestamp.com/.

[14] «How to build an Electron app using create-react-app. No webpack configuration or "ejecting" necessary.», side: https://medium.freecodecamp.org/building-an-electron-application-with-create-react-app-97945861647c.

[15] «Build a File Metadata App in Electron», side: https://codeburst.io/build-a-file-metadata-app-in-electron-a0fe8d32410e.

[16] «How to make an electron app using Create-React-App and Electron with Electron-Builder.», side: https://gist.github.com/matthewjberger/6f42452cb1a2253667942d333ff53404.

[17] «Turn The Famous React Boilerplate Into Electron Desktop Application», side: https://medium.com/@mjangir70/turn-the-famous-react-boilerplate-into-electron-desktop-application-68d91dce8d3a.

[18] «A complete guide to packaging your Electron app», side: https://medium.com/how-to-electron/a-complete-guide-to-packaging-your-electron-app-1bdc717d739f.

[19] «Building a JavaScript and Node.js project», side: https://docs.travis-ci.com/user/languages/javascript-with-nodejs/.

[20] «Build Environment Overview», side: https://docs.travis-ci.com/user/reference/overview/.