Communication service

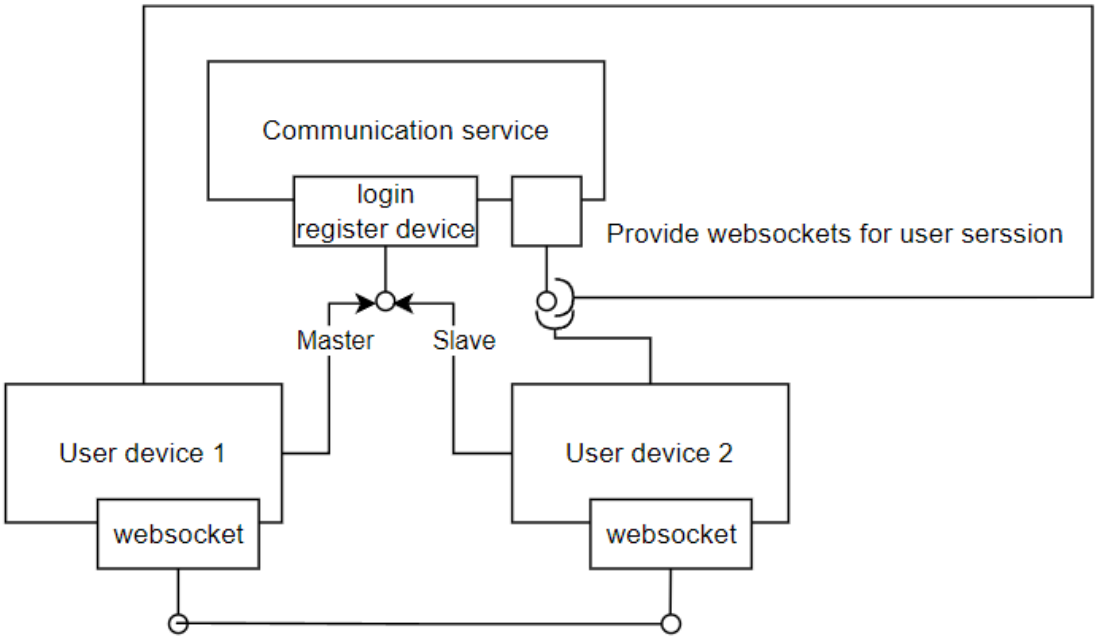
PÄIVÄMÄÄRÄ
Secure Programming

Eero Santala
151179055

Contents

General description

The communication service provides a platform for a user to connect their devices easily. User logs into the service using different devices and registers the device as either “slave” or “master” and after registering the devices the platform provides information for both devices to create a websocket communication between the devices.



Most mobile devices have a dynamic ip-address and port configuration can vary, this makes connecting devices quite annoying in the long run and losing the internet connection can cause a failure to reconnect as the ip-address might change. This is what this service aims to solve.

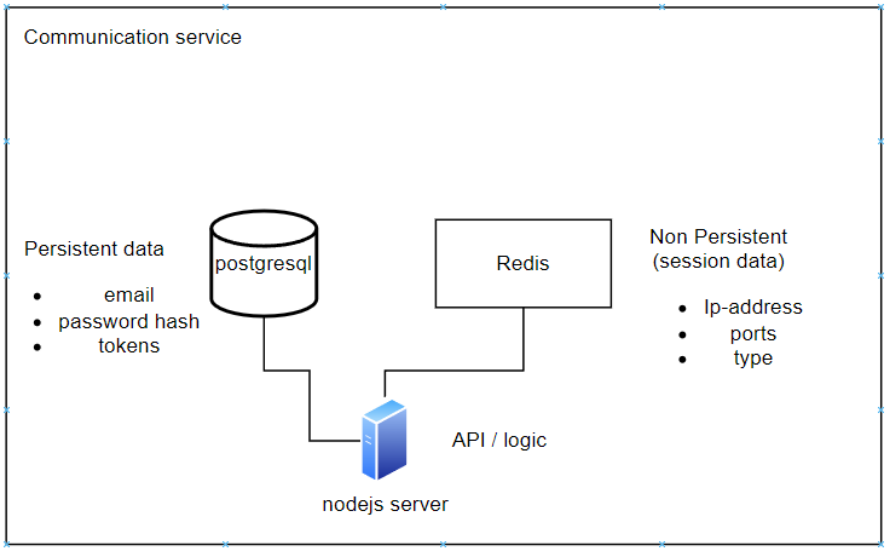
Structure of the program

The program is structured to be modular and clear. Root directory contains configuration settings on the project level such as environment variables, docker-compose files, REST-requests and so on. Separate docker-compose files are for production and testing, testing environment is separated from production and can be ran with single command “docker-compose -f docker-compose_tests.yml up --build”. Having different testing environment enables easy integration testing as the database can be made persistent or cleared before each run.

Moving level down contains Dockerfile for each service if necessary and configuration files for the service. Source code is located in “src” folder where code is structured according to their component type.

```
├── README.md
├── backend
│   ├── Dockerfile
│   ├── Dockerfile.tests
│   ├── package.json
│   ├── server.js
│   ├── sonar-project.properties
│   └── src
│       ├── config
│       │   └── db.js
│       ├── controllers
│       │   ├── authController.js
│       │   ├── deviceController.js
│       │   └── errorController.js
│       ├── middleware
│       │   ├── authMiddleware.js
│       │   └── validationMiddleware.js
│       ├── routes
│       │   ├── authRoutes.js
│       │   ├── deviceRoutes.js
│       │   └── errorRoutes.js
│       └── tests
│           ├── controllers
│           │   ├── authController.test.js
│           │   ├── deviceController.test.js
│           │   └── errorController.test.js
│           ├── middleware
│           │   ├── authMiddleware.test.js
│           │   └── validationMiddleware.test.mjs
│           └── routes
│               └── deviceRoutes.test.js
```

Communication service consists of nodejs server, postgresql database and redis datastorage. Database provides data persistence for our needs, redis stores session related information that by design isn't saved and nodejs server handles all of the logic.



Here is an overview of the components in nodejs server
config required information to form a connection to database are stored here.
controllers functions that are used to perform logical parts of the program, algorithms.
middleware functions that are used to limit or secure other functions. These are used to protect something that comes after them in order to disable malicious activity and reduce the size of the functions they are protecting.

routes endpoint routes and their allowed methods are defined here also describes which middlewares must be passed before information is passed on to the next function.
tests unit test cases are stored here, they follow similar structure as above.

Secure programming solutions

Secure programming practices were the main focus and producing high quality code instead of having larger project with less quality. Several protection aspects were implemented following OWASP Top 10 guidelines.

User authentication

Authentication is based on valid email and password credentials. When a user signs up the password is encrypted using bcrypt with salt round accorinding to $SALT_ROUNDS environment variable (12 in our case). The salted password is stored into the database.

JsonWebTokens are used in logging in a separate authentication and refresh token are generated and stored to the database. Authentication token has an expire time of 15 minutes and refreshtoken 7 days. If the authentication token is expired a new token can be acquired using the refreshtoken. When a user logs out all of the tokens are deleted by design.

Datastorage

Only necessary information are stored in persistent storage using database, those being email-address, hash of the password and tokens. More sensitive information such as ip-addresses and ports are only stored during the session in redis storage.

Input validation

Middlewares are used to validate inputs in the rest requests. Correct method must have only whitelisted fields and the information must be in correct format. Storing to database is performed in strict way to limit attack vectors.

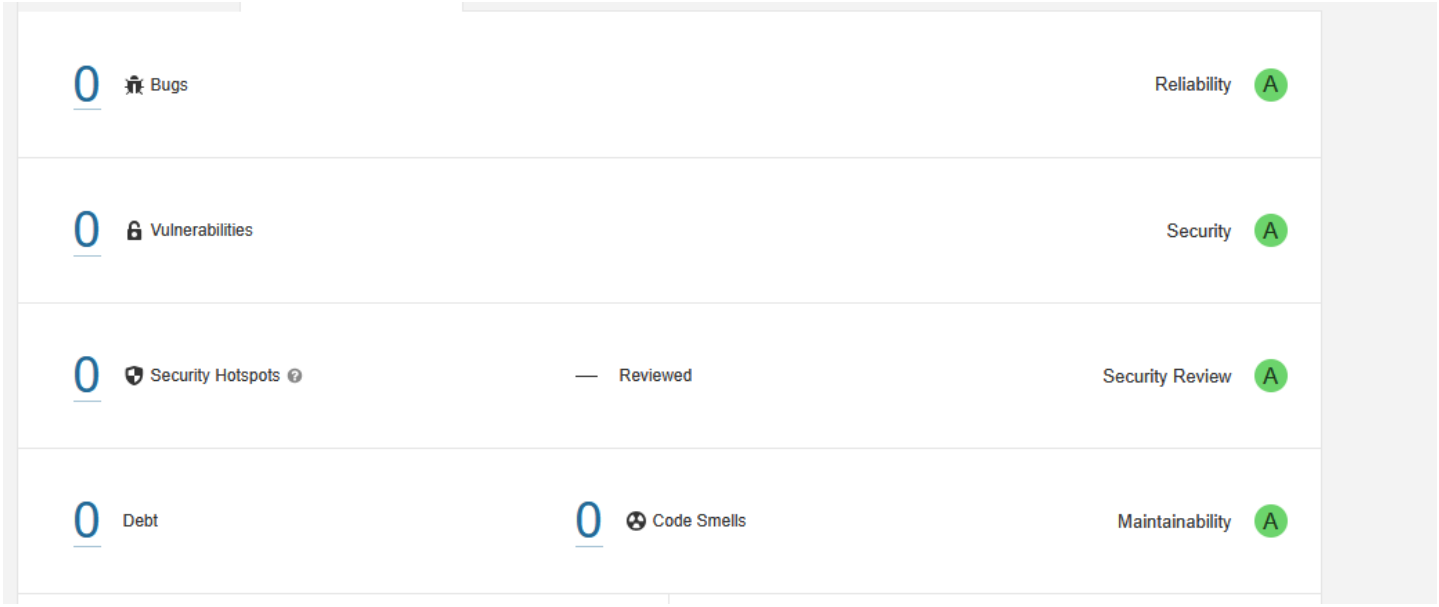
Protected routes

Only allowed routes and methods are enabled, all others are handled with errorRoutes.js. Middlewares are used to protect the endpoint functions.

Security scans

Static code analysis

Sonarqube was used for static code analysis, and all faults found were corrected.

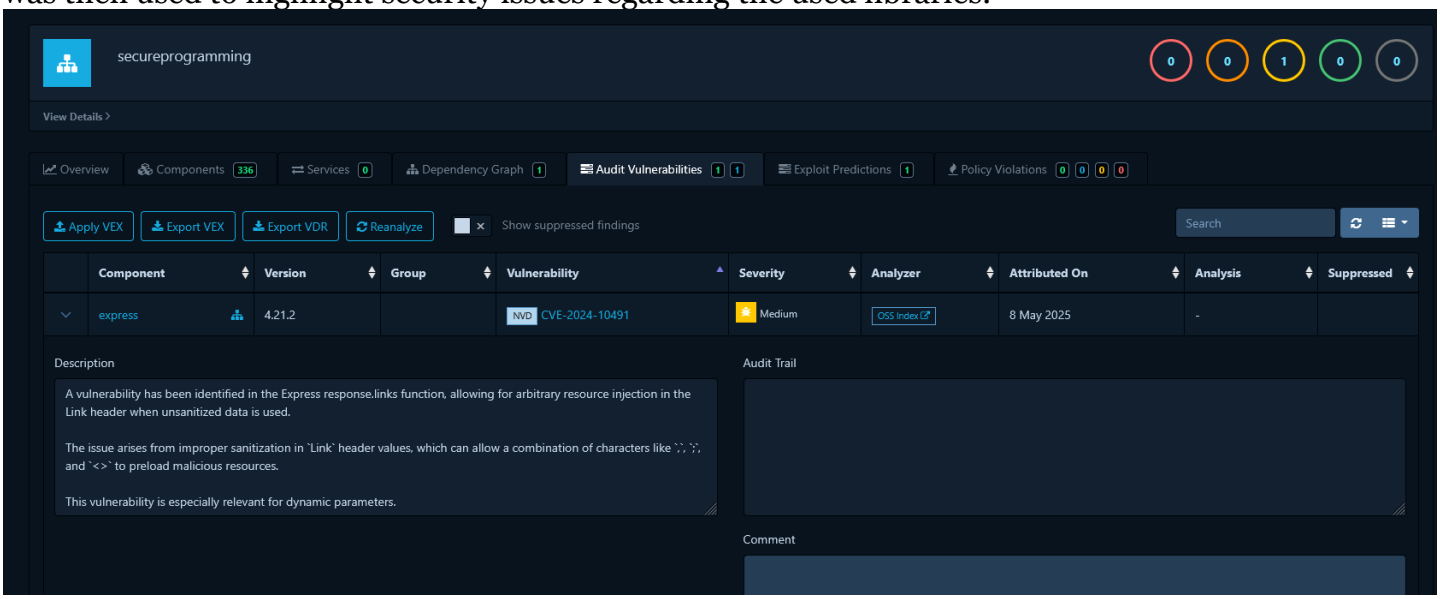


Vulnerability scans

Trivy was used to find vulnerabilities and misconfigurations. All findings were corrected.

```
449a08de36ac (alpine 3.21.3)
============================
Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
```

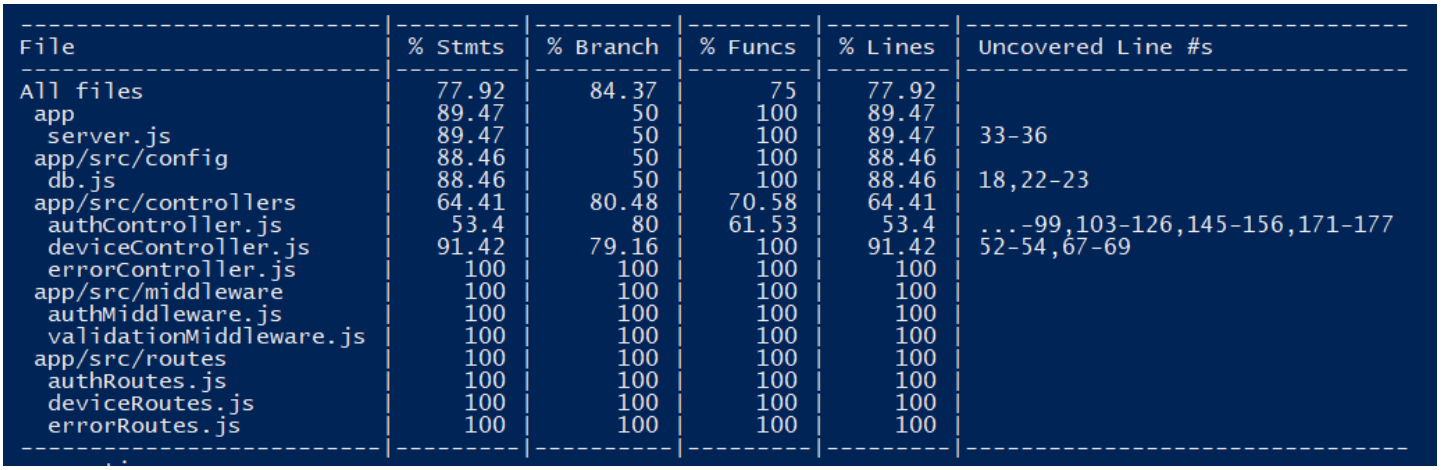
cyclonedx was used to generate SBOM including transitive (indirect) dependencies. Dependencytrack was then used to highlight security issues regarding the used libraries.



Dependency track showed 1 medium vulnerability that comes from Express. This vulnerability is mitigated as resource injection in the link header is not allowed.

Software testing

Unit testing was performed reasonably well and code coverage was considered adequate.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	77.92	84.37	75	77.92	
app	89.47	50	100	89.47	
server.js	89.47	50	100	89.47	33-36
app/src/config	88.46	50	100	88.46	
db.js	88.46	50	100	88.46	18,22-23
app/src/controllers	64.41	80.48	70.58	64.41	
authController.js	53.4	80	61.53	53.4	...-99,103-126,145-156,171-177
deviceController.js	91.42	79.16	100	91.42	52-54,67-69
errorController.js	100	100	100	100	
app/src/middleware	100	100	100	100	
authMiddleware.js	100	100	100	100	
validationMiddleware.js	100	100	100	100	
app/src/routes	100	100	100	100	
authRoutes.js	100	100	100	100	
deviceRoutes.js	100	100	100	100	
errorRoutes.js	100	100	100	100	

Unit testing was performed early on and it gave the project regression testing as a side project.

Integration testing was performed mostly manually, when a new feature was being developed.

Observations and solutions

Sonarcube provided quality improvements to the code and highlighted a potential denial of service vulnerability in a regex method, all the issues were corrected. Dependency track highlighted potential issues regrading nodejs express server, that was considered.

Ai usage

Chatgpt was used to provide some information on errors and help to mitigate them. Also some boilerplate code was generated with chat-gpt.