Design of Dynamic Web Systems - M7011E

Green Power

Institutionen för system- och rymdteknik
Luleå tekniska universitet
971 87 Luleå, Sverige

Hampus Holmström[1]
Edvin Sladic[2]

January 18, 2020

[1]E-post: `hamhol-5@student.ltu.se`
[2]E-post: `edvsla-5@student.ltu.se`

1 Introduction

The Green Power website is an online dashboard for controlling electricity usage from prosumers and managers of such system. Inside the system prosumers can monitor their current electricity consumption and production. The system is supervised and managed by a manager. The manager has a special login and can administrate users but also control the coal power-plant which can produce electricity even when there is no wind in the vicinity.

The dashboard is designed to be easy to intuitive for the user. Hence there is only one main screen on the dashboard where the user can monitor everything available.

2 Design Choices

The Green Power project consists of two major parts; the website itself, and a simulator.

Since there is no real equipment or wind data collected everything is simulated. The simulator is not entirely standalone, but it is well decoupled for the core of the website which makes it easy to replace by another simulator, or real data if available in the future.

Both website and simulator logic are written in JavaScript. The web-server itself uses NodeJS with ExpressJS (2). A MongoDB database is used for booth logging simulator values to the system and the website for storing user data. (3)

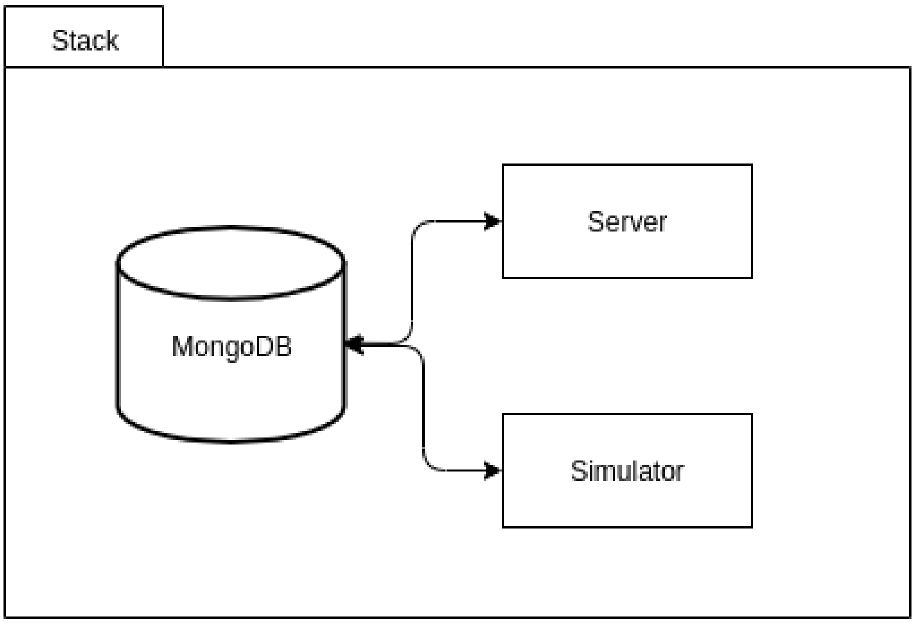

Figure 1: Simplified system overview.

The live version is hosted on AWS with ElasticBeanstalk from a docker container containing the source code. A DevOps approach has been used throughout the development where CI/CD has been setup with much focus being on the CD part towards AWS. On every push to the master branch the environment automatically builds a docker image from the source and pushes it to ElasticBeanstalk which seamlessly put that image into production.

Since the whole system was abstracted into two parts, the server and the simulator, it was decided to have two different database collections as well. One of the key reasons was to offload the server database as much as possible by letting the simulator write global state values to a separate database. Since the global state variables, such as wind, price and general system consumption are standalone and not dependent on specific user information this split up was executable.

The split up of databases in the end probably did not have any remarkable performance gain, but at the time of the decision to do so, it was unclear if the MongoDB driver did lock the whole collection while reading and writing or not. However the cost of splitting up the data was non existing, but resource wise in terms of effort, performance and money.

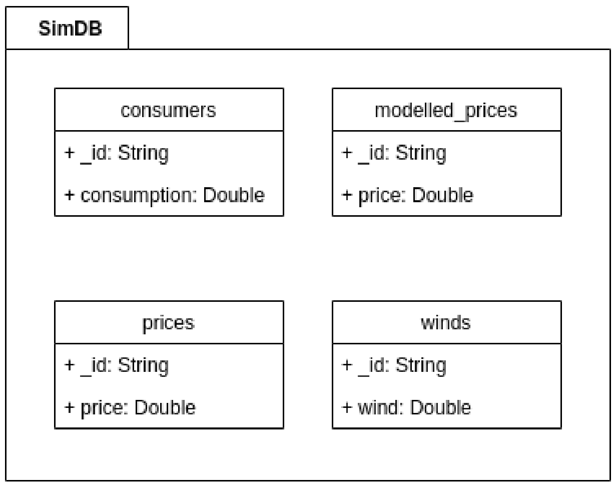

Figure 2: Models of the simulator database.

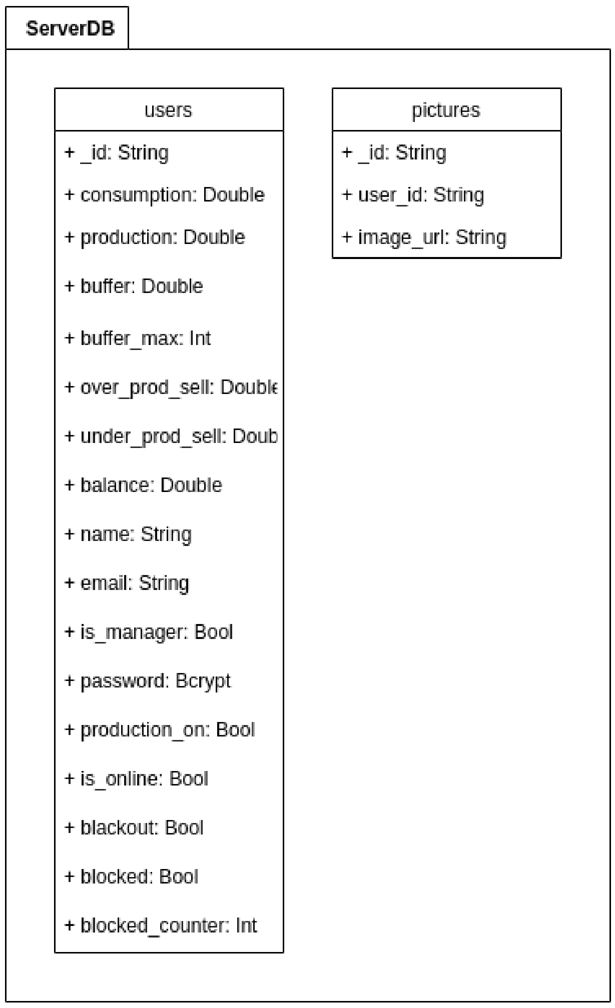

Figure 3: Models of the server database.

3 Scalability Analysis

As previously mentioned...

Precaution has been taken inside the simulator, where some thought has been given to ensure good cashing to speed things up while simulating the environment. At the same time there is still much room for improvements to make the simulator more efficient and improve vertical scalability.

To improve efficiency in the simulator the previous wind is cached, and the next wind simulation is based on that cached value, meaning that the simulator does not read from the wind database when simulating. This improves the simulator performance radically. However writing to the database is still performed after each new wind value simulated to ensure consistency in case of the simulator loosing memory or being shut down unexpectedly.

However there are several reads and writes being made from the simulator to the user database during simulation. This is necessary because the simulator has to act on the latest user data. The logic that performs on the global state and user data is not optimized. For many users the solution may not be sufficient. Slow simulating could in the end lead to unexpected problems where a round of simulation takes too much time and slows down the entire system.

4 Security Analysis

4.1 Production environment

The system used in production, both web-server and database, are protected by strong passwords. The production database is also set to only be accessible trough the IP-address of the physical web-server from Amazon. There are also different users with different access levels to the database, where one is dedicated to be used in production only and has no more access right than necessary.

4.2 Password handling

User passwords are hashed and salted using the bcrypt library which is seen as a good and secure solution for production environments. Each password goes trough 10 rounds of hashing prior to inserting the value to the database, which today is the recommended number of rounds. (4)

Improvements to the password aspect of Green Power would be to enforce users to have a certain length and complexity to their passwords. That enforcement is not implemented which may put the user to risk of not actively choosing a secure password. This is a fairly big issue, and should be implemented since the security will never be better than the weakest password. For now, Green Power, is putting the responsibility on the users to choose a secure password, but users without knowledge may not know the risks of a weak password.

4.3 API protection

The API endpoints are protected in several secure ways. The most important protection is not in the endpoints themselves but the architecture where the web-server is using sessions where a secure session-ID is sent between the client and the server. This makes it harder, in fact impossible, for the user to manipulate an object client side and send back manipulated data to gain unwanted access.

Apart from using sessions each API endpoint handling user data is set to only be accessible by authorized users, which in most cases are only the user itself, but in some cases also includes the manager.

5 Advanced Features

The following features, provided from advanced features of assignment, is implemented in the project.

5.1 Simulator

- Your simulator could provide historical data
- The simulator could, in extension to the REST API, provide streaming simulation data over a socket or similar

5.2 Prosumer

- The monitoring panel can be made more user-friendly by for example including gauges for displaying: Consumption, wind speed and sliders or other suitable controls for determining the ratio from/to the market and buffer
- The monitoring system should be responsive (e.g. have a mobile view)
- The monitoring system should handle multiple logins (e.g. one from mobile, one from desktop)
- There should be a profile page where the Prosumer can update credentials and delete his/her account
- The Prosumer can re-order the visual gauges for example using drag-n-drop
- The data stream from the simulator is made "real time" i.e. streaming from the simulator

5.3 Manager

- User interface is not "polling" over REST, but instead streams data over e.g. a socket
- Should be able to use from multiple devices e.g. desktop and mobile (responsive)
- Should be able to use simultaneously from multiple devices
- There is a visual representation (e.g. Gauges) of the different values and graphical tools (e.g. sliders etc) for controlling the plant
- The operator can re-order the visual gauges for example using drag-n-drop

6 Challenges

To start of, the hardest of challenges might have been the during the start-up of Green Power. No clear picture of how the project should have been structured was provided. Not only was it challenging to start the project but also how the different parts of the system should interact with each other (e.g simulator and server). The key to this challenge was research, basically reading a lot of tutorials on how to build systems using a database and RESTful API. The project got much inspiration at the beginning from this article written by Olatunde Michael Garuba (1). It probably saved a lot in terms of time by just reading how different people build and structure their systems and thus learning from their mistakes instead.

Another hard challenge was to provide the simulator with the right mathematical models in order to post relevant values to the database. Not only bringing the right mathematical model but also how the different simulations depends on each other was challenging. The most efficient way of bypassing this challenge was probably to keep low coupling and high cohesion in the simulation part. Each simulation is divided into single methods, also each call to the database is divided into single methods in a utils file. Whenever a simulation needs access to the database, a call has to be done to appropriate method in the utils file. Regardless, the simulation part ended up as quite hard to understand and not as structured as someone could wish for.

Prior to this project neither of the project members had experience from hosting to a major cloud provider. It was challenging and it took some time to set it up, but in the end it was working. At first an IBM cloud environment was set up, but soon after changed to ElasticBeanstalk on AWS which had great support for CD with docker.

7 Future Work

7.1 User perspective

As already mentioned, enforcement of strong password should be implemented as soon as possible since this is a serious threat towards user accounts.

Apart from security perspective, which otherwise is good, some UI and UX improvements should be made next. One good solution to improve UX would be to develop a first time guide, which guides the user around the interface and describes what each widget does in a tutorial manner.

7.2 Development perspective

From a development perspective, test should be written for key parts of the system, which at this time is the simulator, to ensure a smoother development process but also to identify bugs.

Better logging in production should be implemented to ensure easier debugging on bugs that are found in the production environment. At this time there is no external library or solution used for this, but it would definitely help development to have an infrastructure for logging from the production environment.

A remake of the simulator part could also be done in the future. During the course a draft was made initially on how the simulator should work, but after the picture became clearer the code has piled up and is somewhat not optimized and unstructured in the simulator at this time. However it was decided to stick with the initial simulator since it seemed sufficient for the task, and the team did not want to give too much time in developing the simulator at later stages of the development timeline.

Links

- Github Project: `https://github.com/edvinnn/Green-Power`
- Website: `https://greenpower.sladic.se`

References

[1] Codementor Community (2019) *Build Node.js RESTful APIs in 10 Minutes* Fetched 2019-12-17, from: `https://www.codementor.io/@olatundegaruba/nodejs-restful-apis-in-10-minutes-q0sgsfhbd?fbclid=IwAR0CoQcE0Uf4imJLLiBvDqx7dN7J3VohvQfGPN-gGKZ96xmyYxZ01LNAcPA`

[2] Express (2019) *Fast, unopinionated, minimalist web framework for Node.js* Fetched 2019-12-17, from: `https://expressjs.com/`

[3] MongoDB (2019) *The database for modern applications* Fetched 2019-12-19, from: `https://www.mongodb.com/`

[4] AuthO (2019) *Hashing in Action: Understanding bcrypt* Fetched 2020-01-13, from: `https://auth0.com/blog/hashing-in-action-understanding-bcrypt/`

A Appendix

A.1 Time reports & contribution

Based on the time reports we worked somewhere around 300 hours in total, about 150 hours each. The workload has been fairly equally distributed between us. On a daily basis we have made issues based on what needs to be done. Sometimes we have worked on two issues in parallel, sometimes we have solved more complex problems together. Both of us have worked with all parts of the system, both backend and frontend. The vast majority of the time we have worked together by sitting in the same room.

A.2 Grade analysis

We have put down many hours and put up a system that is provided with several features apart from the basic requirements. Different complex aspects of security is accounted for.
The appropriate grade for both team members should be of grade 5.

A.3 Release

The source code is available at `https://github.com/edvinnn/Green-Power` where also releases can be found under the release section.

A.4 Deployment instructions

To run the website locally it is essential to have both docker and MongoDB installed and running. Then:

1. Clone the source code from GitHub. Make sure to clone submodules as well. Or simply download a release.
2. Run

   ```
   docker build -t greenpower .
   ```

   while standing in the root directory of the project. You can skip this step if you have downloaded a release.
3. Start the server by running

   ```
   docker run -e SERVER_DATABASE=address -e
   SIMULATOR_DATABASE=address -e SECRET=secret
   ...
   greenpower
   ```

   but remember to add all envoiroment variables as an argument. A default `.env` file with default parameters for running on localhost is provided in the repository.
4. The website is now accessible at localhost port 3000.