Redesigning an application for computational chemists to serve their quantum-chemical database

Dennis Granberg
Abstract

Redesigning an application for computational chemists to serve their quantum-chemical database

Dennis Granberg

Computational chemists have access to a web application called Virtual Chemistry which provides models and input data for molecular simulations. The data comes from a database called the Alexandria library, which is continuously worked on by a research group. In addition to Virtual Chemistry another application called Gentop is available which provides further inputs for simulations. Virtual Chemistry is aged with an architecture mainly developed in PHP and Ajax and the demands for the applications capabilities have changed. The areas in the application which is in need for improvements are compatibility with Gentop, maintainability, portability and some usability. The objective of this thesis was to develop a new prototype of Virtual Chemistry to address these issues. During the requirements analysis the use of the frameworks Angular, NodeJs and Docker was motivated with TypeScript as the main language. The prototype was later evaluated with a quantitative task analysis, Google lighthouse and by doing a deployment test. All the results showed improvements in their areas with the task performance halving the time it took to perform a certain task. The prototype laid a solid foundation for being built upon in the future.
Contents

1 Introduction 6
  1.1 Purpose and Goal ............................................. 6
  1.2 Delimitation .................................................. 7

2 Overview of Virtual Chemistry 8

3 Requirements and theory 11
  3.1 Users ......................................................... 11
  3.2 Gap analysis and detailed requirements ...................... 11
  3.3 Non functional requirements .................................. 13
  3.4 Diving deeper ................................................ 13
    3.4.1 Alexandria database .................................... 14

4 Design 16
  4.1 Multi-page and single-page applications ..................... 16
  4.2 Model-View-Controller ..................................... 16
  4.3 Portability .................................................. 17
  4.4 Compatibility with Gentop ................................... 17
  4.5 Frameworks .................................................. 17
  4.6 Architecture ................................................ 18
  4.7 User interface .............................................. 19

5 Implementation 20
  5.1 Server ....................................................... 20
  5.2 Client ......................................................... 20
    5.2.1 Homepage ................................................ 22
    5.2.2 Alexandria and News .................................... 23
    5.2.3 Compound database ..................................... 23
    5.2.4 Molecule ................................................ 25
  5.3 Docker ....................................................... 27

6 Evaluation 27
  6.1 Lighthouse .................................................. 27
  6.2 Task performance ........................................... 28
  6.3 Deployment .................................................. 28
  6.4 Conclusion .................................................. 28
  6.5 Further work ............................................... 29
1 Introduction

The Alexandria library\(^{(1)}\) is an open access quantum-chemical database that is actively being worked on by a research group. This library can assist experimental and theoretical scientists with systematic development and training of empirical force fields for a broad range of molecules. In order to use the library as a resource, the web application Virtual Chemistry was developed. This application allows for a way of checking how well certain molecular species or compounds is reproduced in a specific force field or quantum chemistry calculation. More specifically, the web application features pages to show results from classical molecular dynamics simulations and gas-phase quantum chemistry calculations, a tool to make correlation plots between the available data sets and GROMACS\(^{(2)}\) input files for simulations corresponding to the results.

In addition to the Virtual Chemistry application, a newer application to complement Virtual Chemistry called Gentop was developed. Gentop can be used to generate further inputs for molecular simulations. This application was developed using different technologies compared to Virtual Chemistry.

The first iteration of Virtual Chemistry was completed around 2011, a overhaul was done in 2015 to expand on the functionality of the system. At its core however, the application still use the same architecture and technologies since its inception. The demands of the applications capability has therefore changed as the years progressed and improvements in several aspects are to be desired.

The creators of Virtual Chemistry wish to improve the application in numerous ways. These include better compatibility with Gentop, which would enable the ability to integrate functionality between the two applications, improved maintainability by reducing the numbers of frameworks and languages used by the applications and increased portability to make the application easier to deploy. The user experience of the application needs some improvements as well to keep up with today’s standards.

1.1 Purpose and Goal

The purpose of this thesis is to redesign the Virtual Chemistry application to better suit the needs of its creators and improving the interface for a better user experience. The goal is to have a working prototype of the redesigned application which incorporates these improvements. This prototype can then be used as a foundation for the creators to fully move the application with all its functionality to the new system.

To be more concrete, this thesis will make a new design for the Virtual Chemistry application, implement a prototype of said changes and evaluate them. The prototype will aim to have improved the application in the following aspects:

- Compatibility with Gentop
- Maintainability
- Portability
• Some usability

The work will evaluated by using a tool to measure the quality of the prototype and compare it to the old version. A quantitative task performance will also be measured to ensure that the functionality of the application has not diminished. Lastly, a deployment test will be conducted to see how easy it is to deploy the prototype on other systems.

1.2 Delimitation

This thesis will not rebuild the entire application with all its current functionality as the scale of that work would be too big for this project. Instead a prototype will be built which will lay the foundation of a new application which can later easily be expanded upon. The user interface will also see some improvements but will not be prioritized.
2 Overview of Virtual Chemistry

The homepage of the application is the first view that is presented to the user, see figure 1. The entire application has some common components that are present throughout all the views, namely the navigation menu on the left and the header component up top. Furthermore the homepage has some connection to the database established as evidenced by the statistics graphs. From a design perspective, the entire application follows an overall rudimentary design which feels a bit dated.

![Virtual Chemistry](image)

Virtual chemistry here refers to molecules including both covalently bound compounds and ionic systems in a computer rather than in a laboratory. We try to predict properties of compounds using theoretical models that are calculated or simulated in a computer. The website features:

- Results from classical molecular dynamics simulations of liquids with experimental data to compare results to.
- GROMACS input files for simulations corresponding to the results. Currently three force fields are supported, CGenFF, GAFF and OPLS/AA. We provide validated input files in order to make sure that results based on those files are consistent with the force field used.
- Results from gas-phase quantum chemistry calculations using a range of methods. At a later stage we will provide input files for reproducing these calculations as well.
- A tool to make correlation plots between the available data sets.

Experimental as well as theoretical scientists can use this resource in order to check how well a certain molecular species/compound is reproduced in a specific force field or quantum chemistry calculation. Since we provide experimental and calculated or simulated physical properties side by side, you can form your own opinion. Obviously, these numbers can be used for force field benchmarking and development using other software packages as well.

Enjoy!

**Database Statistics**

![Database Statistics](image)

*Compounds with predicted properties in the database: 3475*

**Liquid Properties (THM entries for I properties)**
- Liquid density
- Surface area
- Static dielectric
- Warburg
- Surface tension
- Viscosity

**Gas Properties (THM entries for I properties)**
- Internal energy
- Entropy
- Density
- Diffusion
- Energy of sublimation
- Standard E

**Solid Properties (362 entries for I properties)**
- Liquid density
- Crystallinity of melting
- Crystallinity of dissolution
- Diffusion of
- Melting E
- Solubility

Figure 1: Homepage of Virtual Chemistry
Navigating further through the application and to the "Compound database" page will take the user to a view of molecules present in the database, see figure 2. The user can filter the table of molecules by their states and other properties. Clicking a molecule will take the user to another view which will contain more detailed information about that specific molecule, see figure 3.

The combined functionality of the Homepage, compound database’s page and the molecule’s page is what the prototype will implement. By choosing these parts of the application to reproduce in the new application will provide a real life scenario on how a user would interact with the application. Furthermore, this prototype will provide a good foundation for expansion of the system.

Figure 2: Compound database view
Figure 3: Molecule view
3 Requirements and theory

As stated in the purpose and goal section of this thesis the work will revolve around replacing an old existing system with a new one. Consequently this means that the usual blueprint of a requirements analysis for a new system will not be followed, instead we will follow other recommended practices common for replacing old systems (3 ch.21). As previously stated, this work will revolve around creating a prototype with only a number of features implemented from the original system, therefore some parts of the application will not be implemented in the new one at this stage.

3.1 Users

One of the most important questions that needs to be addressed when designing an application is Who are the users?. The principle of knowing who the users are, states that the “interaction and interface should cater to the needs and capabilities of the target user of the system in design”(4 ch.1). The users who will be interacting with this system can be split into two categories, the regular users and the administrators. The regular users will mostly consist of experimental and theoretical scientists while the administrators are the ones maintaining the application. As previously stated the user interface has a lower priority in this work, but it will see some welcoming changes to bring the application up to modern standards. Therefore it is important to ensure that the user experience and functionality will be at least on the same level as the original application. This will be evaluated after the implementation stage through a quantitative task performance. The quantitative task performance will be measured by the number of clicks and time it takes to navigate and perform certain tasks on the application, this will then be compared to the old application to ensure that it has not degraded(4 p.130).

3.2 Gap analysis and detailed requirements

A gap analysis compares the old system to the new system in a simple and intuitive way. It shows what functionality to be dropped, migrated and added to the new system, see figure[4] The arrows pointing out and in of nodes show what functionality to be dropped and added respectively. The arrow connecting the nodes shows what functionality to be migrated.
A more detailed look at the requirements for each module of the prototype is as follows:

**Homepage**
- Navigation component
  * There should be a component which allows the users to navigate through the application.
- Third party graphs
  * Graphs should be rendered to display statistics about the database.
- Send query to server
  * The application should be able to make API calls to the server and act on the data responses.

**Compound Database**
- Navigation component
- References component
  * The database holds a table for references to different studies, these should be displayed dynamically on the page based on the data presented.
- Table with pagination
  * There should be a table component which lists all the molecules in the database.
- Filter and search table
  * The table should be able to be filtered and searchable.
- Send queries to server
– Data entries in the table should be clickable.
  * Users should be redirected to a molecule page containing specific information about the molecule which have been clicked.

Molecule

– Navigation component
– References component
– Display summary of current molecule
  * There should be a component which displays a summary of the information present in the database.
– Two tables with liquid and gas properties of current molecule
  * There should be two tables which contains more specific information about the molecules properties which is retrieved from the database.
– Send queries to server

Some of the requirements is common amongst the modules, for example, the navigation component. These components will be developed in such a way that they are reusable.

3.3 Non functional requirements

The new non functional requirements of the prototype are the ones pointed out in the purpose and goal section of this report. The prototype will strive to improve the application in the aspects of compatibility, maintainability, portability and some usability.

3.4 Diving deeper

Context diagrams (§ ch.5), can be used to further understand each modules specific functionality and behaviour. The diagram illustrates the data flow (arrows), external entities (rectangles) and the system in focus (circle). A diagram was drawn for each module the prototype will implement. See figure 5, 6 and 7.
3.4.1 Alexandria database

The Alexandria database is the heart of the application, the entire application is built on serving and presenting the data present in the database. It contains data for molecular optimized geometries, frequencies, thermochemistry, polarizability and molecular electric moments for organic and inorganic molecules. On a more technical level the database is comprised of 14 tables and the size of the file is roughly 70mb. The database is built using SQL and managed by the relational database management system SQLite.
As the application is built for presenting the data present in the database, the entire application is heavily dependent on the database. An entity-relationship model of the database was constructed to better understand the database, which will make the process of retrieving the correct data easier. See figure 8 for the model.

Figure 8: Entity-relationship model of the Alexandria Database
4 Design

The new application will be different from the original version in many aspects. The prototype will have a new architecture to meet the demands of the new requirements. New techniques and frameworks will be introduced and most of the old code base will be discarded.

4.1 Multi-page and single-page applications

The old application appears to be built in the traditional way as a multi-page application (MPA) which means that the browser reloads entire new pages every time a user navigates to new pages. This can be confirmed by looking at the source code of the different pages, much HTML code and connections to the database is repeated. While there is nothing wrong with this approach to web applications, it has its drawbacks in certain aspects when comparing to the newer approach of single-page applications.

Single-page applications (SPA) has gained in popularity in the past couple of years, for good reasons. As the name implies, the entire application runs as a single web page. This means that after the initial page load of the application, all the necessary components of creating and displaying the application has been downloaded and are ready for use. There is no need for full page refreshes when navigating through the application, instead the content of the application are presented dynamically as different views within content regions of the page. The presentation logic resides entirely in the client and is thus separated from the server side, the server side can then focus on data only. By decoupling the client and server in this way allows each to be updated and maintained separately. By designing a web application as an SPA, the application will have the feel and responsiveness of a native application that runs in the browser (5, ch.1).

4.2 Model-View-Controller

MVC (short for Model-View-Controller) is an interactive application development methodology where the idea is to separate the user interface and the logic of an application. The separation of the application is done in three parts, the Model, View and Controller. The model is the part that handles the data of the application, typically this is the database. The model is tasked of retrieving the data needed and provide it to the controller part of the application. The controller in turn is the part that process and manipulates the data before providing it to the view part. The view’s responsibility is to structure and present the data and information to the user. The view is basically what the user sees and interacts with. The idea is that the view cant communicate directly with the model but has to go through the controller and vice versa. This separation allows for better maintainability and easier expansion of the application because modifying one of these parts will not affect the other parts substantially (4, ch.7). This methodology is a good practice to use and something that will be kept in mind when implementing the new Virtual Chemistry prototype.
4.3 Portability

Web applications are usually hosted on servers and in the past only one application could be run at a time per server. When hosting an application, there are many things to take into consideration, the operating system, system specifications and dependencies. Virtualization is a popular method of hosting applications which allows multiple applications to be run on the same server. This is achieved by using Virtual Machines (VM). Each VM has its own dedicated operating system that runs on top of the server operating system. It has its own allocated portion of CPU, RAM and other resources, this provides an isolated environment for running the application without interference.

Another method that is similar to VM’s but more lightweight and portable is that of deploying applications in containers. Containers allows for packaging an application together with libraries and specific dependencies, all in isolated environments, just like VM’s. However instead of virtualizing the entire hardware stack as VM’s do, containers virtualize at the operating system level, meaning that the containers also share the host operating system kernel. This gives containers a number of advantages over the more traditional way of hosting applications. Multiple containers can run at the same time, they are very lightweight and more flexible with the systems resources. They are isolated in consistent predictable environments, developers can create these environments with specific software dependencies within the confines of the container. The container can also run on practically any system and is thus highly portable. All this combined makes a container an excellent choice for developing a portable and easy to deploy application.

4.4 Compatibility with Gentop

Gentop is the other resource used by the same target group. This application is mainly built on the framework Angular. This framework serves as a platform for building scalable web applications. It is built on TypeScript which is an open-source language which builds upon JavaScript. This has consequently led to the platform to be designed around this language and expects the applications built using Angular to use it. One of the new requirements for the prototype is increased compatibility with Gentop. By using the same language, this can be achieved as the code can be re-used in both applications.

4.5 Frameworks

The choices of what frameworks and software the new application will use is pretty straightforward after the analysis of the requirements so far. The client side of the application will be built in Angular. The main reason for this is the requirement of compatibility with Gentop and increased maintainability. Gentop is already built on this framework and by having TypeScript as the main language of both applications will increase the compatibility and maintainability.

The server side will be built in Node.js using the Express.js library. Node.js is a JavaScript runtime designed for scalable applications. Node.js is a prerequisite for developing Angular applications due to the need of transpiling the TypeScript
code to JavaScript code, making the code runnable in browsers. This in turn makes it a good choice as framework for the server and it allows the server to be written in TypeScript as well.

The whole application will be packaged in a Docker container for portability and deployment purposes.

4.6 Architecture

The architecture of the new application will have a separation of the server side and client side. The server side will handle all the interaction with the database and establishing the initial connections of serving the application to the client. The client will thereafter communicate with the server through an application programming interface (API), which serves the client with the requested data from the database. The application only needs to read data and present it, therefore the only API requests needed are GET requests. The entire application will be residing within a container for portability purposes, this will ensure that deploying the application on different hosts will only require the host to have the container environment configured. See figure 9 for an illustration.

Figure 9: Architecture of the new prototype
4.7 User interface

The user interface will see an overhaul to improve the look and feel of the application. The main changes will include moving the navigation bar to the top, a different color scheme and an improved mobile version. See figure 10 for a sketch of the new user interface.
5 Implementation

This section covers how the prototype was implemented. It will not go into a technical level on how every component was developed but rather provide an overall overview.

5.1 Server

The server part of the prototype is quite lightweight, it is in charge of connecting to the database and feed the client with the requested data in a JSON format. It manages this by having prepared statements that executes depending on which API call the client does. Some of these statements can take in parameters to retrieve specific data which the client has requested, this will be used for filtering tables and retrieving data for a specific molecule. An example of a API call and the resulting JSON response can be seen in figures 11 and 12. Implementing the server this way allows for easy way of expanding the functionality prototype when new database requests are needed.

![Figure 11: API function](image1)

![Figure 12: Part of JSON response when calling /phasestats/1](image2)

The server is also in charge of redirecting the user to the homepage when needed, such as when the initial connection is established.

5.2 Client

The client part of the prototype is built in Angular, we have an index.html page which serves as the foundation of the entire application as per the SPA paradigm. The application has a routing module which routes the user to all the different pages. From a design standpoint, the framework Bootstrap was used throughout the application to make simple and aesthetically pleasing modules.

In Angular, the application is composed of different components which can be reused throughout the system. A component is comprised of an HTML page, a TypeScript class and a CSS file which defines the components styling. Each page of the prototype has their own dedicated component which in turn can consist of smaller components. For example the compound database page takes use of smaller components which contains the table of molecules and a list of references. See figure 13 for an illustration of the components.
In addition to the components aspect of the prototype a service was created for handling the API calls to the server. A service is a dependency injection which can be used by the components classes. In the prototype, every part that needs to communicate the server uses this service. To correctly make calls to the server the client redirects all *http calls* made to */api* to the servers port via the `proxy-conf.json` file. See listing 1.

```
{
  "/api": {
    "target": "http://vchemistry-back:8080",
    "secure": false
  }
}
```

Listing 1: Contents of the `proxy-conf.json` file

To adhere to the MVC paradigm as stated earlier in this report, the structure of how the application was built can be seen in figure 14. The controller is the components class (the typescript file), the view is the component template (the HTML and CSS file) and the model is the services that handles the data, in this case it would be the service that communicates with the API of the server.
5.2.1 Homepage

The homepage was the first view to be implemented in the prototype. This page makes four API calls to the server, the first is done to retrieve the number of molecules currently present in the database, the other three are done to retrieve properties on the three different phases of the molecules (gas, liquid and solid). These are rendered into graphs by a third party framework named Chart.js. The SQL statements used to retrieve the data can be seen in listing 15 and the resulting homepage of the prototype can be seen in figure 16.

```javascript
// Homepage statistics
router.get("/molecule_count", (req, res) => {
  var sql = "select count(*) as total from molecules"
  const row = db.prepare(sql).get();
  res.json({
    "message": "success",
    "data": row
  });
});

// Homepage statistics for graphs
router.get("/phasestats/:id", (req, res) => {
  var sql = "select propid, longprop, count(*) as total from molproperty
              natural join proptypes where phaseid = ? group by propid"
  const row = db.prepare(sql).all(req.params.id);
  res.json({
    "message": "success",
    "data": row
  });
});
```

Figure 15: Prepared SQL statements for the homepage
5.2.2 Alexandria and News

The Alexandria and News page of the application are just static pages which contain some information and references about the database as well as news on the work that has been done on the research. As these contain no interesting logic, this report will not go into more detail regarding these pages.

5.2.3 Compound database

The compound database lists all the molecules in the database with some additional information about each specific molecule. This page has functionality for pagination, filtering and quick search. Furthermore, clicking on a specific molecule will redirect the user to a new page containing additional information about the molecule. See figures 17, 18 and 19 for the implemented compound database of the prototype.
Compound Database

The compounds are listed with their systematic names (IUPAC), an alternative name, their Chemical Abstract Service registry number (CAS), ChemSpider ID (CSID) and formula. A full description of how the topologies were generated and how the physical properties are calculated can be found in the references. Suitable for download here.

The force fields used for the benchmark underlying part of this website [2,3], were the Generalized Amber force field (GAFF)[4] and the Optimized Potential for Liquid Simulations for all atoms (OPLS/AA)[5]. More information on these force fields can be found the references and on their respective websites (click on links). We kindly ask that you cite our papers [2,3] if you use the models from this website in your own scientific publications.

<table>
<thead>
<tr>
<th>IUPAC</th>
<th>Formula</th>
<th>Mol. Weight (kD)</th>
<th>CAS</th>
<th>CSID</th>
</tr>
</thead>
<tbody>
<tr>
<td>argon krypton</td>
<td>Ar/Kr</td>
<td>123.748</td>
<td>51184-77-1</td>
<td>11538670</td>
</tr>
<tr>
<td>selenium selenbrryllium</td>
<td>SeSe</td>
<td>87.972</td>
<td>12232–25–6</td>
<td>74875</td>
</tr>
<tr>
<td>calcium selenide</td>
<td>CaSe</td>
<td>119.038</td>
<td>1305–84–6</td>
<td>8385457</td>
</tr>
<tr>
<td>selenium germanium</td>
<td>GeSe</td>
<td>151.6</td>
<td></td>
<td>21770184</td>
</tr>
<tr>
<td>argon helium</td>
<td>Ar/He</td>
<td>43.051</td>
<td>12254–69–2</td>
<td></td>
</tr>
<tr>
<td>helium krypton</td>
<td>He/Kr</td>
<td>87.803</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

References

Figure 17: Compound database page of the prototype
5.2.4 Molecule

The molecule page is where the user is redirected from the compound database page after clicking a molecule. This component is supposed to display detailed information about a specific molecule, which means that this component needs to make several different calls to the server to retrieve the information. The first information a user is met with is the summary of the molecule. Beneath this, two
tables with liquid and gas properties respectively is shown. At the bottom of the page, references to the molecules properties are listed. See figures 20 and 21 for the final result.

Figure 20: Molecule page of “Nitrate”

Figure 21: References to the properties of “Nitrate”
5.3 Docker

When building a production build of the client the resulting distribution folder is placed inside the folder containing the server part of the prototype. The server is instructed to use this build of the client when deployed. Structuring the prototype this way will make it possible to run the entire application inside a single container. A *Dockerfile* is a file that contains instructions on how to build an image in a Docker environment. To build the prototype, the Dockerfile instructs Docker to build a *Node.js* based image, copying all the files and starting the server. The deployment is further simplified by having a *Docker-compose* file which provides some additional instructions, such as which ports to open and have it restart in case of a crash.

6 Evaluation

As stated in the first section the evaluation of the prototype will be done in three parts. First a tool will be used to measure the quality of the prototype against the old version of the application. A task performance will be made to see how long it takes to perform a certain task and compare the results. Lastly the prototype will be tested by deploying it on a different system than the one it was developed one, this is done to ensure that the portability has indeed improved.

6.1 Lighthouse

Google has a tool available for developers called Lighthouse. This tool can measure certain aspects of a web application. The results from running the tool on both versions can be found in figures 22 and 23. The results shows that an improvement has been made in the *best practices* metric, while the *performance* and *accessibility* metrics remain largely the same.

![Figure 22: Old results, Google Lighthouse](image1)

![Figure 23: New results, Google Lighthouse](image2)
6.2 Task performance

The task that was used in this performance test was a simple one, starting from the homepage, navigate to the page containing the properties of the molecule nitric acid. This task was performed on both the old version of the application and the new prototype. The average time and the number of clicks to navigate to the page was measured. There are several way to achieve this task, for this test however the fastest approach was chosen. This approach includes searching for the molecule. The results are compiled in the following table.

<table>
<thead>
<tr>
<th>Application version</th>
<th>Average Time</th>
<th>Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>9.75 s</td>
<td>5</td>
</tr>
<tr>
<td>Prototype</td>
<td>5.35 s</td>
<td>3</td>
</tr>
</tbody>
</table>

The number clicks include the click that is used to start input text into a search bar. As is evident by the results, the prototype performs much better than its original counterpart. This is mainly thanks to the quick search feature which was added to the prototype. The quick search filters the results live after each keystroke, this results in the correct molecule being found without the need to type out the entire name. The live results also does not need a search button due to the nature of how it works. The quick search feature is also present in all the tables, which removes the need to click the search tab before searching.

6.3 Deployment

When testing the deployment of the prototype, a build of the prototype was copied over to a different machine than the one it was developed on. In order to properly deploy the prototype an installation of Docker was required. Once that prerequisite was fulfilled, the prototype could be deployed in the same manner as the previous machine, simply by running the Docker-compose file. This created a new image with the newly developed system running as intended.

6.4 Conclusion

This thesis set out to improve an existing application in numerous ways by rebuilding the application in a new environment. The main areas that was due for an improvement was compatibility with Gentop, maintainability, portability and some usability. The improvements made to the portability and usability aspects of the application was tested earlier in this section to satisfying results. The compatibility aspect was however not tested but due to the fact that the prototype shares the main frameworks used in Gentop, it is safe to assume that this has indeed been improved as well. Likewise, the maintainability aspect was not tested either due to it being a hard thing to measure. Doing similar conclusive work, the fact that the entire prototype is written in the same language and having a clear structure to the components of the prototype, one can also conclude that this too has been improved.
6.5 Further work

The obvious next step to take is to transition the prototype to a full version of the application with all its original functionality. This would make it possible to completely replace the old application with the new one. Further improvements could include adding a login functionality for administrators along with a component to add news and updates.

References


