

Final Report

|  |
| --- |
| School of Computing  Faculty of Engineering AND PHYSICAL SCIENCES |

Backend development using C++

Bogdan-Alexandru Ciurea

Submitted in accordance with the requirements for the degree of  
Computer Science MEng, BSc

2022/23

COMP3931 Individual Project

The candidate confirms that the following have been submitted*:*

*<As an example>*

|  |  |  |
| --- | --- | --- |
| **Items** | **Format** | **Recipient(s) and Date** |
| *Final Report* | *PDF file* | *Uploaded to Minerva (DD/MM/YY)* |
| *Scanned participant consent forms* | *PDF file / file archive* | *Uploaded to Minerva (DD/MM/YY)* |
| *Link to online code repository* | *URL* | *Sent to supervisor and assessor (DD/MM/YY)* |
| *User manuals* | *PDF* | *Sent to client and supervisor (DD/MM/YY)* |

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)

© <Year of Submission> The University of Leeds and <full name of candidate>

# Summary

Nowadays, we are experiencing a shift in the backend development of the programming industry from programming languages that have faster computational speeds but are harder to write to languages that have slower computational speeds but are faster write and also have features such as memory safety or the safety of not managing a buffer overflow attack. This project aims to prove that, although not ideal in some scenarios, it is better for a software in this field to use better compile speeds, and, to prove this, a backend software is going to be developed in the C++ language.

The software that this project intends to build is a fully usable school management system/web application that is going to have multiple features such as managing users, files, courses, or announcements or assigning grades/results for different users. Furthermore, multiple non-functional requirements are applied to make the system more complex but also to better compare this software with other methods of writing such as software. The non-functional requirements are going to be the security of the system and the privacy of the data but, the most important will be the scalability and the reliability of our system.

The last two features are going to be achieved by choosing a suitable database that is going to fulfil the needs of having a database that is capable of storing large amounts of data, a big number of writes/reads per minute but also has a replication factor so that, in the case that the database is down for any reason, the data is not going to be lost. One of the databases that fulfils these needs, but it also has a C++ library to be used by our backend, is Apache Cassandra.

Considering that this application is intended to be deployed and used by any person that does not necessarily know computer science, the deliverable is also going to have a basic frontend to be used in the browser. For this, the software is going to be written in Next.js and is going to display to the user how the information may look in a better format than JSON.

# Acknowledgements

*<This page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by others to the project.>*

*Note that it is not acceptable to solicit assistance on ‘proof reading’ which is defined as “the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the text”; see*

https:://www.leeds.ac.uk/secretariat/documents/proof\_reading\_policy.pdf

# Table of Contents (example of how to format)

Summary iii

Acknowledgements iv

Table of Contents (example of how to format) v

Chapter 1 Introduction and Background Research 1

1.1 Introduction 1

1.2 Literature Review 1

Chapter 2 Methods 2

2.1 Backlog and UML diagrams 2

2.2 Overall architecture 2

2.3 Database 2

2.4 Database integration with C++ 2

2.5 Relationship managers 2

2.6 API Mapping and request/response mapping 2

2.7 Frontend 2

2.8 Deployment 2

Chapter 3 Results 3

3.1 Tests for database - C++ integration 3

3.1.1 Database speed 3

3.1.2 Actual tests 3

3.2 ±Tests for relations managers 3

3.3 API speed 3

Chapter 4 Discussion 4

4.1 Conclusions 4

4.2 Ideas for future work 4

List of References 5

Appendix A Self-appraisal 6

A.1 Critical self-evaluation 6

A.2 Personal reﬂection and lessons learned 6

A.3 Legal, social, ethical and professional issues 6

A.3.1 Legal issues 6

A.3.2 Social issues 6

<Discussion of social issues> 6

A.3.3 Ethical issues 6

<Discussion of ethical issues> 6

A.3.4 Professional issues 6

Appendix B External Materials 7

# Chapter 1 Introduction and Background Research

## 1.1 Introduction

Nowadays, we are experiencing a shift in the backend development of the programming industry from programming languages that have faster computational speeds but are harder to write to languages that have slower computational speeds but are faster write and also have features such as memory safety or the safety of not managing a buffer overflow attack. This project aims to prove that, although not ideal in some scenarios, it is better for a software in this field to use better compile speeds, and, to prove this, a backend software is going to be developed in the C++ language.

The software that this project intends to build is a fully usable school management system/web application that is going to have multiple features such as managing users, files, courses, or announcements or assigning grades/results for different users. Furthermore, multiple non-functional requirements are applied to make the system more complex but also to better compare this software with other methods of writing such as software. The non-functional requirements are going to be the security of the system and the privacy of the data but, the most important will be the scalability and the reliability of our system.

The last two features are going to be achieved by choosing a suitable database that is going to fulfil the needs of having a database that is capable of storing large amounts of data, a big number of writes/reads per minute but also has a replication factor so that, in the case that the database is down for any reason, the data is not going to be lost. One of the databases that fulfils these needs, but it also has a C++ library to be used by our backend, is Apache Cassandra.

Considering that this application is intended to be deployed and used by any person that does not necessarily know computer science, the deliverable is also going to have a basic frontend to be used in the browser. For this, the software is going to be written in Next.js and is going to display to the user how the information may look in a better format than JSON.

## 1.2 Background reach

## 1.3 Existing solutions

There are many methods of building a backend server with the same properties that the described system has (those being having a backed server independent from a frontend client)

## 1.4 Requirements

In order to define exactly what the application should do, the next functional and non-functional requirement have been defined:

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Type | UI Component | Priority |
| Support user accounts and user login | Functional | 1,2,3 | 1 |
| Store and send the data securely | Nonfunctional |  | 2 |
| Avoid possible leaks of information | Nonfunctional |  | 2 |
| Create new users. The password will be sent to the new user via email | Functional | 1 | 1 |
| Create/modify/delete lectures | Functional | 1,2 | 1 |
| Create/modify/delete grades | Functional | 2 | 1 |
| Display the calendar for each user | Functional | 2,3 | 2 |
| Send email to user references when a grade is received | Functional | 3 | 3 |
| Allow users to add folders/files to a course | Functional | 1,2,3 | 2 |
| Allow users to create/edit/delete custom tags for users | Functional | 1,2 | 3 |
| Students may ask questions on the course page | Functional | 1,2,3, | 3 |
| Users can respond the announcements | Functional | 1,2,3 | 3 |
| Allow users to create/ edit/delete a personal todo list | Functional | 1,2,3 | 3 |

*UI Component 1 – admin; 2 – teacher; 3 – student*

## 1.5 Chosen technologies and frameworks

### 1.5.1. Database

The database chosen for this project is Apache Cassandra. The main reasons why I chose this database are:

1. Scalability and reliability. Cassandra clusters/nodes can be connected, and they will communicate between them so that the data is stored on every node. This will ensure that, if one node breaks down, the data will not be lost but will be spread around all the available nodes.
2. Chart

   Description automatically generated with medium confidenceSpeed. Comparing to other databases, Cassandra is much faster especially when we are using it with multiple connected nodes as we can see from the article “Performance test on Cassandra NoSQL”. The problem that we are going to run into is the slow read of Cassandra but this, can be overcome with structuring our data in a way that allows the database to execute the fast reads.
3. How the data is stored. Cassandra is a nonrelational database, and it is not stored in a json format, but as a table. Although we will not be able to use some of the features a relation database has (such as joins), Cassandra has an important feature: we can use maps, sets and lists.
4. Compatibility with C++. The [Cassandra cpp-driver](https://github.com/datastax/cpp-driver/tree/2.16.0) made by datastax is an easy to use, reliable and tested driver used by multiple companies for their software.

### 1.5.2 Programming language

In order to achieve this project’s scope, the programming language that was chosen is C++. The main reasons are that C++ has one of the fastest compile speeds (reference: https://link.springer.com/article/10.1186/1471-2105-9-82) but it also supports classes, compared to C.

On the other side, one of the main problems that we will have to take care is the possibility of a heap attack, therefore making out system less secure and vulnerable to a hacker attack. This problem will be taken care of automatically, using the already implemented API Framework.

### 1.5.3 API Framework

In order to make the work process faster, the backend will use an already existent API framework. The one that was chosen for the project is the Drogon C++ library as it already has implemented features such as asynchronous programming, SSL communication, endpoint definitions that accept parameters, JSON formatting but also file download and upload. One the only problems with this library is that the documentation does not go too deep into explaining some features, the integrations with the browsers CORS policy is lacking in functionality and it is also hard to deploy, as the building of the library it time consuming but it also takes more than 1Gb of storage.

### 1.5.4 Other Frameworks used in the backend

In order to successfully achieve some of the functional and non-functional requirements previously mentioned, the application will need to have some additional modules added. For this the next libraries have been imported:

|  |  |  |
| --- | --- | --- |
| Name | Link | Functionality |
| Google tests | https://github.com/google/googletest.git | Used for testing the application. |
| SMTPMail | https://github.com/ihmc3jn09hk/SMTPMail-drogon.git | Used for sending emails. |
| Bcrypt | https://github.com/hilch/Bcrypt.cpp.git | Used for encrypting passwords |
| JWT-cpp | https://github.com/Thalhammer/jwt-cpp | Used for header encryption of the request |

### 1.5.5 Frontend

For the frontend part of the application, one of the most convenient libraries to build and deploy at the moment, is Next.js. It is a library built on top of the React stack, therefore it will be able to use any React libraries. The reasons why this library was chosen are:

* Automatic routing. The routing of the browser pages is automatically done by the library.
* Server side rendering. Next.js increases the security of the application (compared to base React) by automatically creating the pages in the backend rather than on the user’s browser.
* Easy deployment. Because Next.js is built by the company Vercel, they have one of the best deployment experiences for any Next.js application. Even more, the deployment is also optimised for better runtimes.

# Chapter 2 Methods

## 2.1 Backlog and UML diagrams

Talk about:

* Functional requirements
* Add UML diagrams

## 2.2 Overall architecture

The application is going to be split into 3 main parts: the database node(s), the backend and the frontend.

## 2.3 Database

### 2.3.1 Database diagram

The table will be split into two keyspaces (think of them as some kind of groups). The environment keyspace will contain the tables schools, countries, and holidays\_by\_country\_or\_school. The rest of the tables will be stored in the schools keyspace.

TODO: Redo the database design in <https://www.lucidchart.com/>

Diagram

Description automatically generated

### 2.3.2 Database objects in C++

For most of the tables there will be a C++ class that will store the correlated information from the database. This is done in order to keep the information easy to access.

The tables that will have a class will be:

|  |  |
| --- | --- |
| Database table | Correlated C++ class |
| environment.schools | SchoolObject |
| environment.countries | CountyObject |
| environment.holidays\_by\_country\_or\_school | HolidayObject |
| schools.announcements | AnnouncementObject |
| schools.answers | AnswerObject |
| schools.course | CourseObject |
| schools.tags | TagObject |

The classes will be simple and will have three main parts: the constructor and destructor, a function that will return the data in a Json format and all the variables that the class will have, these being all the fields that can be found in the correlated table.

The constructor will take as parameters a variable for each of the class’s variables. Note that, for every uuid variable, the constructor will take a string as parameter and change it to a uuid later.

As mentioned, each of these classes will have a function named Json::Value to\_json(bool secure) that will be responsible for returning the class’s data in a json format so that it can be transferred from the backend to the frontend.

The secure parameter represents if we want to add information such as the school’s id of the object. Considering that the end user will not need to receive such information, the system will not give it because of security reasons.

Furthermore, all the variables will be stored as public variables of the classes in order to keep the code clean. Usually, in production, in order to get or set a variable, you would have to add two functions for each variable of a class: a getter and a setter, but, as mentioned above, we will prioritise keeping things simple and clear and not using this approach.

## 2.4 Database integration with C++

Copy paste from the 3rd document (CQL Managers). This will add another 7 pages.

## 2.5 Relationship managers

Quickly talk about the same ideas a the 4th document (relationship Managers). This will add 4 pages. Probably add the entire document in the Appendix.

## 2.6 API Mapping and request/response information

Talk about the information that is going to be received and parsed by the system, as well as the information that is going to be responded with.

In this section I am probably not going to go into much detail into the API but I will add a table something like this image to make it clear exactly what endpoints are opened:

Table

Description automatically generated

Image taken from the 1st coursework for Web Services and Web Data; this part of the table has been done by me.

Go further into detail and structure each API endpoint as follows:

Table

Description automatically generated

## 2.7 Frontend

## 2.8 Deployment

Describe the communication between the docker containers and how they will work.

## 2.9 Software development

### 2.9.1 Version control

Using GitLab

Graphical user interface, application

Description automatically generated

### 2.9.2 Work distribution

The work was distributed into sprints that wanted to achieve different results.

|  |  |
| --- | --- |
| Sprint period | Goal that the sprint wanted to achieve |
| 1st November 2022 – 15th November 2022 | Project setup and definition of features |
| 16th November 2022 – 31st December 2022 | Database Setup and Integration |
| 1st January 2023 – 31st January 2023 | Relations Managers |
| 1st February 2023 – 31st February 2023 | API Definitions |
| 1st March 2023 – 31st March 2023 | Frontend Work |
| 1st April 2023 – 31st April 2023 | Testing, bug fixing, and finalizing the project for release. |

# Chapter 3 Results

## 3.1 Tests for database - C++ integration

### 3.1.1 Database speed

### 3.1.2 Actual tests

## 3.2 Tests for relations managers

## 3.3 API speed and testing

Compare the resulted speeds of our backend to the average speeds of a normal API. The response time is usually around 80ms from my observations but a diagram to show the results is needed.

If the average response time of the API is lower than 100ms we can consider that the speeds is “excellent” as an article from *Datadome.co* points out

TODO: Move this to the discussion chapter

Graphical user interface, text, application, email

Description automatically generated

The testing was done using Postman and each of the endpoint was tested individually.

# Chapter 4 Discussion

<Everything that comes under the `Results and Discussion' criterion in the mark scheme that has not been addressed in an earlier chapter should be included in this final chapter. The following section headings are suggestions only.>

## 4.1 Conclusions

<Text in 11-point size and 1.5 line spacing.>

## 4.2 Ideas for future work

<Text in 11-point size and 1.5 line spacing.>

4.3 Limitations

# List of References

Anon 2019. Comparing database management systems: MySQL, PostgreSQL, MSSQL server, MongoDB, elasticsearch, and others. *AltexSoft*. [Online]. [Accessed 20 March 2023]. Available from: <https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/>.

Anon 2018. Performance test on Cassandra NoSQL. *Isaac*. [Online]. [Accessed 20 March 2023]. Available from: <https://www.isaacbigdata.com/performance-cassandra/>.

Anon n.d. *Datadome.co*. [Online]. [Accessed 20 March 2023]. Available from: https://datadome.co/learning-center/how-to-reduce-server-response-time/.

# Appendix A Self-appraisal

<This appendix must contain everything covered under the ’self-appraisal’ criterion in the mark scheme. Although there is no length limit for this section, 2-4 pages will normally be suﬃcient. The format of this section is not prescribed, but you may like to consider the following sections and subsections.>

## A.1 Critical self-evaluation

## A.2 Personal reﬂection and lessons learned

## A.3 Legal, social, ethical and professional issues

<Refer to each of these issues in turn. If one or more is not relevant to your project, you should still explain *why* you think it was not relevant.>

### A.3.1 Legal issues

<Discussion of legal issues>

### A.3.2 Social issues

### <Discussion of social issues>

### A.3.3 Ethical issues

### <Discussion of ethical issues>

### A.3.4 Professional issues

<Discussion of professional Issues>

# Appendix B External Materials

<This appendix should provide a brief record of materials used in the solution that are not the student's own work. Such materials might be pieces of codes made available from a research group/company or from the internet, datasets prepared by external users or any preliminary materials/drafts/notes provided by a supervisor. It should be clear what was used as ready-made components and what was developed as part of the project. This appendix should be included even if no external materials were used, in which case a statement to that effect is all that is required.>