2DV609

GymUP!
Design Document

Author 1: Fabian Dacic
Author 2: Yuyao Duan
Author 3: Fredric Eriksson Sepúlveda
Author 4: Li Ang Hu
Author 5: Long Ma

Contents

Introduction **5**

Purpose **6**
- Design decisions 7
- Constraints that limit the design 8
- Justifications that shape the design 8

Design goals and philosophy **10**
- Philosophy of the design 10
- System efficiency under unusual circumstances 10

Assumptions and dependencies **12**
- Application type 12
- Service object and user interfaces 12
- Application development 12
- Development resources 13
- Software deployment 13
- Design pattern analysis 13

Architecturally significant requirements **14**
- Customer Booking functionality 14
- Administrator Functions 14
- Program Main functionality 14
- Customizable program 15
- Administrator Functions 15
- Database Functionality 15

Decisions, constraints, priorities, and justifications **16**
- Type of application: Client-server (web-application) 16
  - Advantages of web-based applications 17
  - Disadvantages of web-based applications 17
  - Alternatives of web-based applications 18
  - Justification for web-based applications 18
- Architectural pattern: Multi-layer architecture 19
  - Architectural pattern: Multi-layer architecture 19
  - Disadvantages of multi-layer architecture 20
  - Alternatives of multi-layer architecture 20
  - Justification for multi-layer architecture 21
- Back-end Frameworks: SpringBoot and MyBatis-Plus 23
  - Spring 23

SpringMVC 24
SpringBoot 25
MyBatis-plus 25
Justification for the back-end frameworks 26
Front-end languages: HTML, CSS, and JS 26
Hyper-text markup language 26
Cascade style sheets 27
JavaScript 27
Justification for the front-end languages 28
Mixed of top-down and bottom-up 28

Key abstractions 30
Member 31
Administrator 31
Equipment 31
Booking 31

Architectural Patterns 32
Architectural Pattern: Multi-Layered 32
Client: Thin Client 32
Multi Layered Layers: 32
Client Layers 32
Presentation Layer 32
Controller Layer 32
Server Layers 33
DAL Data Access Layer: 33
Functionality Layers 33
Business Layer 33
MySQL Database 33
Mybatis 33
Legacy code 33
How does a multi-layered pattern follow the design principles? 34
Design Principle 1: Divide and conquer 34
Design Principle 2: Increase cohesion where possible 34
Design Principle 3: Reduce coupling 34
Design Principle 4: Keep the level of abstraction as high as possible 34
Design Principle 5: Increase reusability where possible 34
Design Principle 6: Reuse existing designs and code where possible 34
Design Principle 7: Design for flexibility 35
Design Principle 8: Anticipate obsolescence 35

Design Principle 9: Design for Portability 35
Design Principle 10: Design for Testability 35
Design Principle 11: Design defensively 35

Architectural views 36
Component diagram 36
Deployment diagram 37
Sequence diagrams 38
Booking equipment 38
Member checks all of their own bookings 40

Performance modeling and analysis 41
Table of calculation result 42
Model 42
System Response time 42
Throughput 43
Utilization 44
Discussion 45

References 46

Appendix – Time Report 49

1. Introduction

This design document discusses various aspects related to the design of the application intended for the gym equipment booking system named GymUP! The document addresses themes such as the purpose of the design, the goals, philosophy of the design, assumptions and dependencies, software architecture and so forth.

The design document will serve as a foundation for the application however it is to be noted that it is flexible as well considering that decisions may change throughout the project's timeline. Throughout the document, there are also references to the previous document regarding requirements' specification.

System's architecture and system design are overlapping terms however can be processed with different and separate stages in a project's timeline therefore prior knowledge of the terms related to software design would be beneficial long-term. When it comes to views, this document serves as a mixed high and low level overview of the design itself and the goals that the team aim to accomplish. This document also covers to an extent business and technical requirements of the design and product as a whole.

2. Purpose

The design philosophy will be dictated by intuitiveness, flexibility, function over form, empathy and focus, simplicity and minimalistic approach, and last but not least, open for extension and closed for modification principle. This design philosophy is focused on the client's requests and the utility of the product.

1. **Intuitiveness**

   The reason for intuitiveness as a design philosophy is to ensure that the design itself is easy to understand and simple to implement. When a design is intuitive, it is usually meant that it will utilize well-known principles from various domains and once the product is provided, the users will not have to wonder about how to operate it. This is more related and a reference to the user interface and user experience in general.

2. **Flexibility**

   The reason for flexibility as a design philosophy is to ensure that the design can be flexible for additional features meaning that the software itself needs to be flexible i.e needs to be designed so that it changes to the most common type of change it will encounter.

3. **Functions over form**

   With function over form, it is implied that the shape of the product should primarily relate to its intended function and this should be prioritized over the aesthetics of the product.

4. **Empathy and focus**

   The reason for empathy and focus as a design philosophy is to ensure that the decisions made throughout the design process should also involve trying to satisfy the client's requests and focus on requests relevant to the product which in turn will lead to a pleasant experience.

5. **Simplicity**

The reason for simplicity as a design philosophy is to ensure that the design is a minimalistic one meaning that complex designs will only overcomplicate the process of delivering the actual product.

6. **Open for extension, closed for modification**

   This principle is quite important since whenever adding new features, it should be done by simply adding new code instead of modifying the old one therefore foresight is needed throughout the process of delivering the design and product in the end as well.

   This also involves iterative processes of feedback and refactoring.

2.1. Design decisions

Due to the nature of the system, it being more of a complementary system to a domain one and it is heavily focused on providing users with a product that is intuitive to use, design decisions will be comprehensive and suitable in a booking environment.

Considering that the approach is that the product will serve more as a module that can be attached to various gym-related software applications, this is in agreement with the design philosophy aforementioned above, mostly related to flexibility.

The legacy system, which in this document will refer to the organization's already existing code, during development, will involve adding new code instead of vastly modifying it. This decision is also in line with the open for extension, closed for modification principle although it is important to note that one should strive to achieve such a goal however in most real-life cases, it is quite difficult to do so.

A database will also be utilized and it will store and organize data related to the gym and that the product will use throughout its life cycle. The database for the system will be a cloud database which is to be accessed through a cloud platform. This results in ease of access for the parties involved, scalability, and relatively effective disaster recovery as backups are kept.

The product is ought to be a web application with its focus being mobile applications however it should be accessible by personal computers as well. This is due to many reasons, primarily because all users can access the same

version therefore compatibility issues are mitigated, flexible in regards to platforms, decreases hardware constraints, etc.

2.2. Constraints that limit the design

Considering that the product ought to be a module capable of being attached to already existing software applications, it requires a close connection to it in regards to data. By that, it means that the gym contains the necessary information for the product to function properly i.e members and equipment related information and in turn, the product needs to be able to communicate the changes back to the domain system. This creates a sort of supplier-dependability.

Considering that the product is a web application, it is important to note that the performance is heavily dependent on the response of the browser from which the application is running and quite dependent on the internet, the user experience is at various times a drawback since web applications tend to adapt less to the device used and offline functionalities are often excluded. There is also the matter of security, as with web applications, customers expect sufficient security measures to protect their data.

Considering that the database will be a cloud one, there are a few points to keep in mind as it can be quite efficient and productive in the long term. There are also drawbacks. Cloud databases are notorious for being vulnerable to attacks, during the past few years, establishments from around the world suffered from large-scale data breaches and attacks. Another thing to consider is that due to it being dependent on the internet, it will severely affect bandwidth if it so happens that backups are being made during business hours and therefore additional charges could be applied if that is the case.

2.3. Justifications that shape the design

To build upon and make additions to a legacy system makes it easier to determine limitations with the design and other means that might affect functionality. Legacy systems are often familiar for those who have experience or that were involved with it at one point or another and quite efficient for specific tasks. This in turn also makes it accessible when it comes to implementing the product to the said system since it is essentially mostly adding rather than modifying.

Cloud databases are often preferable when it comes to small-medium-sized establishments considering the reduction of costs. Cloud databases are heavily reliant on the internet and therefore can be quite scalable, elastic, and multi-tenant in nature.

The product being a web application enables a quick and cheap solution for the establishments to consider. That is due to maintenance not being prominent as there is no need to create various versions for different platforms and regular updates are not entirely necessary, the web environment is slowly becoming better since they attract users, utilizes languages that are quite popular, and have extensive support and generally, there are fewer bugs present in web applications.

There are quite a few elements that play a huge role when it comes to design and as mentioned before, the client's requests and general experience with the product will be the main motivators of the design as it aims to aid them and improve the workflow in the establishment as a whole. It is important to note that although there are several design philosophies, decisions, constraints, and justifications mentioned, the design is not confined to them and is to be considered as flexible as requests and environments may change although mentioned aspects serve as a foundation for the product as of now.

3. Design goals and philosophy

3.1. Philosophy of the design

The GymUP system is immune to complex deployment issues. Although our system is a subsystem, we need to deploy very little. The system can build the database it needs by itself. The gym only needs to change the database username and password in the program to use it. For legacy systems, since the system is a subsystem, we consider adapting legacy systems. All the legacy system needs to do is provide a link or button in our system to allow the user to access our website.

As for the performance aspect of the design which is related to the function over form design philosophy aforementioned, since it is only provided to small gyms, the number of customer demands is not large therefore the performance of the hardware (server) does not need to be considered. More should consider the performance of the software (GymUP system). In this way, if more gyms need our services in the future, we only need to update the hardware, and there is no need to rebuild the software.

As for robustness and flexibility, since the legacy gym management system only has the function of booking time or classes, it does not allow members to know whether there will be many people using the same equipment at a specific time. Our system solves this problem by allowing gym members to pre-book the equipment they need and the period for which they will be used. Due to the wide range of users of the system, GymUP may face a fewer variety of errors and requirements, so GymUP needs to be robust and intuitive. Since the system is seen as a module that can be used with various software applications, GymUP should also be reusable. Flexibility is also necessary as more functionality may need to be added to the system in the future.

3.2. System efficiency under unusual circumstances

3.2.1. Offline availability

The system is not required and will not operate offline therefore there is no need to consider efficiency in such circumstances.

3.2.2. Compromised security

The system will have to be shut down for maintenance work until the security issue is resolved which results in the users not being able to log in or just generally not using the system.

3.2.3. Outdated browser

The system can partially function in an outdated browser however the website might not be displayed entirely correctly and in such a case said users with the outdated browser will get a pop-up to update their browser.

3.2.4. Database related maintenance work

Considering that the system revolves around being able to make use of data and process it, a database related maintenance work will result in a synchronized shutdown or present error messages until the maintenance of the database is finished and therefore the users will be able to log in however other functionalities such as booking equipment or canceling bookings will not work as intended.

3.2.5. Lack of general maintenance

Depending on the severity of the lack of maintenance, the system's functionalities might be impacted in a range from being able to run although not efficiently with a few interruptions in addition, to not being able to function at all and therefore the system should be maintained regularly to avoid such circumstances. Overall, efficiency will be difficult to achieve in case of a lack of general maintenance.

4. Assumptions and dependencies

4.1. Application type

This application is intended as a subsystem of the legacy gym system which focuses on gym equipment booking. The legacy gym system is actually the website of the gym which provides the general services and functions for the gym members. The functionality of GymUP indicates it needs to serve both the gym members (the customers) as well as the gym administrators. From a practical point of view, development costs as well as the legacy system, it is considered that by implementing the equipment booking system as a web-based application will contribute to best cost-efficiency performance due to the fact that it can serve clients from different platforms via browsers.

4.2. Service object and user interfaces

Considering the above analysis and practical scenarios, it is assumed that most gym members will access the system via smartphones and mobile devices, therefore the user interface design for the customers needs to focus on small touch-screen usage scenarios. It is assumed that the gym administrators who work on the backend side of this management system will mostly use PCs, therefore the user interface design should be compatible with computer usage scenarios that indicate the information presentation and user interaction need to consider mouse and keyboard.

4.3. Application development

To develop this system, the related frontend frameworks and backend frameworks are assumed to be used:

1. Frontend frameworks:

   For the front-end development of GymUP, it is assumed that the mainstream frontend techniques will be used, in order to realize the dynamic webpage, the HTML, CSS, JS may need to be used. Beyond that, the popular frontend library JQuery could be the potential dependency due to its strong compatibility of event handling, CSS animation, and Ajax.

2. Backend frameworks:

   For backend development, it is assumed that a web-focused framework needs to be utilized in order to achieve efficient object management, business logic classification, code classification, and efficient business layer and persistence layer interaction. The free

open-source framework with a large user base and third-party libraries support will be preferred.

3. Architectural design pattern:

   A close logic connected to the backend framework is an architectural design pattern. The assumption on this part is that the GymUP backend needs to apply a multi-layer architecture pattern, the system will be divided into different layers. The reason for design consideration is that a multi-layer structure can moderate the increasing complexity of modern applications as well as enabling the development team to work in a more agile manner.

4.4. Development resources

During this project development process, it is assumed that the gym will be able to support the GymUP development team in terms of providing contacts with the development team of the legacy system. The gym can also be able to offer the necessary meetings and provide essential other resources for development.

4.5. Software deployment

It is assumed that the gym does not want to own their own servers due to the operation and maintenance cost. Therefore, the application is assumed to be deployed on the cloud servers. Therefore, the state-of-art container techniques may need to be used in order to adapt to the cloud server environment.

4.6. Design pattern analysis

The assumption of GymUP's design pattern is that GymUP will utilize a multi-layer architectural pattern due to the fact that it is the most commonly used industrial standard and it normally integrates well with most back-end frameworks. Implementing a multi-layer design pattern can optimize the structure of the application and reduce the coupling among the different modules. Multi-layer structure enables the development team to separate the complicated business logic to various simple and small parts. It can benefit the development team in the long-term in terms of code modification and maintenance. From this perspective, a top-down design pattern shall be used. On the other hand, due to the development team having many available code resources, namely there are many existing modules from previous projects that can be reused, therefore a bottom-up design strategy may also be

included. Therefore, a mixture of the top-down and bottom-up approaches is assumed to be used.

5. Architecturally significant requirements

5.1. Customer Booking functionality

Requirements:

- **GU.M.FR.1:** Manage (add and cancel) own equipment bookings.
- **GU.M.FR.2:** View all your equipment reservations based on selection criteria (equipment or date).

Rationale: To meet these requirements the program's front end must contain an interface that displays all bookings and include functions and buttons that allow the adding and removal of equipment the front end will be using HTML and the back end must contain ways to connect to the database and present all available information so it can be managed.

5.2. Administrator Functions

Requirements:

- **GU.AD.FR.1:** Manage (add and cancel) any equipment reservation.
- **GU.AD.FR.2:** Manage (add, delete, change state) any equipment.

Rationale: Implement functionality that allows the distinction between customer and administrator so the administrator is allowed to perform the requirements listed, these will require adding extra functionality implemented as well.

5.3. Program Main functionality

Requirements :

- **GU.AP.FR.1:** Book equipment
- **GU.AP.FR.2:** Cancel equipment booking

Rationale: To implement these requirements the usage of Java frameworks will be used, other languages are also valid however the team is most comfortable with java, in this manner, it provides a means to contact the database and so add and remove information as it sees fit.

5.4. Customizable program

Requirements :

- **GU.AP.FR.3:** Displays a list of equipment bookings sorted by time
- **GU.AP.FR.4:** Displays a list of equipment bookings sorted by equipment ID/add equipment

Rationale: To implement these requirements Java frameworks and database indexing will be used, so the information will be primarily indexed by the database and the java application just communicates with the database so it provides the desired outcome.

5.5. Administrator Functions

Requirements:

- **GU.AP.FR.5:** Add and remove equipment

Rationale: In order to implement this requirement we would have to already have a functionality that distinguishes between a customer and an administrator, then create a function with Java that contacts the database so it can add and remove equipment databases.

5.6. Database Functionality

Requirements:

- **GU.AP.FR.7:** Connect to a cloud or local database

Rationale: To implement a database, MySQL database API will be used.

4.8 Internet Functionality

Requirements:

- **GU.AP.FR.8:** Apply REST API to achieve the interaction between the client and the web servers.
- **GU.AP.FR.9:** Using TCP and HTTP communication.

Rationale: To maintain a sustainable server REST API will be implemented, and the basis for taking requests will be TCP and HTTP

6. Decisions, constraints, priorities, and justifications

6.1. Type of application: Client-server (web-application)

Whenever considering the type of application, although there are strong opinions regarding whether it should be a web-based, mobile, or desktop application, oftentimes it is often reliant on the business or establishment and the nature of the application itself [1]. Various questions must be asked whenever choosing the type of application such as

1. What features will be included in the application?
2. What is the budget in regards to the application?
3. What is the schedule in regards to the application?
4. Would there be a monetization strategy involved in the application?
5. Should the users be able to access the application offline?

and so forth, however, the questions above usually are early indicators that give a hint of the type of application that should be used [1].

The features of this application will be related to the bookings of equipment present in the gym for members and allow administrators to manage equipment and managers (or more specifically the members' bookings) alike. The difference between the two is that the most suitable client-server with a thin client application would be a web application that will be adaptable for various platforms such as mobile devices (smartphones) and computers [2] [3].

For members, mobile devices will be more useful considering portability however for administrators, computers will be more efficient considering the added functionalities and elevated privileges (being able to handle management of equipment and members) [1]. Even though there is a distinction between the two actors, the web application can be interchangeable concerning platforms meaning that administrators can also access the app via mobile devices as well. It is to be mentioned that the features will act more as additions to an already established system.

Most of the gyms that will take into consideration the product will be small-medium establishments and therefore budget will be limited. In addition to the budget being limited, the schedule will be tight due to time constraints since small-medium businesses operate in such a way due to the influx of customers. A monetization strategy will be involved although this is related to the business aspect of the system as it will be a subscription-based service. In terms of the inner web application system, no monetization strategy will be involved such as third-party involved billing. The users should not be able to access the application offline due to functionalities being reliant significantly on the internet (cloud database related).

In regards to the answers above and the different points mentioned, a web application would suit the criteria of the establishments interested in implementing this product.

6.1.1. Advantages of web-based applications

Some of the advantages of the application being a web-based one include [3]:

1. **Accessibility and portability** → web-based applications provide a wide range of support for various platforms regardless of the operating system for example [3].
2. **Flexibility** → users can access the application from any location as long as they have an internet connection [3].
3. **Updatability** → web-based applications allow for more manageable updates considering that they are done via servers that do not require any downloading [3].
4. **Deployment** → web-based applications oftentimes do not require any approval from the app stores on Android or Apple-based operating systems and therefore can be launched quickly [3].

There are various advantages related to the general development of the web-based application although these are the primary points to note. As above, so below and such is the case with web-based applications as well as there are disadvantages as well.

6.1.2. Disadvantages of web-based applications

Some of the disadvantages of the application being a web-based one include:

1. **Offline unavailability** → web-based applications do not function offline [3].
2. **Limited accessibility and discoverability** → considering that web-based applications have little to no relation with the mobile devices' app stores, it affects accessibility and discoverability as the application is listed in a database of sorts [3].
3. **Performance** → web-based applications are usually slightly slower than native-oriented applications and have fewer features which affect advancement [3].

There are some disadvantages of web-based applications.

6.1.3. Alternatives of web-based applications

The alternative to web-based applications are native applications (desktop and mobile applications) [4]. The main difference between native and web-based applications is that, as the name implies, native applications are intended for a specific device in comparison to web-based applications. This involves that native applications need to be downloaded from an app store and due to that it also means that it has been approved for usage, therefore, implies that it is safe and compatible, and the functionalities are somewhat integrated with the native device's features, and so forth. However due to the nature of native applications being tailored for a specific device, this is quite costly when it comes to development as there are different languages needed depending on the device the app is being made for, and monetization strategies are more difficult to integrate considering that the app store is also involved to some degree, etc [4].

6.1.4. Justification for web-based applications

The justification for web-based applications is mostly related to it offering a wide range of support for various platforms and that it is quite cost-effective, easier to develop in comparison to native applications, and update ability is easier to work on.

6.2. Architectural pattern: Multi-layer architecture

When it comes to software architecture, it is important to provide customers with a product that is of good quality service and a user-friendly environment. Software architecture serves as the foundation for making the software development process easier as it deals with how a system is structured, behaves, and how it is influenced.

Therefore an investigation was conducted and various questions need to be asked before choosing the architectural pattern considering it is the blueprint of the software and it will impact the software development process significantly. Questions such as

1. Does the application need to be built quickly?
2. Are the teams involved in the process experienced enough developers?
3. What are the conditions regarding performance, maintainability, and testability?

etc, help guide when it comes to selecting the appropriate architecture pattern.

Throughout the investigation, consultation with various sources, and pattern analysis, the most relevant software architecture patterns to the type of application and its nature were: multi-layer, event-driven, microservices, client-server, and MVC architectures [13]. Considering that the application is intended for small-medium businesses or establishments, that the application needs to be built quickly, and the team of developers can be considered inexperienced, the best architecture pattern that is most suitable for the application is the multi-layer architecture pattern [13] [14] [15].

6.2.1. Architectural pattern: Multi-layer architecture

Some of the advantages of the multi-layer architecture include: simplicity considering that a layered structure is both easy to learn and implement which in turn has made it into somewhat of a de facto standard in the industry as it is widely known by developers, experienced and inexperienced alike [13] [14]. The concept of it being layered means that the layers themselves are isolated meaning that changes to one do not apply to the other and that in turn also helps when it comes to overall system consistency, layers also separate concerns in that aspect as well which helps the management of code, etc. There is also the fact that considering the objects are clumped together, it makes browsability easier when a change is necessary [15].

6.2.2. Disadvantages of multi-layer architecture

Some of the disadvantages of the multi-layer architecture include: a lack of scalability as the principles of multi-layer architecture significantly impact the growth and scale of the project as a whole, therefore, requiring organization in order to mitigate this issue, tight coupling between the layers as changes to one layer requires a redeployment of the entire system or application, and each layer has a direct dependency between each other which does not allow for a dependency inversion [13]. Another important point is that perhaps it is too complex in regards to simple operations of creating, reading, updating, and deleting (CRUD) [14] [15].

6.2.3. Alternatives of multi-layer architecture

The alternatives as aforementioned in the previous sections were: event-driven, microservices, client-server, and MVC architectures. For the event-driven architecture, although it is highly compostable, suitable for graphical interfaces, and quite procedural in programming regards, it relies heavily on services being triggered by events which is not ideal for the type of application that is intended for the gym equipment booking [14] [15]. For microservices, it is a collection of services combined to make an application. Its advantages are countless such as: improved performance and productivity, enhanced resiliency, improved scalability, flexibility, etc however as mentioned above, it needs a collection of services in order to truly shine and there is an ever growing complexity associated with the scale of them [14] [15]. Costs are significant and considering that modules deal with data being exchanged between them, it also raises security concerns. This is a viable approach for the future if the system is deemed to expand and have more services, however for the moment, it might be too complex and simply too much for both the application and teams involved. For client-server, it is essentially a two-layer architecture meaning that there is a client and a server [14] [15]. The client usually refers to the front-end and the server mostly the back-end. The server has the resources and the client requests a particular resource. Some of the advantages that client-server architecture has are a centralized server meaning that there is oftentimes a single server hosting the database, improved scalability both horizontally and vertically, and it is quite cost-effective however the downside is that there is a single point of failure as well associated with a centralized server and to sum it up shortly, it is really beneficent for applications that are stand-alone or heavily focus on client-server (say a chat or messenger) relation so to speak which is not entirely the case for the application that is intended [14] [15].

For MVC architecture, it essentially makes a distinction between model, view, and controller with the model being the part of the application that implement the logic of the application’s domain whereas the view is part of

the application that deals with user interactions or more concretely input and output, and finally the controller manages or handles user interaction, communicates with the model, and selects a view to render [16]. There are quite some advantages associated with MVC such as: high cohesion, and low coupling, and that it allows for simultaneous development and interaction; however the disadvantages are related to code navigability and multiple representations being present [16] [17]. MVC is a triangle architecture whereas multi-layer is a linear architecture as it can be seen above and oftentimes is used in tandem with multi-layer architecture where MVC represents the presentation layer and its influence is quite heavy in said layer [17].

6.2.4. Justification for multi-layer architecture

Despite its advantages and disadvantages the multi-layer architecture is the appropriate architecture for the type of application that is aimed for in this project after having weighed the advantages and disadvantages of the alternatives and performing the pattern analysis where multi-layer architecture conforms to all 11 design principles however suffers heavily on overall agility, deployment, performance, scalability although it excels in terms of development and testability.

In the future, if the application is to be upgraded with more features and possibly even migrate to become a system of its own, microservices architecture might be more feasible.

6.3. Cloud database, MySQL, and Docker

Databases can be considered as organized and structured data typically stored electronically [18]. Databases are usually controlled by a management system referred to as a database management system (DBMS), and together they are referred to as simply a database [18].

The alternative to databases are spreadsheets and they are intended more for a single user or a very small establishment. It can be noted by its characteristics therefore a database would be more suitable for this application [18].

When it comes to choosing a database type, there are many options available such as relational database, object-oriented database, open-source database, cloud database, and so forth although the choice for this application is the cloud database [18].

DBMS is quite important as it facilitates key aspects of managing databases such as overseeing and performing CRUD operations [18]. Considering that there are many DBMS options, the one that was chosen was MySQL as it is widely known and used for web applications and is an open relational database management system based on SQL which in turn is a programming language used for all relational databases to manage data. Alternatives include Microsoft Access, Microsoft SQL Server, Oracle database (DB), and so forth [18].

With the important decisions regarding the database out of the way, another question remains: what are the advantages of having a cloud database and how will it work?

A cloud database is essentially a database managed and deployed through a cloud service or more rightfully said a cloud environment [19]. The one that will be chosen for this application is a database as a service that is subscription-based on a contract [19] [20] [21]. There are many advantages to a cloud database such as: lower costs depending on the service provider, improved flexibility as they can be set up and decommissioned with relative ease, and the risks are significantly reduced due to the service provider being responsible for the security of the cloud database and thus providing security updates regularly, and so forth. Some of the disadvantages include: that switching from one service provider to another can prove problematic as different databases support different formats, and there can be downtimes associated with updates and risks which will result in a stall of the system if it happens during business hours [19] [20].

The service provider for the cloud database was chosen to be Amazon and their service is named Amazon Web Services (AWS) or more specifically Amazon RDS for MySQL [22]. The alternative would be for example Azure although AWS is chosen due to lower costs [23].

An alternative to the approach of having physical in-house servers although there are a couple of points to note:

- Physical in-house servers have a limited lifespan as the components do not last forever and need maintaining or replacing [21].
- When having physical in-house servers, maintenance and patching has to be done by the establishment itself and the associated network

infrastructure may need to be accommodated in accordance with it [21].

- Considering the two previous points, additional costs may apply which affect the project's budget [21].

Considering the aforementioned points, it is more feasible for the establishment to opt for a cloud service instead as it has better availability, scalability, and lower operational costs [23]. In case of a change of architecture from multi-layer to microservices in the future, perhaps integrating Docker and switching to Azure as a cloud service would be good as they will be needed in such a case of the switch [23].

6.4. Back-end Frameworks: SpringBoot and MyBatis-Plus

For the back-end of GymUP, the selection for this project is the most popular combination: SpringBoot+Mybatis-Plus. Due to the business nature of GymUP, the frameworks will properly handle the frontend requests, middle layer business logic, and bottom layer data persistence. The two frameworks are widely accepted as very efficient and robust web application frameworks which could largely reduce the complexity of developing projects. Furthermore, all team members of GymUP have worked with the frameworks for many years, and the high demand for utilizing the frameworks indicates the outstanding long-term performance of SpringBoot and MyBatis-Plus. SpringBoot can be considered as an encapsulation version of previous Spring+SpringMVC which does not demand adding many dependencies in the pom.xml file. By implementing SpringBoot, we will encounter less configuration errors and speed up development progress.

6.4.1. Spring

The Spring Framework is one of the most efficient frameworks for managing object instantiation which could effectively save server resources including CPU and RAM. By implementing the important concept "inversion of control (IoC)", the framework efficiently creates the objects, configures and assembles their dependencies, and manages the objects' entire lifecycle. Spring IoC container mainly utilizes "dependency injection (DI)" to realize components management of the application [9].

The top pros of Spring include the following: Spring is a very lightweight framework due to POJO (entity) implementation which means developers do

not need to inherit any classes or implement interfaces; it is an open-source framework including flexible libraries trusted by developers all over the world; The developer can use XML or Java annotations to configure options; due to the DI features, Spring application can easily achieve loose coupling which is very important for object-oriented programming.

The most frequently discussed cons of Spring Framework are the following: the complex nature requires a lot of expertise which means the learning curve is high, therefore it requires development experience; developing a Spring-based application normally requires a lot of XML files which may lead to configuration errors; the last but not least, Spring has a parallel mechanism that provides developers with multiple options which could confuse developer for decision making.

As for now, there are quite a few new frameworks such as Vaadin, Spark, Grails, and Play which can be considered as alternatives to Spring. However, after around 20 years of development since its first release in 2002, the Spring community has become the most dominant framework in the industry. There are tons of libraries available to support this framework, and its performance features in terms of robust and secure are widely acclaimed. Therefore, Spring as the most mature technique is determined to be used [9] [11].

6.4.2. SpringMVC

The Spring Web MVC is an essential framework for developing web applications. This is the second backend framework we selected. SpringMVC provides “Model-View-Controller” architecture and related components that can be used to develop flexible web applications [11] [12]. The MVC pattern enables GymUP developers to separate different aspects of the application such as input logic, business/service logic, UI logic, persistence logic, etc, which will achieve high cohesion and low coupling program [9].

- Model: encapsulating the application data which will consist of Entities/POJOs (Plain Old Java Objects)
- View: presenting the model data to the frontend, it will generate HTML output that clients’ browser can parse
- Controller: processing client’s requests, building an appropriate model, passing it to the view for presentation

As discussed above, once Spring is decided to be used, then SpringMVC becomes the prior choice for web application development in terms of compatibility, functionality, and stability of SpringMVC. SpringMVC has good compatibility with Spring since they come from the same developer and

both frameworks could cooperate efficiently. Similar to Spring, SpringMVC is a flexible lightweight request-driven web development module [9] [11] [12]. It encapsulates Dispatcher Servlet, ModelAndview, and View Resolver which will make web application development easier and more convenient. Same as Spring, SpringMVC uses annotations which could make the development model the same as POJO which will lead to testing of the controller much easier. The main disadvantages of SpringMVC are that SpringMVC and Spring's Servlet API are coupled which means that it is not easy to run independently without using the container. However, in general, it is easy to write programs with SpringMVC and it normally generates excellent performance, making the decision to use this framework a no-brainer [11] [12].

6.4.3. SpringBoot

SpringBoot is an innovative framework that can make developing web applications and microservices with Spring and SpringMVC faster and easier. It is more like an encapsulation of Spring, SpringMVC and other Spring related frameworks. By utilizing the special mechanism "Autoconfiguration", it largely improves the speed of software development as well as reduces the number of dependencies. Furthermore, it is embedded with Tomcat and it can Automatically configure Spring and 3rd party libraries whenever possible. Compared with SS(Spring+SpringMVC), there is no code generation and no requirement for XML configuration in SpringBoot [25].

6.4.4. MyBatis-plus

For the GymUP application persistence layer, MyBatis-plus is selected as an extra bonus framework to simplify the project structure [10]. Mybatis-plus is an upgraded version of the standard Mybatis framework. It will simplify the standard version of MyBatis framework in terms of demanding of writing a lot of SQL mapper.xml files. It even encapsulates many general SQLs so the developers do not need to write. MyBatis-plus is an excellent persistence layer framework that supports custom SQL, stored procedures, and advanced mapping which means GymUP will avoid almost all JDBC codes and related manual parameter settings and retrieval encapsulation of result sets [10] [11]. MyBatis-plus can map interfaces and Java entities to records in the database using simple XML or annotations for configuration and native maps. This enables GymUP to extract a large number of SQL statements in the program and configure them in the separated configuration files to realize a flexible configuration of SQL. The so-called "semi-automatic" ORM implementation will realize the "SQL Mapping " mechanism that will eventually benefit GymUP persistence layer management in the long term [11] [12].

6.4.5. Justification for the back-end frameworks

Considering the combination of SpringBoot and Mybatis-plus are still one of the most popular selections for web application development, the benefits of using them in terms of development time, development cost, and performance are significant. Most of the existing java applications were developed using the above frameworks, they were also chosen for the GymUP project.

6.5. Front-end languages: HTML, CSS, and JS

The front-end languages that will be used in terms of front-end are: hypertext markup language (HTML), cascade style sheets (CSS), and Javascript (JS). Oftentimes, these three languages are used in combination to format, design, and program various web pages [5].

6.5.1. Hyper-text markup language

HTML is used more like a language that utilizes tags to identify various content types such as style, format, structure, and so forth rather than say a programming language like Java that is used more for performing functions so to speak. HTML is used to structure the web page and serves as the "format" language of the three [5].

Some of the advantages of HTML include an easy and a quite simple language, it is considered an industry standard as it is accepted by almost all modern browsers, it is platform-independent, and has a large supporting community that is active, etc [5].

Some of the disadvantages of HTML include: considering that code can be quite large, it can then prove to be quite difficult to handle, security on its own is quite weak and relies on third-party services for it, HTML is often used in combination with other languages as by itself cannot provide much functionality, each web-page has to be programmed separately as it is a "decentralized" code structure, etc. Despite its advantages and disadvantages, HTML has proven to be one of the most important languages in regard to web development as every page on the web contains HTML. For the moment, there is no widely known and accepted alternative to HTML and therefore HTML is a primary choice when it comes to the development of web-page [5].

6.5.2. Cascade style sheets

Cascade style sheets or CSS is the language used to style an HTML document, in other words, it describes how the elements should be displayed therefore CSS can be considered the “design” language of the bunch. It can be used for basic styling, and creating layouts, and can also be used for adding effects such as animation. CSS is a style rule language that applies styles to HTML content, such as setting background colors and fonts and laying out content in multiple columns [5].

Some of the advantages of using CSS include: time efficiency as one change defined in one file will also apply to all of the references of that file, spontaneous and consistent changes considering they are applied consistently throughout the work, not complex, etc [5].

Some of the disadvantages of using CSS include: CSS works differently on different browsers and therefore that can be a cross-browser issue and requires testing beforehand, there are different levels of CSS (CSS, CSS2, CSS3, etc) can be confusing for non-developers, and beginners in general, scarcity of security is also an issue as it is easily accessible due to its nature of being an open-text based system, etc [5].

As of now, as with HTML, there is no known and widely accepted alternative and therefore CSS was the primary choice [5].

6.5.3. JavaScript

JavaScript is a script, a programming language that can implement complex functions on web pages and is considered a core technology for the web. Whenever working with a web page information is needed [5] [6]. What web pages show you nowadays is no longer simple static information, but real-time content updates, interactive maps, 2D/3D animations, scrolling videos, and more [6]. In a technical context, it is a high-level and often just-in-time compiled language that ensures it is compatible with various browsers. JS has many features such as application programming interfaces (APIs), the document object model (DOM), standard data structures, regular expressions, and so forth [5] [6].

Some of the advantages of JS are [7]: due to it being a just-in-time (JIT) compiled language it means that it is quite efficient and fast since it is run with the client’s browser and is not slowed down by calls from a back-end server, its syntax is quite simple and beginner-friendly which makes it relatively easy to learn from a developer’s perspective, there is also a huge supporting community for it as well and there are frequent updates provided for JS which is important in terms of the life cycle of a web-page, etc [6] [7].

Some of the disadvantages for JS are [7]: it may prove difficult when trying to develop large applications, JavaScript code is always visible to everyone and therefore this becomes its main disadvantage as it lacks from a client-side security perspective and can even be exploited for malicious intent (one of the main reasons why disabling JavaScript is done by some), and it is a language that is interpreted differently by different browsers [6] [7].

As of now, the only alternatives are those that compile to JavaScript are the only somewhat realistic ways to even consider alternatives for it and it will likely remain so for a while considering that it is a core component of web development and an industry-standard so to speak. There are many such languages: Dart, Elm, TypeScript, etc although the questions arise [8]. How long will it take for a newcomer to come up to pace with said languages? Do they have too few or too many features and how complex are they? Will other environments be targeted in the future? These are questions that impose risks that are too great for this project and therefore JavaScript was chosen instead as the primary choice in regards to the application [6] [7] [8].

6.5.4. Justification for the front-end languages

Considering that the trio of HTML, CSS, and JS are the main components of the world wide web, they essentially complement each other in terms of design, structure, and programming, and they are languages that are widely used, known and supported, they were chosen for this project and will likely remain as definitive choices [6].

6.6. Mixed of top-down and bottom-up

Nowadays there are three basic design approaches when it comes to software design [24]:

- Bottom up
- Top down
- Mixed approach

and they focus on different aspects of the design.

A bottom-up approach is also known as inductive reasoning when it comes to design where it starts with the most basic and specific components of the system and can be considered as composition [24]. The process continues as then the lower tiered components are identified, they help in composing the higher tiered components of the system and it does so until the system is evolved to be dispersed enough to not be considered as a single component thus increasing the abstraction level. This approach is more suitable when a

system has to be based on an already existing one and said system needs to be scalable and adaptable [24].

Top down approach is also known as a step-wise approach to designing software and is often referred to as the decomposition where a component can be broken down into further sub-components so to speak [24]. The system is considered as one entity according to the top down approach and then it is decomposed into smaller parts and this process continues until the lowest level of the system is achieved. This approach is more appropriate for software development and solutions that need to be worked upon from scratch [24].

In order to make the most of the design approaches, a mixed approach will be used for this project [24]. A mixed approach is the best of both worlds, which results in top down preparation and bottom up response. This means that the approaches are changed depending on the progress, if the entities of the system are found by using top down then the basic elements of said entity are identified by using bottom up approach and thus this continues until a middle-ground has been found between the two approaches [24]. This allows for flexibility and overall, if the design approaches are considered individually, they are not as practical as they would be combined [24].

7. Key abstractions

Key abstractions are defined as a class or object that forms a part of the vocabulary of the domain. In this case, the domain is a gym booking system and therefore the key abstractions are mostly related to members, gym staff, managers or administrators, equipment, and appointments or bookings.

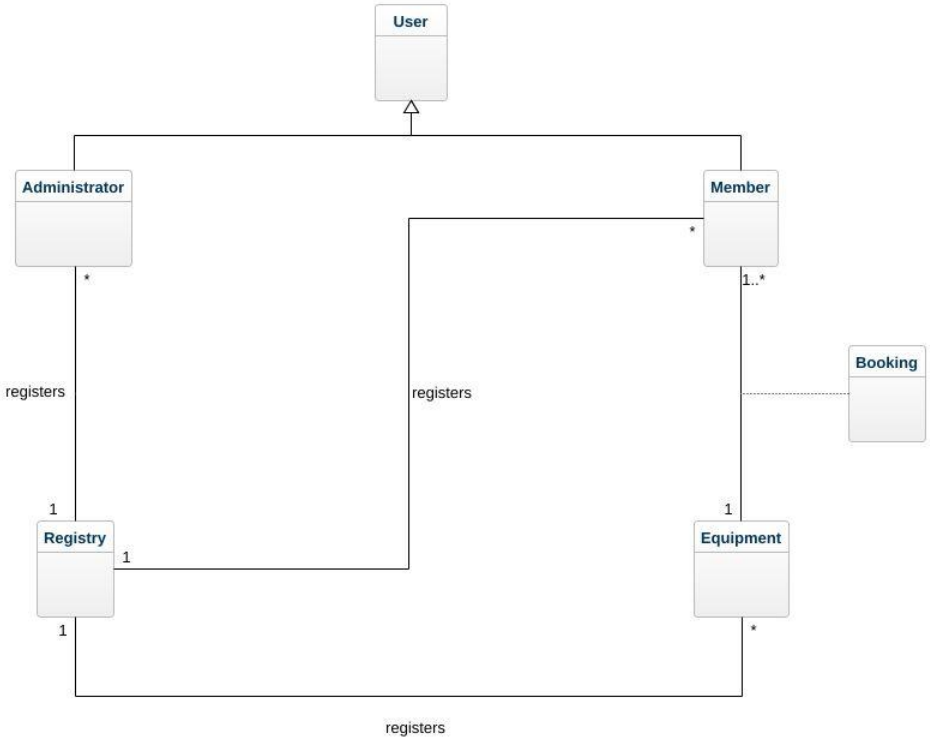


Figure 1. *A very simple domain model of the gym equipment booking system that implies the key abstractions.*

A very basic domain model of the application in which it identifies the main conceptual classes of the system: member, equipment, booking, administrator and registry and the different relations the entities have with each other. In regards to domain modeling, this is quite primitive as it heavily relies on plain associations to portray the relationships; however it serves as a general idea of how the classes will interact with each other. In short, the registry records data of members and equipment, members are associated with equipment through bookings as a member can have many bookings however bookings can only be associated with one member and this indirectly translates to equipment as well. Registry is essentially a database.

1. **Member**

   This is one of the key abstractions of the system due to the fact that the system will revolve around members. Members are customers of the gym that may or may not have a subscription. Members will be associated with bookings and equipment i.e they can have a booking for equipment for the gym. Key information regarding the member would be: their IDs, full names, a membership that they have, bookings associated with them, and so forth.

2. **Administrator**

   Considering the scale of the business is small to medium, this term encapsulates gym staff and managers alike. Administrators will have elevated privileges compared to members i.e administrators will be able to manage members and their bookings, and equipment as well hence they will be associated with the entirety of the key abstractions as they are quite a focal point when it comes to the management of the booking system as a whole.

3. **Equipment**

   The equipment is the key abstraction of the system as it is essentially based upon a member being able to book an appointment for equipment and therefore information will involve equipment at one point or another. Equipment will be identified by various values such as its ID, bookings associated with it, and members that have bookings related to respective equipment.

4. **Booking**

   Bookings are one of the key abstractions of the system as it links members to the equipment with it being an appointed time. Once a booking is established, it is both associated with the member and the equipment as it is the link between the two in the system.

8. Architectural Patterns

8.1. Architectural Pattern: Multi-Layered

A proper architecture is critical to a software system. Multi-layer architecture is a classical architecture pattern, whose layers can have the same functionality but at different levels of abstraction. This is the idea behind the UNIX architecture. Classical four-layer architecture pattern, which has four main logic layers from up to bottom Data Access Layer (DAL), Business Logic Layer (BLL), Controller Layer (CL), and Present Layer (PL).

8.2. Client: Thin Client

Since the program will run in a web browser this does not allow us to make any local changes to the local device so for this reason we have no other choice but to use a thin client server since the other it's impossible to produce without changing the whole architecture, however for the purposes of functionality this client allows more moderation from the server so queries can be accounted easily.

8.3. Multi Layered Layers:

8.3.1. Client Layers

8.3.1.1. Presentation Layer

Requirements: Means to present functionality to the users.

Rationale: This layer is entrusted with anything regarding the UI of the application, this will be used by the client in order to utilize the functionality.

8.3.1.2. Controller Layer

Requirements: Expose the backend API via HTTPS (more secure) endpoints.

Rationale: GymUp is elaborated and discussed in terms of the business architecture that produces and runs it. The controller performs validations on the input from the system and coordinates the necessary steps to fulfill the request. IT service management efforts, as well as consulting broadly across IT operations regarding the process, data, and systems for the business of IT are orchestrated by controllers.

8.3.2. Server Layers

8.3.2.1. DAL Data Access Layer:

Requirements: Provide means of communication between server and databases.

Rationale: This Layer is responsible for establishing communication between the server and the database, this would allow the program to abstract information and set up efficient indexing as well.

8.3.3. Functionality Layers

8.3.3.1. Business Layer

Requirements: Implement business rules and logic.

Rationale: Firstly, business rules examine the software development, the data, and systems in other words it provides functionalities to different layers in this single Layer. The main focus of this layer is more on business configuration and various infrastructure services to make sure that each object in the system interacts with the other. Last but not least, business logic would in this case model IT Assets such as account and equipment inventory.

8.4. Dependencies that may or may not affect the consistency and uniformity of the architecture.

8.4.1. MySQL Database

In order to maintain a database and also have efficient indexing we would require this API, but since this API goes through a moderate amount of maintenance, this would not affect the consistency and uniformity of the architecture of the program.

8.4.2. Mybatis

Allows easier managing of the Data Access Layer, Mybatis works in tandem with SQL and also is an API that receives usual maintenance updates so its security is reinforced, this will not affect the consistency and uniformity of the architecture of the program.

8.4.3. Legacy code

Since we require some functionalities of a legacy code in order to meet the requirements, the legacy code uses static and does not receive updates so the

legacy code that we use in this project will not affect the consistency and uniformity of the architecture of the program.

8.5. How does a multi-layered pattern follow the design principles?

8.5.1. Design Principle 1: Divide and conquer

Yes, the design is addressed: Multi-Layer Pattern follows this design principle because each layer can be worked by a different team of developers and the multi-layer makes it easier to change code without affecting the whole design.

8.5.2. Design Principle 2: Increase cohesion where possible

Yes, the design is addressed: Cohesion is maximized under a multilayer system because each layer it's only interested in working with the relevant code.

8.5.3. Design Principle 3: Reduce coupling

Yes, the design is addressed: Multi-layer approach reduces coupling because the means that different layers will interact with each other will be so that if one layer is affected there would be no issues with other layers apart from the functionality that the layer gave in the first place.

8.5.4. Design Principle 4: Keep the level of abstraction as high as possible

Yes, the design is addressed: Multi-Layer approach allows abstraction by definition because every means of interaction between layers will require a function that does the interaction and these can be configured so it allows abstraction.

8.5.5. Design Principle 5: Increase reusability where possible

Yes, the design is addressed: Multi-layer approach allows an easier time at implementing reusability since all code in a layer is related in some way and so reusing it is easier to implement.

8.5.6. Design Principle 6: Reuse existing designs and code where possible

Yes, the design is addressed and can be reused in some layers and some code can be reused as well.

8.5.7. Design Principle 7: Design for flexibility

Yes, the design is addressed Because if Design Principles 2, 3, 4, 5, and 6 are true then this principle is true as well.

8.5.8. Design Principle 8: Anticipate obsolescence

Yes, the design has addressed many of the APIs we are going to use are reliable for instance: Mybatis has a good reputation and maintenance, and MySQL database is also popular and it is regularly maintained and has effective means of indexing.

8.5.9. Design Principle 9: Design for Portability

Yes, the design is addressed since layered methods allow easier time porting.

8.5.10. Design Principle 10: Design for Testability

Yes, the design is addressed Multi-layer systems allow testing easier because by knowing beforehand what each layer does then testing what that layer is supposed to do is easier.

8.5.11. Design Principle 11: Design defensively

Yes, the design is addressed since means of communication between layers are regulated and so it prevents undesirable outcomes.

9. Architectural views

9.1. Component diagram

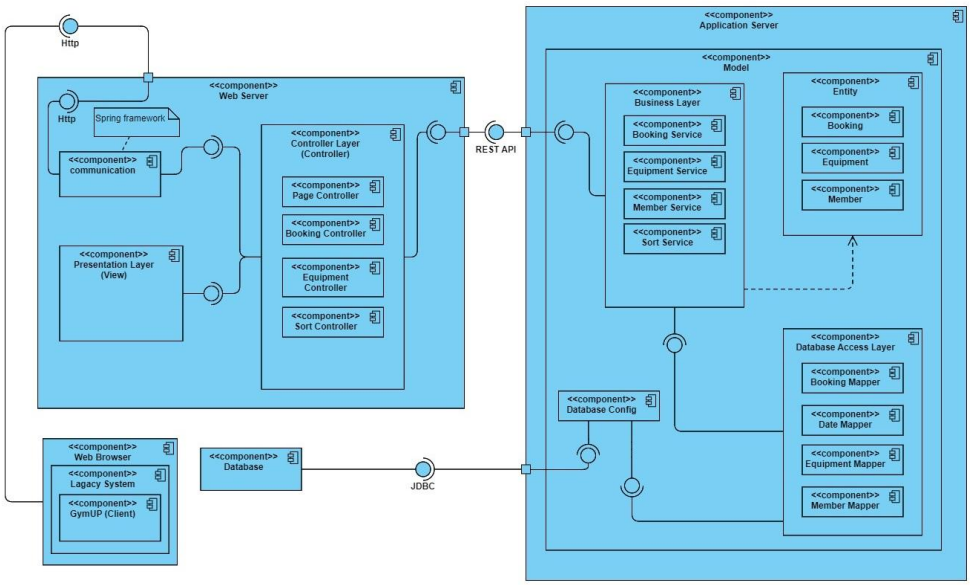


Figure 2. *Component diagram*

This component diagram outlines how the GymUP system components relate to each other. The system serves mobile devices and PCs via a web browser. The system will identify the client and return the web page. By using the Spring Boot framework, the application is divided into different layers. The communication component is responsible for the interaction between the client browser and the system controller. The presentation layer contains the front-end web pages that the controller will call to render the correct content. The controller layer plays one of the most important roles in the overall system, it will coordinate all other system components. The Controller layer will communicate with the business layer, process the business logic and return the results to the controller. The business layer will access the Mapper (database access) layer to get the relevant data of the object. The Mapper layer will access the database to interact with the database.

9.2. Deployment diagram

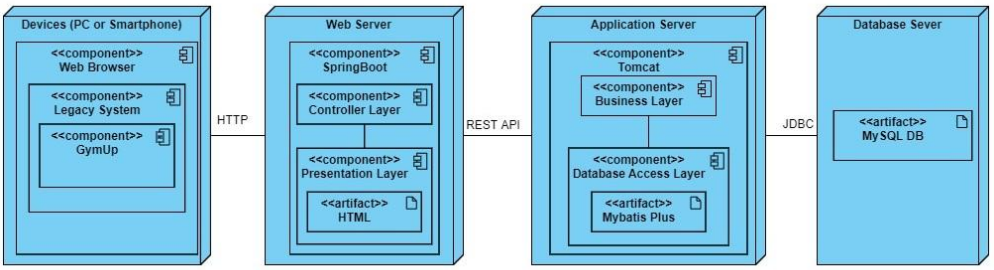


Figure 3. *Deployment diagram*

This deployment diagram provides information on how to deploy the GymUp application system. For all users, all clients are web-based, so users access the application via a web browser. This allows all customers to have the same convenient experience. By using the multi-layer architecture, the system components will be divided into different layers.

The presentation layer and the control layer are included in the Spring framework which is included in the webserver. The **presentation layer** is used to expose functionality to the users. The **controller layer** exposes the backend API via an HTTPS endpoint in terms of security considerations.

The data access layer and the business layer are included in the Tomcat environment and are deployed in the application server. The **data access layer** provides the communication method between the server and the database, which can be implemented more easily with mybatis plus. The **business layer** implements business rules and logic. Finally, **databases** are used to store data.

9.3. Sequence diagrams

9.3.1. Booking equipment

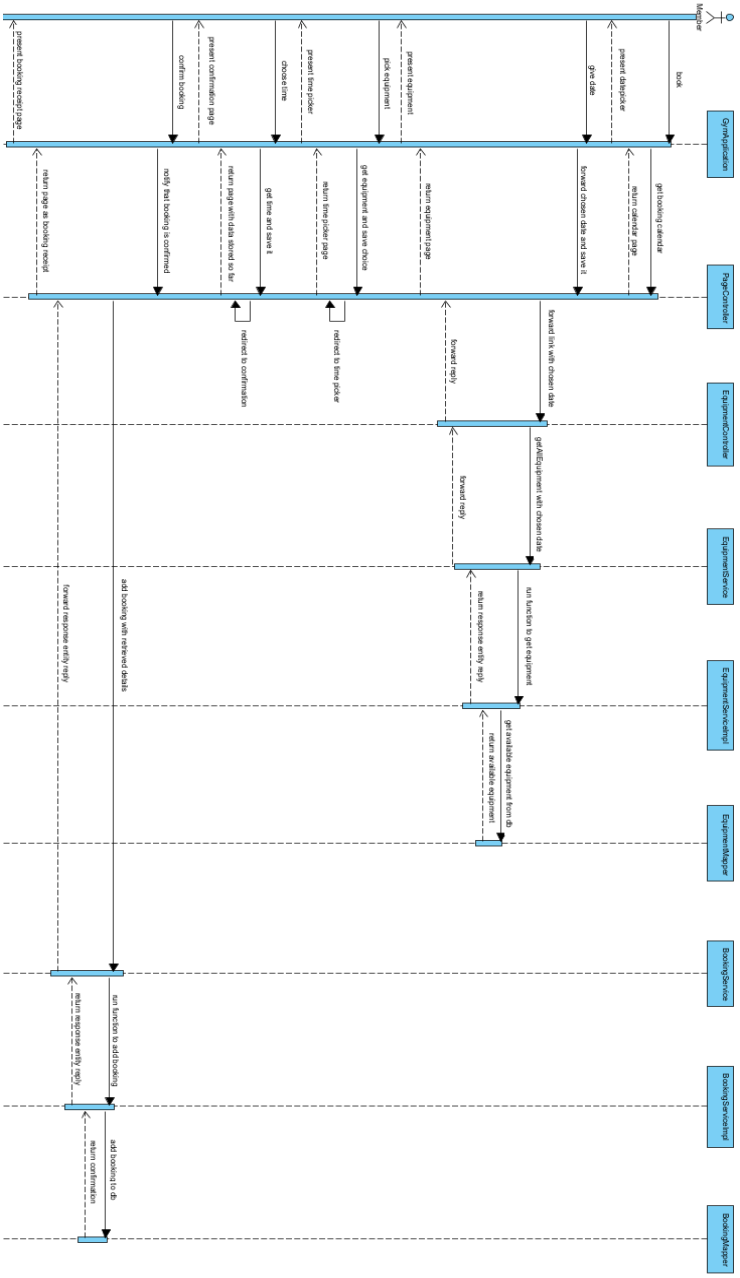


Figure 4. *Customer booking equipment scenario*

The sequence diagram describes the process of a member booking an equipment and the different classes involved when it comes to doing so. It starts by the member choosing the “Book” option in the main menu and then the respective controllers aforementioned in the sequence diagram interact

with each other to redirect the member to the correct page, that being the choosing a date one or datepicker. The datepicker is then presented to the member by the system, and the member chooses the date which they wish to make the booking for. After doing so, the system stores the date in a storage and then forwards the user to the available equipment for that day, and it only shows equipment that the user does not already have a booking for. This can be changed later on however for this example, a member can not have more than one booking per equipment on a day. After the member chooses the booking, then the system stores the chosen equipment and forwards the user to the time-pickers where the members have to choose the times they wish to train on. Once the member has chosen the times desired, the system stores the dates and then forwards the member to a confirmation page where they double check their choices and then the system gathers all of the collected information then adds the booking to the database and returns a confirmation page.

9.3.2. Member checks all of their own bookings

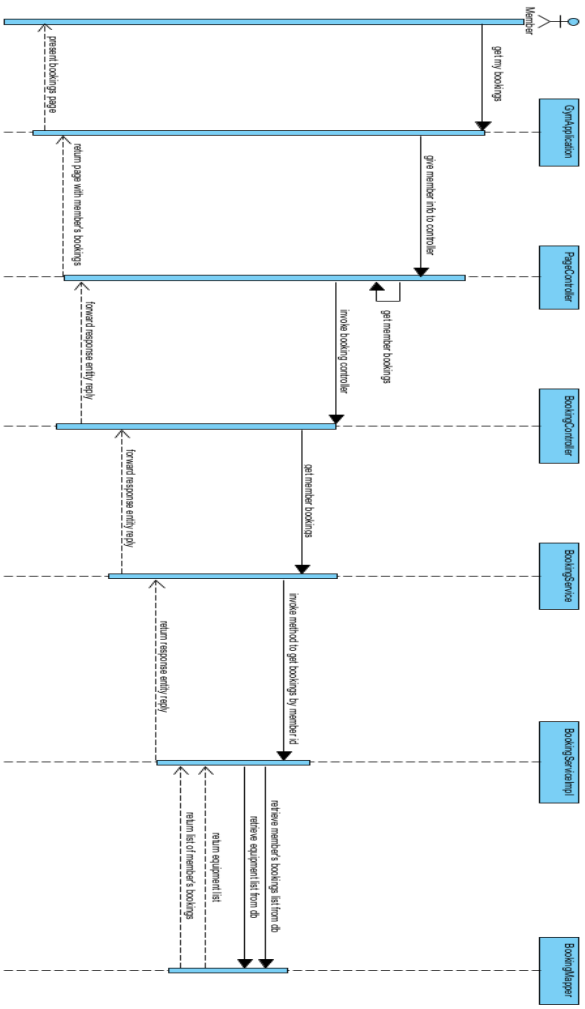


Figure 5. *Member checks their own bookings scenario*

The sequence diagram describes the scenario of a member viewing all of their bookings in the system and how the flow of said process looks like. The process starts with the member being on the main menu and choosing the option to view “All bookings”. After doing so, the system contacts the respective controllers aforementioned in the sequence diagram above by redirecting the member from the main menu to a redirecting page where said page is used as a preliminary check to see which member is logged in and get their bookings based on their member identification. After their member identification is retrieved, it is then used by service implementation to retrieve the bookings associated with that identified member and through a method which intercepts the response of the response entity, the bookings are returned and presented to the member by the application.

10. Performance modeling and analysis

We predict that the system needs to complete 30 execution requests in a minute. The system consists of five service centers: WebServer, MemberServer, AdminServer, Database, and Security.

Each user's request is executed once in the WebServer. The WebServer has a single resource. The response time for each task should be at most 75 milliseconds (0.075 seconds). After the WebServer executes, the request can follow two different paths, depending on whether the requester is a Member or an administrator.

If the requester is a Member, which may happen 90% of the time, the request is executed in MemberServer, where it iterates 3 times. MemberServer has a single resource. In order to make the service as stable as possible, memberServer utilization needs to be 30%.

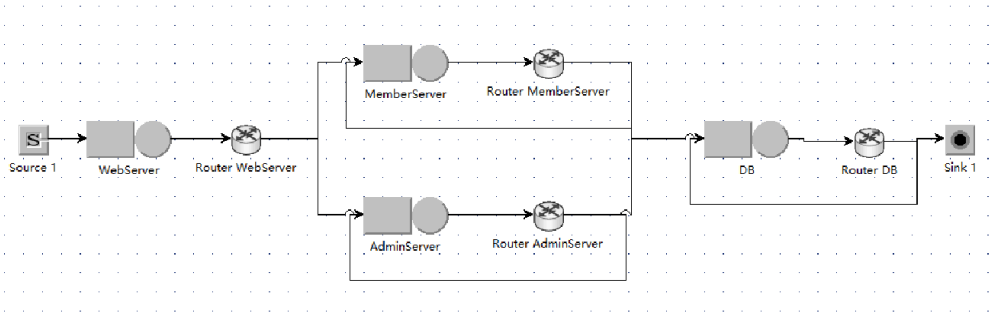
If the requester is an administrator, and the occurrence rate is 10%, the request is executed in the AdminServer, iterating 2 times. AdminServer has a single resource. In order to make the service as stable as possible, the utilization of AdminServer needs to be 0.07%.

Finally, every request (whether it's an admin request or a Member request) needs to perform some calls to the database (change data, get data…). There is a single resource for executing the database.

10.1. Table of calculation result

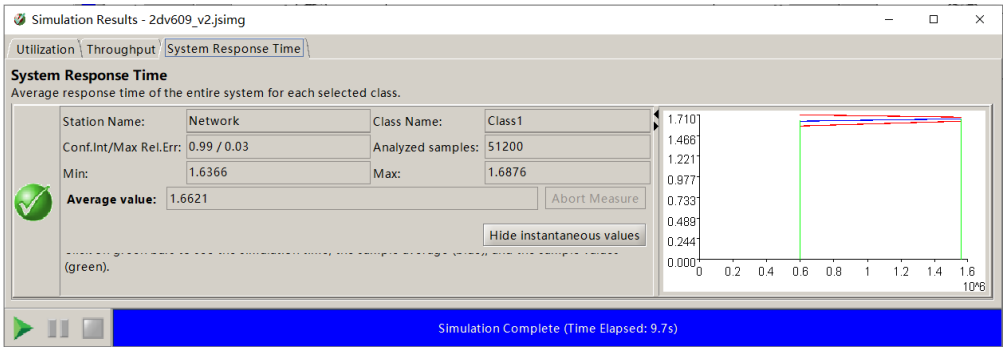
Server	S	X	U
WebServer	0.075	0.5	0.0375
MemberServer	0.2222	1.35	0.3
AdminServer	0.7	0.1	0.07
Database	0.4	1	0.4

10.2. Model



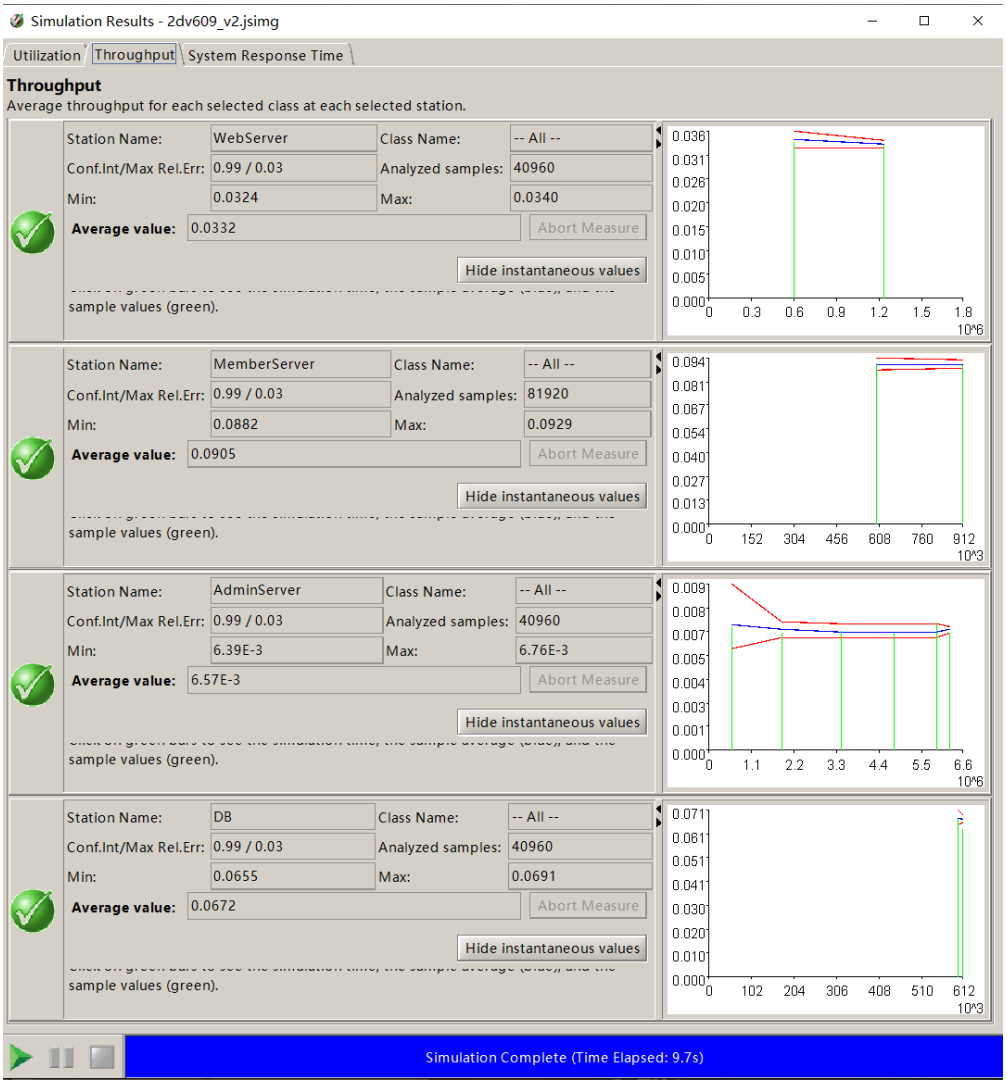
When a task is transferred from the client to the system, it will first pass through the network server. Afterwards, they are assigned to their respective servers according to the permissions of the accounts to perform task execution. After the task is over, it will enter the database for data storage. Finally finish the task.

10.3. System Response time



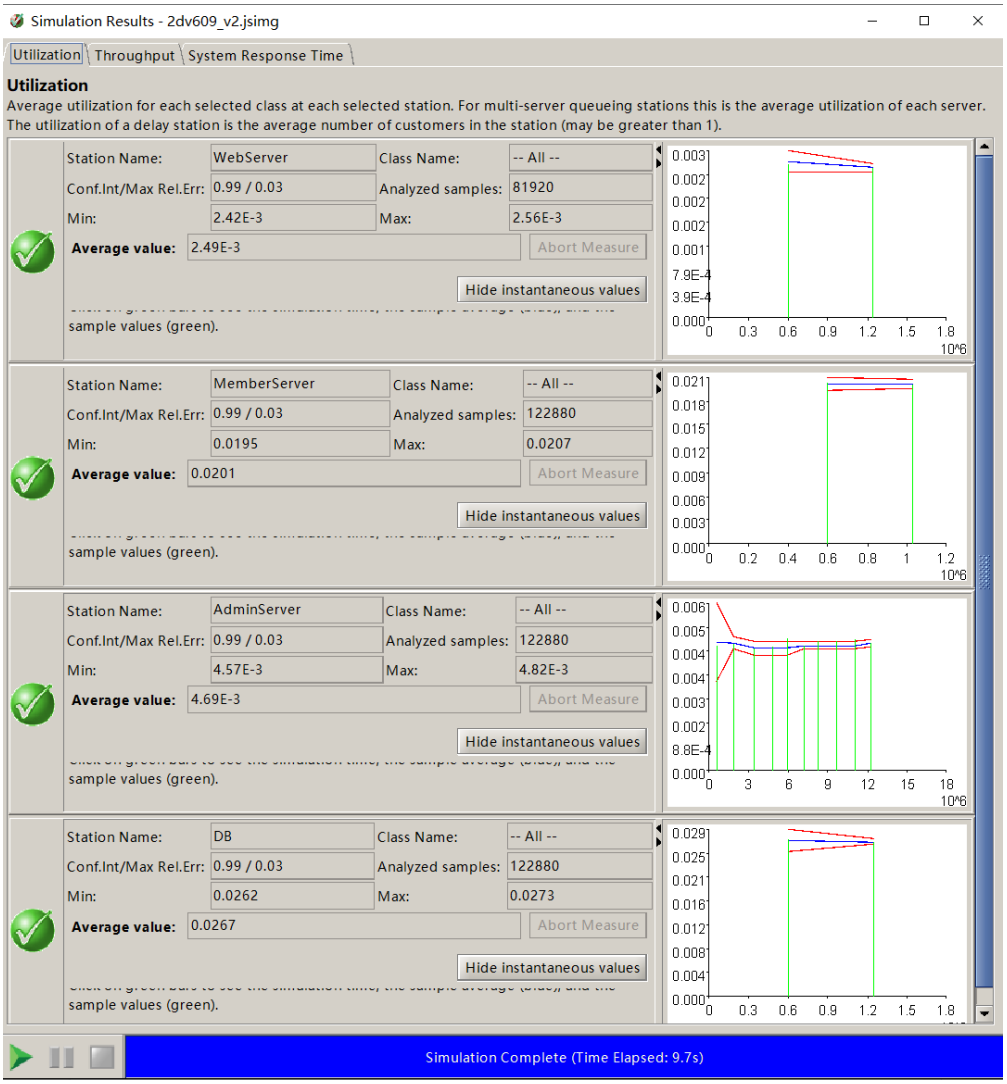
In the simulation, the average system response time is 1.6621 seconds, which is in line with the requirements.

10.4. Throughput



In the simulation, we got all 4 green ticks, which means the simulation passed.

10.5. Utilization



In the simulation, there is no excessive utilization, which means that our structure can handle normal mission demands as well as some sudden visitor growth.

10.6. Discussion

Here are all our NFRs:

- **GU.AP.NFR.1:** The process of booking and giving a response should take no more than 10 seconds.
- **GU.AP.NFR.2:** Utilization of the application and database tiers should be <= 50% for peak workload during normal operations.
- **GU.AP.NFR.3:** The system can handle 30 requests per minute.

According to 9.3 System Response time, the average response time of the system is 1.6621 (maximum 1.6876), which conforms to GU.AP.NFR.1.

From the calculation results obtained in the 9.1 Table of calculation results, we can know that the utilization rate of all components is below 50%, which is in line with GU.AP.NFR.2.

The number of samples used in our simulation is 30 samples per minute, and the above results are obtained. So GU.AP.NFR.3 is also compliant.

So all NFRs are satisfied.

References

1) lvivity.com. n.d. *Web-Based Application: What It Is, and Why You Should Use It*. [online] Available at: <https://lvivity.com/web-based-applications> [Accessed 4 May 2022].
2) Hswsolutions.com. 2022. *Mobile Website vs. Mobile App (Application) – Which is Best for Your Organization?*. [online] Available at: <https://www.hswsolutions.com/services/mobile-web-development/mobile-website-vs-apps/> [Accessed 4 May 2022].
3) Tagneto.org. n.d. *Pros And Cons Of Web Applications – Tagneto*. [online] Available at: <https://tagneto.org/pros-and-cons-of-web-applications/> [Accessed 4 May 2022].
4) lifewire.com. n.d. *Native Apps vs. Web Apps: Which Is a Better Choice for Developers?*. [online] Available at: <https://www.lifewire.com/native-apps-vs-web-apps-2373133> [Accessed 4 May 2022].
5) freeCodeCamp.org. n.d. *Learn Web Development Basics – HTML, CSS, and JavaScript Explained for Beginners*. [online] Available at: <https://www.freecodecamp.org/news/html-css-and-javascript-explained-for-beginners/> [Accessed 4 May 2022].
6) "ECMAScript® 2020 Language Specification". Archived from the original on 2020-05-08. Retrieved 2022-05-04.
7) recro.io. n.d. *Pros and Cons of JavaScript: Choose Wisely - Recro | Blog*. [online] Available at: <https://recro.io/blog/pros-and-cons-of-javascript-choose-wisely/> [Accessed 4 May 2022].
8) stackshare.io. n.d. *What are some alternatives to JavaScript? - StackShare*. [online] Available at: <https://stackshare.io/javascript/alternatives> [Accessed 4 May 2022].
9) Tutorialspoint.com. n.d. *Spring - MVC Framework*. [online] Available at: <https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm> [Accessed 4 May 2022].
10) Mybatis.org. 2021. *mybatis – MyBatis 3 | Introduction*. [online] Available at: <https://mybatis.org/mybatis-3/> [Accessed 4 May 2022].
11) developpaper.com. 2022. *Vigorous construction of Spring + spring MVC + mybatis - Develop Paper*. [online] Available at: <https://developpaper.com/vigorous-construction-of-spring-spring-mvc-mybatis/> [Accessed 4 May 2022].
12) developpaper.com. 2020. *SSM (Spring + SpringMVC + MyBatis) framework integration - Develop Paper*. [online] Available at:

<https://developpaper.com/ssm-spring-springmvc-mybatis-framework-integration/> [Accessed 4 May 2022].
13) geeksforgeeks.org. 2021. *Types of Software Architecture Patterns - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/types-of-software-architecture-patterns/> [Accessed 4 May 2022].
14) Dhaduk, H., 2020. *10 Best Software Architecture Patterns You Must Know About*. [online] www.simform.com. Available at: <https://www.simform.com/blog/software-architecture-patterns/> [Accessed 4 May 2022].
15) Wayner, P., 2021. *5 software architecture patterns: How to make the right choice*. [online] techbeacon.com. Available at: <https://techbeacon.com/app-dev-testing/5-software-architecture-patterns-how-make-right-choice> [Accessed 4 May 2022].
16) Solution, S., 2017. *Why MVC Architecture?*. [online] medium.com. Available at: <https://medium.com/@socraticsol/why-mvc-architecture-e833e28e0c76> [Accessed 4 May 2022].
17) Mburu, D., 2017. *What are the pros and cons of the MVC pattern?*. [online] quora.com. Available at: <https://www.quora.com/What-are-the-pros-and-cons-of-the-MVC-pattern?share=1> [Accessed 4 May 2022].
18) Oracle.com. n.d. *What is a database?*. [online] Available at: <https://www.oracle.com/database/what-is-database/> [Accessed 4 May 2022].
19) Oracle.com. n.d. *What Is a Cloud Database?*. [online] Available at: <https://www.oracle.com/database/what-is-a-cloud-database/> [Accessed 4 May 2022].
20) orageek.com. 2020. *Pros and Cons of Online Cloud Database*. [online] Available at: <https://orageek.com/cloud-database/pros-and-cons-of-online-cloud-database/> [Accessed 4 May 2022].
21) Karwin, B., 2017. *Cloud database server vs physical database*. [online] stackoverflow.com. Available at: <https://stackoverflow.com/questions/47859993/cloud-database-server-vs-physical-database> [Accessed 4 May 2022].
22) Amazon Web Services, Inc. n.d. *Fully Managed Relational Database - Amazon RDS - Amazon Web Services*. [online] Available at: <https://aws.amazon.com/rds/> [Accessed 4 May 2022].
23) AWS Databases: Break Free to Save, a., n.d. *Purpose-Built Databases on AWS | Amazon Web Services*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/products/databases/> [Accessed 4 May 2022].

24) geeksforgeeks.org. 2020. *Difference between Bottom-Up Model and Top-Down Model - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/difference-between-bottom-up-model-and-top-down-model/> [Accessed 4 May 2022].
25) spring.io. *Spring Boo*t. [online] Available at: <https://spring.io/projects/spring-boot>[Accessed 26 May 2022]

Appendix – Time Report

Date	Member	Activity	Time (hours)
2022/04/29	Long Ma	Writing Section 2	2
2022/04/29	Yuyao Duan	Writing Section 3	2
2022/04/29	Fredric Eriksson	Writing Section 1	3
2022/04/29	Li Ang Hu	Writing Section 7	2
2022/04/30	Yuyao Duan	Writing Section 3	5
2022/04/30	Fabian Dacic	Writing Section 1	3
2022/04/30	Fabian Dacic	Writing Section 6	1
2022/05/01	Fredric Eriksson	Writing Section 4	3
2022/05/02	Long Ma	Modify Section 2	1.5
2022/05/02	Fredric Eriksson	Modify Section 7	2
2022/05/02	Li Ang Hu	Modify Section 7	3
2022/05/02	Fabian Dacic	Writing Section 5	5
2022/05/02	Yuyao Duan	Writing Section 5	3
2022/05/02	Yuyao Duan	Review Section 2	0.5
2022/05/02	Long Ma	Writing Section 9	3.5
2022/05/03	Long Ma	Writing Section 9	0.5
2022/05/03	Li Ang Hu	Editing Section 7	3
2022/05/03	Long Ma	Drawing Section 8	5
2022/05/03	Long Ma	Writing Section 8	0.5
2022/05/03	Yuyao Duan	Writing Section 5	4
2022/05/03	Fabian Dacic	Writing Section 5	5
2022/05/03	Yuyao Duan	Co-op with Long Section 8	2.5
2022/05/03	Fredric Eriksson	Writing Section 7/reviewing other sections	4
2022/05/04	Fabian Dacic	Drawing Section 8	3
2022/05/04	Yuyao Duan	Co-op with Fabian Section 8	3
2022/05/04	Fabian Dacic	Modify Section 2	0.5
2022/05/04	Fredric Eriksson	Reviewing other sections	3
2022/05/04	Fabian Dacic	Adding references	1