Software Design Principles and Architectures – a case study

Suraj Mishra

Haaga-Helia University of Applied Sciences
Degree Programme in Business Information Technology
Bachelor's Thesis
2024

Abstract

Author
Suraj Mishra

Degree
Bachelor of Business Administration, Degree Programme in Business Information Technology

Thesis Title
Software Design Principles and Architectures – a case study

Number of pages and appendix pages
47 + 5

Design principles and architectures are decisive factors for developing robust and efficient information systems in this volatile technological environment. This thesis aims to explore the fundamental design principles and architectures used in modern software development. A sample software project has been chosen to study and conduct thorough analysis of applied architecture and design patterns. Based upon the gathered theoretical knowledge and practical implementation from the sample project, analysis of sample project's architecture and design principles has been made and further improvements are suggested.

The research has been carried out by finding in-depth theoretical knowledge from different scientific sources, e-books, articles, reports, and relevant blogs from software professionals. Furthermore, research has been supported by exploring the sample project's architecture and design. This research has been done in an iterative manner by delving into the project's codebase and establishing the relevant theoretical framework.

Development of sustainable information system should follow well defined methodology with various development phases, suitable architecture, and proven design principles. This thesis presents software development methodologies within traditional plan-driven and modern agile dimensions. Various phases of software development cycles are discussed, and architectural characteristics and various design principles has been highlighted. Moreover, main tools and technologies used in sample project has been explored thoroughly along with the project's background information.

During the analysis of sample project's architecture and design principles, considerations has been made by reflecting to the gathered theoretical knowledge. Further improvements to the project's architecture and design have been suggested. The analysis and improvement suggestions further include author's own development experience from the project work.

This thesis showed that, by following suitable architecture and well-practiced design principles, we can develop sustainable, maintainable, efficient, and robust solutions. Furthermore, quality of system can be examined from various angles of architectural characteristics. It is a challenging task to satisfy all the expects of architectural characteristics in an application, however we need to find a balance between application requirements and project's limitations.

Key words
Software Development Methodology, Software Development Life Cycle, Software Design Principle, Architecture

Table of contents

Table of figures

Abbreviations and terms

Here is the list of abbreviations and key terms that is commonly used in this report.

Table 1. List of abbreviations

CORS	Cross Origin Resource Sharing
DBMS	Database Management System
MDN	Mozilla Developer Network
REST	Representational State Transfer
SDLC	Software Development Life Cycle
SRS	Software Requirement Specification
VS Code	Visual Studio Code

Front-end part of application is the program responsible for creating digital interface for users through which they can interact with system resources and access various services offered by the system. It refers to everything what users can see in any application (Coursera 2024).

Back-end part of application is all about the technologies that is required to process the incoming request, handling of business logics and sending responses (Codecademy 2024). Backend provides resources, data, functionalities in response to a request from another program, typically a frontend.

Database is a permanent housing of data, which is a fundamental component of information systems. It is a shared collection of logically related persistent data and description of this data, designed to meet the information needs (Begg & Connolly 2015).

Git is the de facto version control system for software developers developed by Linus Torvalds in 2005. It is amazingly fast, very efficient with large projects and has an incredible branching system, that makes working with many developers from different corners easy and efficient. (Chacon & Straub 2014)

GitHub is a cloud-based platform to store, share, and collaborate with others to write code. When we are uploading our code in GitHub, we are storing it in a git repository, meaning that any changes that we have made are automatically tracked and managed (GitHub 2024).

1 Introduction

The ever-increasing demand of the digitalization of products and services has led to creation of many different software applications. Almost every sector is facilitated and powered by software products nowadays and we have become totally dependent on it. A complete software package is often an integration of many different components and technologies where software engineers play crucial role in designing, developing, and implementing the final solution. In general level, a complete application comprises of a front-end part, back-end part, and database part communicating seamlessly each other via a network protocol. Designing of these different components is a challenging task and one should follow comprehensive design principles and architectures. All developed software projects are often not implemented because of not having sound business case, comprehensive planning, execution, good design principles, architecture, and delivery.

The presence of software in this information-based society is quite widespread beyond the expectation, from home appliances to nuclear submarines. It consists of different programming instructions telling computer how to execute certain tasks. During the early days of software development, programs were created using low-level or assembly languages whereas programs nowadays are written in high-level programming languages which later converted into machine readable code by compiler or interpreter (Yost 2018). Developing great software and implementing it into practice is not only about strong coding and problem-solving skills. It requires great planning, execution, and delivery. Often software projects are done by team, which involves different mind and skillset of people. Smaller projects are easier to develop, implement, and maintain but when project gets larger than it becomes more complicated to manage everything.

This thesis presents different software development phases and methodologies, architectures, good architectural characteristics, and widely practiced design principles. Later, various technology stacks used in the sample project and its overall development has been explored. The main research outcome is presented by analysing the sample project's architecture, design patterns, and development work by reflecting to the gathered information. Based upon the knowledge gained from research and my own experience during the development of project, some improvements to the architecture, design patterns, and overall development work of sample project has been suggested.

1.1 Research background

This thesis topic became my interest of research when I started working in an existing codebase of software project. The development of project was quite smooth as we already had the defined architecture and standard coding patterns in practice. Because of my personal interest, I wanted to learn a bit more on software design principles and architectures and I choose to write this thesis report. I am writing this thesis, by reflecting to the February 2024 code version of project and its improving further.

This report will provide valuable information about software architecture, characteristics, and design principles to interested individuals in software engineering fields. Furthermore, a set of technologies from front-end, back-end and database will be presented from a sample project, which may add value to readers.

1.2 Objectives

The main objective of this thesis is to study and analyse the architecture and design principles of sample project by reflecting to the existing knowledge and widely adapted techniques. By establishing the theoretical knowledge about software development methodologies, phases, design principles, architectures, and exploring the sample project, this thesis aims to find out common answers to the following questions.

How we can

- work in the same software project with different people together?
- make sure that one developer's code can be understood by another developer within the team and even by new colleagues in future?
- refactor the code faster if necessary?
- make debugging faster?
- make sure that our programs work well enough?
- make sure our programs are well maintainable in future?

2 Research plan

To explore the widely adapted design concepts and architecture, a comprehensive literature review is required to gather existing knowledge and insights from scientific sources and scholars. The rapid development of different tools and technologies across industries are changing in a fast speed, however the main foundational tech stacks remain the same. Most practiced and well-established software development methodologies, design patterns, and architectures will be followed for a long period of time, until and unless there is a dramatic technological change. This research will include finding existing information from different technical articles, journals, research papers, e-books, and blogs by software professionals.

In addition to this, I will thoroughly delve into the sample project and perform the overall development analysis. Studying sample project's architecture will help me to understand the used tools and technologies in the project, applied design principles, coding patterns, and identify areas for architectural improvements. The theoretical base will support the comprehensive analysis of sample project to understand the context, requirements, challenges and to identify the role of design principles and architecture in making the sample project successful.

Furthermore, I will perform the evaluation of sample project's architecture and design principles from different angles. In addition to my own experience during the development of project, the reliable information obtained from research will serve as the strong foundation for evaluating the sample project's architecture. This will help me to identify the strength and weaknesses of used architecture and applied design patterns in the project.

Finally, in combination to the literature review, project analysis, and my own experience during the development of this project, I will provide few suggestions for further improvement of project architecture. These suggestions will further demonstrate that the research knowledge has been utilized fully and architecture of project has been thoroughly studied. I strongly believe that I will be able to provide some valuable suggestions for improvements which might be beneficial for future development projects as well.

3 Software development life cycles and methodologies

Software development field is in constant evolution. Business information systems needs to support and adapt the frequently changing customer behaviour with rapidly growing technologies. Throughout the early days of software development to today's complex software engineering, many different methodologies have been introduced and adapted. Software development methodology is a way of managing software projects, typically by addressing on what versions of different technological stack to choose, what processes do we follow, who works on what, how testing is done, how we implement the final product, how we do maintenance and updates (Young 2013). It is a combination of both practices and values.

In practice, the management of software projects differs from organizations to organizations and sometimes even with different projects within the same organization as well. Several methodologies and practices have been evolved to improve the performance, quality, efficiency, collaboration, and overall development process. Selecting an appropriate methodology, tools, and technologies can have a big difference in completing the project within deadline, cost management, client happiness, and overall success (Parra Rosales 2023). Sometimes even by selecting wrong team members can have misfortune to the overall project success. There are several models for software development from traditional plan-driven models to current agile-development models, but here we discuss about most well-known and widely practiced software development methodologies i.e. waterfall and agile in brief.

3.1 Waterfall model

Waterfall is the most common traditional plan-driven model of Software Development Life Cycle (SDLC) in software development. To put it simple, this method is like constructing a building where next phase initiate once previous phase gets completed. Priority is given to the requirement analysis and system design architecture before doing actual development and testing because of difficulty in accommodating changes afterwards.

This model is particularly suitable for software projects that does not involve frequent client interaction. Generally project is well planned and documented however it restricts flexibility and blocks user feedback until the project has been completed. David Young (2013) included in his article about "feature freeze" in which many organizations refuse to alter feature to be included in the given version once actual development work begins. This results in needed features gets pushed to later major versions forcing software users to wait for years. This type of model can be expensive for smaller projects and ventures.

The general overview of waterfall model is shown in Figure 1 below.

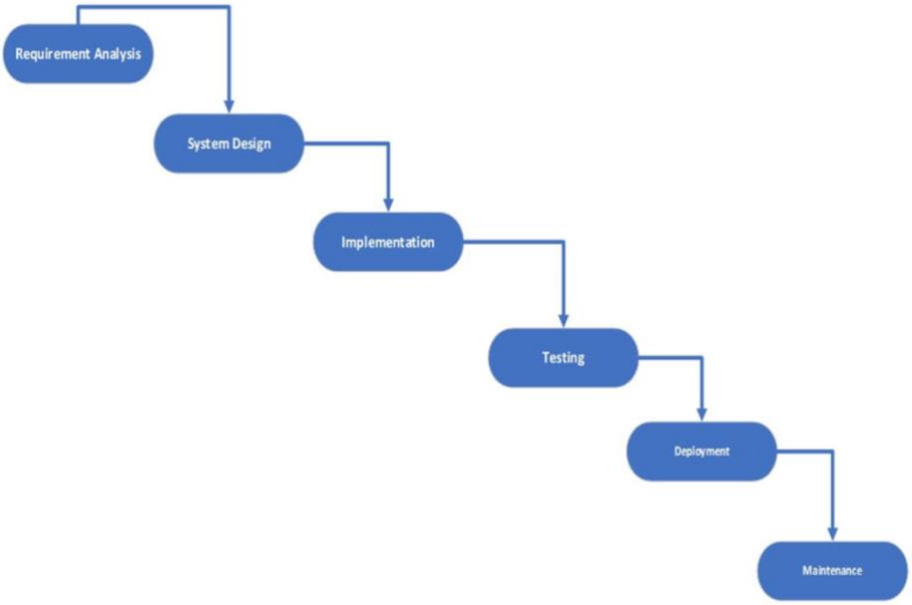


Figure 1. General overview of Waterfall model (Ali Khan 2023)

3.2 Agile model

The term ‘agile’ means able to move quickly and easily. Agile software development is an iterative way of software development that emphasizes collaboration, flexibility, customer satisfaction and early feedback, and adaptive to changing environment, tools, and technologies. It is a broad development methodology, which serves as the foundation for different other methodologies like; Scrum, Kanban, Extreme Programming, Rapid Application Development, Fast-Driven Development, and so on (Risener 2022, 10). It is more of a mindset rather than underlying hard rules on how the development should be done.

The highest priority is given to the customer satisfaction by early and continuous delivery of product. There is enough and frequent communication between development team members, project managers, and business team members. Good communication ensures that client’s requirements are well understood, and they are going to get what they need. This creates a dynamic working environment, building trust between team members, and limits the possibilities of future problems that can be caused by poor communications.

In agile model, changes to the requirements are welcomed, even at the final stage of development and hence provides competitive advantage to customer. It aims to deliver working software frequently and collects customer feedback, thereby improving the final product. The self-organizing development team is further motivated by supportive environment and mutual trust in each other. The principles and values of agile approach of development is defined and mentioned in agile manifesto by several industry experts in 2001 and has been followed widely to improve the final product. The four core values of agile manifesto are mentioned in Figure 2 below.

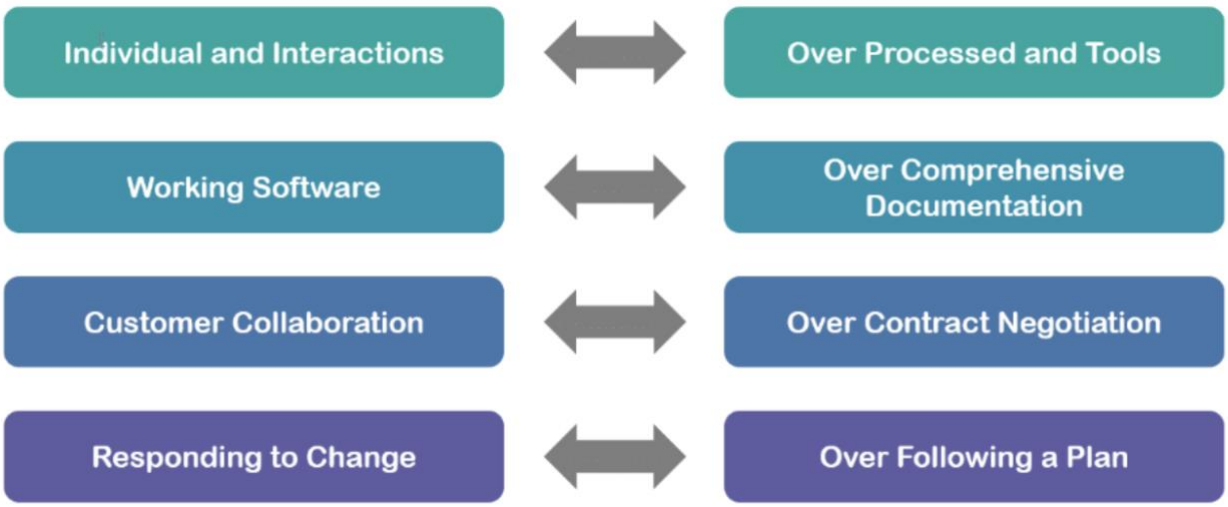


Figure 2. Four core values of agile manifesto (Meyer 2019)

In practice, the project is broken down into smaller parts and developed within different sprints. The development team plans and discusses any issues and makes further improvements in each sprint. Finally, overall project progress is measured by the working software.

3.3 Software development phases

Software development is a complex process of transforming a concept into a functional application. All software projects should follow a suitable methodology with well-defined and structured process for its successful development and implementation. Generally, plan-driven software projects follow structured approach, characterized by well-defined phases and documentation, where each phase starts once previous gets completed. In contrast, projects following agile model are quite iterative, flexible, and adaptive. Regardless of different available development model's nature, they tend to have some similar phases. Software development process contains a complete plan of designing, developing, maintaining, and increasing the efficiency of software (Shylesh 2017, 1).

Different phases of software development cycles are mentioned below and explained in brief by referencing to a blog post written by Sergii Gladun (Gladun 2024).

- Planning

  Planning is the first and most important step in software development once the project concept initiates. The development team performs the requirement analysis for the software and analyses the overall cost. The team collaborates to understand the customer's need and goal of software. The quality assurance, evaluation of technical and financial challenges, risk identification, and sub-plans to mitigate those risks for the software is also performed at this stage of development (Team Split 2023). Finally, a requirement specification document is created in this phase.

- Requirement analysis

  With the completion of planning phase, next step is to define the requirements precisely in Software Requirement Specification (SRS) document, which includes all the product requirement for SDLC and present it to client and stakeholders for an approval (Gladun 2024). A well documentation is made for all approved requirements and any later changes are constantly updated in the documentation in an iterative manner. It will serve as the blueprint for development which provides guidelines in each step, ensuring the purpose of software effectively and efficiently (Krysik 2023). After carefully considering all relevant factors, project manager builds a development team that can implement all the requirements.

- Software design

  Based on SRS document, developers choose and design the most optimized design patterns and architectures for the software and a design specification document is prepared. Depending upon the defined requirements, high-level design and low-level design can be prepared by software architects and senior developers respectively. (Gladun 2024.) Designs can be modified later; however, it is more cost effective, efficient, and convenient if we can develop a suitable design from the start (Koper 2024). This phase may also include developing prototype of product to compare different versions. After precisely considering all possible attributes, the most suitable design is selected for development.

- Software development

  The actual coding of software happens by developers in this phase by following the design principles and architectures mentioned in design specification document. This phase will take significant proportion of overall project (Koper 2024). Different programming tools and technologies, programming languages, are used to develop the most optimized and best software. The development team transforms requirements into code by dividing them down into smaller, modular components and later integrates them together. They experiment the developed product with system design. The goal is to develop a testable and functional software at the end of this phase.

- Software testing

  Even though this phase comes only after software development phase, in practice both coding and testing are carried out simultaneously to avoid higher testing cost (Niculae 2017). In this phase, all the technical requirement of the product is tested. The developed software is tested with multiple test cases to ensure the smooth execution of all functionalities. Software testers will implement different test cases to find any bugs and errors in the programs. Any detected bugs, errors and flaws of the software are then fixed by developers to meet the requirements. Usability and acceptance testing is also performed in this phase with different users and stakeholders respectively, to ensure user satisfaction and positive user experience. The testing team considers all feedback during usability and acceptance testing because it can have great impact in the overall quality assurance of final product.

- Deployment

  Once the software testing is done, finally it is ready for use. The software is installed in the production environment as per specification. If the development of product is an existing application, then possible data migration should also be carried out, which can be an extensive process. Depending upon the business case, the application might need to function during the deployment time as well. (Learntek 2019) Different kinds of configuration management settings and trainings to use the new system is carried out. The development team also performs post-deployment testing to ensure that the software is working fully in production environment (Gladun 2024). Finally, consumers can start using the software product.

- Maintenance and support

  The maintenance phase starts once the software product is in full operation. Often consumers may encounter different unexpected problems with software and a fix is needed. Sometimes a new feature is needed or integration to other systems is required. Time and often, software might need to be upgraded to the latest system version. (Gladun 2024.) Hence, we need to tackle these kinds of issues by offering support and maintenance of software with upgrades and bug fixes. Maintenance and support are necessary to continuously improve the software and it lasts if the software is officially in use (Niculae 2017).

4 Software architecture

Architecture is a big picture of a system which serves as the strong foundation for its development. The precise definition of software architecture can vary from people to people, some says that it is a blueprint of system while others define it as a roadmap to development. It is the collection of tools, technologies, and processes that aims to improve the overall quality of code and development by creating a reliable, sustainable, efficient, scalable, and maintainable workflow (Godbolt 2016, 9). It is critical part of development phase where high-level structure and organization of system is designed, which determines how the system will be implemented and how it will behave.

The architecture of system can be defined in many dimensions. The structure of overall system can be used to define the architectural style, whereas architecture characteristics are other dimensions to define success of system. Architecture decision is another factor that defines the constraints of system. It sets rules for how the system should be constructed and direct the development team on what is and what is not allowed to develop. Another important factor to define architecture is design principles, which is a guideline for constructing a system. (Richard & Ford 2020, 4–8.)

In architecture, we focus more on structure rather than implementation details. Excellent architecture design makes system maintainable, efficient, reliable, scalable and be admired for other development projects. In the architecture design, we need to anticipate expensive choices early enough, that are costly, difficult, and time consuming to change once they are implemented. Defining architecture includes what kind of tools and technologies that we use to develop the product, architecture type, design patterns, development team, system requirements, restrictions, limitations and so on. After getting the overall context, knowing all the things that the system needs to do, how it should behave and what restrictions are in place that needs to be taken into consideration, we need to prioritize them. There might be a conflict between requirements and restrictions or limitations. The designing of architecture should be done by taking one important thing at a time.

It is quite common that developers start writing code without a formal architecture having no clear vision or direction. Determining architectural characteristics of software application becomes difficult if it is not designed by following architecture style. The architectural characteristics of any application can be supported by existing defined structures of different architectural styles. They support the whole designing and development process also by facilitating communication between development team and project members.

It is difficult to maintain all architectural characteristics in an application due to various restrictions and limitations. Some architecture style naturally guides towards highly scalable system, whereas other can guide towards highly efficient system. Knowing the strength and weakness of each architecture style is important to meet application requirements. There are different architecture styles available, which can be classified into two main categories: monolithic architecture (single deployment unit) and distributed architecture (multiple deployment unit). Distributed architectures support more architectural characteristics than that of monolithic architectures. (Richards 2022, 5–7) Let's now discuss about these two categories in brief.

4.1 Monolithic architecture

Monolithic architecture follows a traditional model where all business logics are implemented in a single codebase. It is much simpler, easier to design, implement and can deploy quickly. Cost and simplicity are the strong points of monolithic architecture. With a single codebase, end-to-end testing can be performed faster and debugging becomes easy by following requests and track issues. Easy setup and configuration are some other advantages of monolithic architecture. Besides such advantages, it has poor operational characteristics. If something fails in the application, it causes to fail the entire application and it has longer recovery time (Harris 2024). Updating applications also becomes time-consuming and restrictive as changes requires updating entire stack. This type of architecture is suitable for small and simple applications and websites with tight budget and time constraints (Panchal 2022). Layered architecture, microkernel architecture, pipeline architecture are some of the examples of monolithic architectures.

4.2 Distributed architecture

In this type of architecture, multiple deployment units are working together to perform business logics. It supports strong operational characteristics in applications. As applications has multiple deployment units, if one service fails, another service will continue, and system runs without stopping. System users sometimes doesn't even notice that failure has occurred in the system, as the recovery time of failed services is quite fast. Change applicability is quite fast and easy as different application functionalities are divided into multiple units. Reliability can become an issue in this type of architecture, as different units need to communicate with each other via network protocol and network is not fully reliable. Debugging, managing workflow and consistency can become difficult with this type of architecture style. Besides having a lot of strong non-functional characteristics, applications with distributed architecture are expensive to implement and perform maintenance activities. Event-driven architecture, service-based architecture, micro-service architecture are some of the examples of distributed architecture styles. (Richards 2022, 7–8.)

4.3 Good architectural characteristics

Architecture characteristics are non-functional requirements of an application that defines structure, behavior, and several quality attributes (Mete 2021). They are the core elements of good information systems and software development. There are many different terminologies available for specifying, measuring, and evaluating architectural characteristics. However, it is difficult to fulfill all of them in an application, so the goal is to design an architecture that is suitable for a system by balancing requirements, restrictions, and limitations. Here we discuss the most important architectural characteristics in brief.

- Scalability

  Scalability of systems is the ability to handle increasing workloads without sacrificing performance (Riyani 2023). Scalable architecture naturally supports higher workloads. There can be numerous concurrent user transactions or workloads in the system, and it should handle those conditions without sacrificing performance and stability. This can be managed by allocating enough resources to the system.

- Reliability

  A reliable system performs consistently and predictively under various conditions without having unexpected system errors or crashes. The development of system is often done in multiple deployment units and communication between different units can encounter disruption due to network failure or even database malfunction or system failure (Ashanin 2018). These issues can lead to system become unreliable. Different error handling mechanisms and automatic testing can be implemented to insure reliability of an application.

- Performance efficiency

  Performance efficiency is about meeting both functional and non-functional requirement of system. It is the performance measurement of how efficiently a specific function or component of a system completes certain task in terms of speed, effectiveness, and resource utilization. As for example, performance efficiency of a system can be measured in response time and handling large amount of data without having any issues. (Kokhan 2023.)

- Security

  The system should handle data in a safe manner. User authentication and authorization to services should be handled properly. The input data should be validated with strong validation rules to avoid data inconsistency in database. Maintaining confidentiality and integrity is very crucial and applies to any systems to measure its security. By prioritizing security, we can maximize the protection of application from evolving threats (Watkins 2024).

- Maintainability

  The system should be possible to maintain by future team members, should be able to add new features without having difficulties, should be able to test, and should be possible to update to changing environment. A well written source code divided into modular components can well support this characteristic. Maintainability should be considered carefully in long running applications rather than short time applications because of some cost overhead (Alexandrian 2019).

- Extensibility

  Software systems should be extensible to new features. In this rapidly changing environment, requirements of system might need to be updated or changed. A new feature to the system should be able to develop without affecting other functionalities or components. Similarly, integration with other tools and technologies should also be made possible as this measures the ability to extend a system and implementation effort. (Sharma 2022.)

- Testability

  Each functionality developed in the system should be well tested. Testability refers to the degree to which the functionality, module or component can be tested effectively. This helps in identifying any defects in functionalities by verifying with the system requirements. Different unit tests, integration tests, end-to-end tests, automatic tests can be applied to measure testability of a system.

- Observability

  Observability is the ability to understand the internal state of system by observing the external outputs produced by system (Mia-Platform 2023). This is particularly important in debugging, tracking issues, understanding data flow, and maintenance. Different log messages and event data of system help developers to understand and analyze problems and facilitate fast troubleshooting.

5 Software design principles

Software design is the next step in SDLC after defining the architecture of system. It mainly concentrates on code-level design on how to solve the architectural requirements. Designing great software is a complex process and there are no straight formulas to accomplish it. While excellent programming skills are essential ingredient for developing great software, but they alone are not sufficient. On the other hand, design principles provide solid foundation for developers in writing code that are easy to understand, debug, and maintain. Generally, developers spend less time in actual coding and much time in reading others code, maintaining, and updating it to the new requirements and technologies.

Software design principles are the collection of guidelines, best practices across industries and proven concepts that facilitates the development process in creating well-structured, maintainable, scalable, and efficient systems (Ahmad 2023). They are the most critical ingredient that ensures the quality of system. A good design satisfies software requirements, functionalities and fulfills architectural characteristics if it is followed carefully by the overall development team. Design outcomes are affected not only by designer's cognitive strength but also by development team culture and environment (Hu 2023, 1–3). Design principles will guide us to break down a complex problem into modular structures, following good coding styles, documentation and so on, and hence the coding, debugging, and maintenance is smooth throughout the system lifecycle.

Designing a system is own's thinking process and it may vary from people to people, but following some proven design principles will surely keep us on the right track in creating healthy system. There are many different software design principles available depending upon the context of design but here we will discuss about the most established ones in brief.

5.1 Modularity

Modularity is one of the fundamental design principles in software engineering. It is the principle of breaking down a complex problem into smaller but manageable modules encapsulating specific functionalities and which operates independently (Code Reliant 2023). The degree of modularity depends upon the size and complexity of systems. Generally, smaller size applications have less modular structures compared to large size applications. Modular systems are easier to understand, maintain, and have high reusability.

5.2 Separation of concerns

Separation of concerns is about separating a system into distinct sections, each section responsible for separate concern. It reduces the complexity of system development. Each section of a system plays a meaningful role to develop a well-organized system while maximizing the changeability (Natesan 2019). The responsibility of each software parts should be unique. This enables us to develop new features or make improvements in existing parts without affecting others. Adhering to this design principle, we can develop maintainable, extensible, changeable, and scalable system by improving the overall quality of software.

5.3 SOLID principles

First assembled by Robert C. Martin and later arranged by Michael Feathers, SOLID principles are set of guidelines for designing better software (Krishna 2023). They follow the Object-Oriented Programing (OOP) principles. Robert C. Martin (2014, 95–135) has described about SOLID principles in his book titled 'Agile Software Development, Principles, Patterns, and Practices', which are mentioned below and described in brief.

- Single Responsibility Principle (SRP)

  This principle suggests that each class, interface, module in the program should have only one reason for a change. If a class, interface, or module assumes more than one responsibility, then there will be more than one reason for it to change. Changes to one responsibility may harm others.

- Open-Closed Principle (OCP)

  This principle suggests that each software entities classes, interfaces, modules should be open for extension but closed for modifications. When we change the implemented code in some modules, it can have effect on dependent modules. So, the new functionality should be implemented by adding new code instead of modifying existing code in any entities.

- Liskov Substitution Principle (LSP)

  This principle states that any instance of derived classes should be substitutable for its base class, without affecting the program logics. This principle ensures the consistency and predictability behavior of program.

- Interface Segregation Principle (ISP)

  This principle suggests not to create fat interfaces. Fat interface here means, interface containing many different methods of program. Instead, we need to break down into many smaller and focused interfaces based upon use cases.

- Dependency Inversion Principle (DIP)

  This principle states that high-level modules should not depend upon low-level modules. Instead, both should depend upon abstraction and later detail implementation should depend upon abstract classes methods. This ensures the modularity of program and helps to maintain the code.

(Martin 2014, 95–135.)

5.4 Keep It Simple, Stupid (KISS)

It is very important that, one developer's code can be understood by another. For this reason, we need to write as simple code as possible. This doesn't mean that we need to oversimply or ignore essential requirements, instead finding the simplest solution as possible (Swimm Team 2024). By having simple design and coding practice, developers can reduce the chances of errors. It also ensures maintainability and make it easier for other developers to contribute to the same codebase. This principle doesn't only benefit developers and product team. When we create something that is simple to use, then it is more likely to be adopted by target customers as well.

5.5 You Aren't Gonna Need It (YAGNI)

This principle strongly suggests not to develop a feature that you are not going to need it. Always implement features when you need them, never when you just expect that you may need them (Fernandes 2020). By doing this, developer can focus on the current requirement more precisely. This also reduces the overall system development time and hence the cost. On the other hand, developing unnecessary features adds more complexity to the program and increases bugs. So, development team don't need to anticipate and accommodate future potential requirements while developing a system instead, they should focus on implementing the simplest solution which satisfies the customer requirements.

5.6 Don’t Repeat Yourself (DRY)

Developers should not write repetitive code here and there. Duplication of code makes inconsistency across program and creates difficulty in maintaining it. This principle solves the core problem of knowledge duplication (Plutora 2022). Instead, we need to write code once and reuse them as needed. This principle suggests us to write reusable code that can be scalable as well. To obey this principle, we need to implement our code by using abstractions, modularization, and reusable components. This principle helps in developing the maintainable and scalable systems by focusing on the quality of program code.

5.7 Don’t reinvent the wheel

When developing any system, the development team should do a thorough research about the tools, technologies, and methods needed to accomplish the project. If something exists already, then we should not spend our time in developing that solution. Instead, we need to utilize it in our project. It is the smart choice to take advantage of full-fledged and well-documented solution instead of creating on your own (Heath 2011). By doing so, the development team can focus on other aspects of system that can ensure more quality. On the other hand, by following this principle we can shorten the development time and hence influence the overall project cost as well.

6 Architecture and design principles of the sample project

A full stack web application has been chosen as the sample project to study architecture and design principles. This project has been developed by more than 40 software project course participants at Haaga-Helia University of Applied Sciences in four different semesters. The development of full application has been divided into three different parts, front-end, back-end, and database, integrating different tools and technologies. This project is developed using scrum methodology which is a framework within which we can employ various processes and techniques (Schwaber & Sutherland 2017).

6.1 Sample project background information

The project has been developed for Sibelius Academy, a music institute offering wide range of music programs to various levels of education and comprehensive trainings in different genres of music (Wikipedia 2024). They needed an application to calculate and optimize teaching space and equipment usage for different lessons and music trainings across different program units. I became part of this project in the midway through its development where the Minimum Viable Product (MVP) had already been prepared but not to the final stage meeting all the defined requirements.

An overview of main technological stack used is this project is illustrated in Figure 3. Later, those used technologies in the project will be described in more details in corresponding chapters. The code of the program is written in Visual Studio Code (VS Code), a source-code editor tool. Seamless development of program was facilitated by the supportive features of VS Code, including debugging, syntax highlighting, automatic code completion, code refactoring, and embedded git.

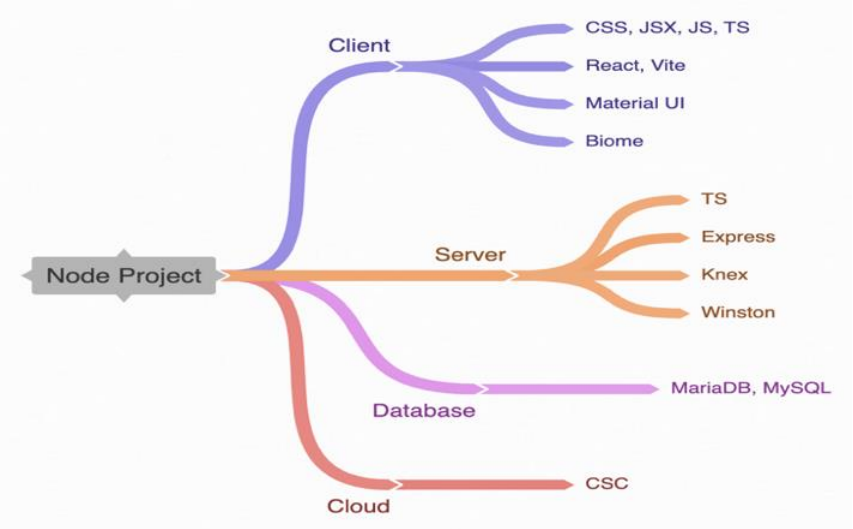


Figure 3. Technological stacks employed throughout sample project development

6.2 Development team collaboration, communication, and project management

In this project we have used Git and GitHub as version control system and source code repository platform respectively. In software development projects, collaboration work within team plays an important role in the success of project. Often in software projects, the use of version management control system is complimentary. They track changes to a file or set of files over time so that recalling to a specific version later is possible. With the use of version control systems, we can see and compare changes, see who modified last and when, who introduced bug and when, backup and disaster recovery and so on.

The communication regarding development of project work has been done mainly through Team's channel. Trello was used as the master project management tool which facilitated the task creation and progress monitoring process.

6.3 Front-end architecture and design

In this project, a web application has been developed which is accessible through browsers. The initial user view of the application is shown in Figure 4 below.

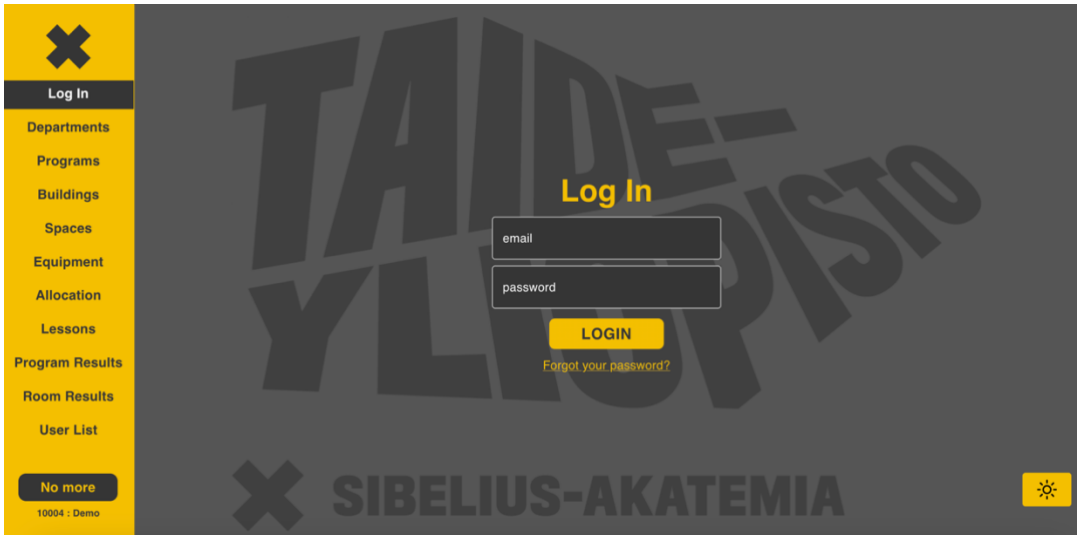


Figure 4. Login page of application

Several technologies have been used to produce the above view for user. Under the hood, the structure and content of the web page is created with HTML which is assisted with CSS and JavaScript. CSS defines the style and layout of web page, whereas JavaScript, as a scripting language, defines the functionality of each components in the web page. React is used as a frontend library to create user interfaces based upon functional components created in this project. The design of user interfaces is implemented with Material-UI, a react component library, which has made

the design more intuitive, user friendly, and responsive. Material-UI has streamlined the styling process by offering wide range of comprehensive customizable built-in components. The navigation in the application is created using react router, a library for handling navigation and routing by enabling dynamic rendering of each components based upon URLs.

In addition to these, the needed functionalities in the frontend part has been facilitated with different libraries, which has made the development faster. The npm has been used to install and manage different libraries in the project, which is included as a standard package manager in node project itself. Different other dependency libraries used in the front-end is included in Appendix 1, which is the content of package.json file. It is the core and crucial file in node project which includes metadata, scripts, modules, packages, and other essential information about the project (Gabby T & Marian Villa 2022). Some custom scripts are also written in package.json file to automate common tasks and starting the development process and thus ensuring consistency across the development environment.

Decisions on what technologies, tools and processes we follow to develop the product is quite crucial since most of the times, later changes are difficult to adjust, or it can become too expensive to make architectural and design changes. Let's now explore about the front-end design patterns, organization of code, and structure of programs.

6.3.1 Simplifying complexity

Looking at the final product, the development of user interfaces would have been very difficult if it was developed at once and in a single place. The overall complex interface has been broken down into smaller but manageable and reusable modular components. Below is the code from the main.jsx file which is the source of script in index.html file in the project. React will render the App component in the web page using the html element with 'root' id. The main starting file of the project is as simple as it can be, hiding several other components and views, styles, programming logics, state management, communication with server and so on behind the scenes.

```
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

6.3.2 Component based architecture

The organization of front-end code can be seen from Appendix 2, where overall development work has been divided into different respective sub-divisions. The smooth navigation has been handled

with react router's components where different views are loaded dynamically based upon user interaction and state changes, which is handled within a single Nav component. The routes of Nav component are mentioned in Appendix 3. Each view is designed to handle distinct tasks and has often integrated many different components. Here, the project adopts the naming convention of 'views' and 'components' to organize folders and respective files systematically, promoting collaboration, well maintainability, and debugging. The components are further divided into various sub-divisions, where each sub-divisions contents smaller components. The organization of views and components is shown in Figure 5 below.

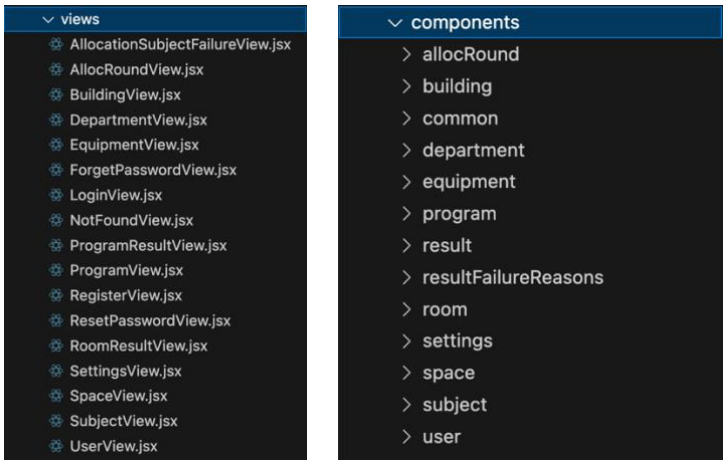


Figure 5. Organization of different views and components

6.3.3 Separation of concerns

The entire codebase of front-end part is divided into different independent sections, each responsible for distinct functionalities and similar technique is followed in back-end part as well. The styles has been defined under the style section for different themes, however the main style and layout of user interface has been implemented with the Material-UI library, which is quite responsive and customizable. Similarly, the type definition of different data types and color type in the application has been defined in one module with interfaces, which makes it easy to manage and make future changes. The extended data types with the use of interfaces, a powerful feature of TypeScript, enabled the handling of different data in the application. However, the use of TypeScript is very less compared to use of JavaScript in the front-end part. Additionally, some of the generic functionalities are defined once and implemented widely throughout the application.

6.3.4 State management and data handling

The management of data is handled with popular react hooks useState and useContext. useState is a react hook that lets functional components to add and update state variables whereas useContext

lets functional components to access data from the defined context. It provides a sophisticated way to pass data through component tree without using props in child components. However, in this project, most of the data flow has been managed by passing props to child components. Another important react hook useEffect has been used to fetch data while rendering respective user interfaces. It provides a way to add side effect to an application. In addition to these, localStorage has been used to store data in a key-value format, which is a property of global window interface allowing storage facility in modern web browsers (MDN 2023a).

6.3.5 Communication with back-end server

The communication functionalities with back-end server has been defined and separated with the rest of the application part and is implemented across various areas. The common request methods for getting item, adding item, updating item, and deleting item has been defined at once in a single place and used several times as per needed. For accessing and manipulating data, fetch function has been used, which is a global method in JavaScript providing an easy logical way for fetching resources asynchronously across the network (MDN 2023b). User credentials has been stored during the login process in local storage and accessed through it for authentication purpose in the application for making different requests to server.

6.3.6 User input validation

User input validation is very crucial part of any web application to maintain data integrity and prevent security vulnerabilities. To maintain this, the generic validation functions has been defined at once in its respective module and implemented across different functionalities through the application. By doing so, any user inputs should pass through those validation rules before they are sent to the back-end. If they are not able to full fill the defined requirements, then proper error message will be shown to the user. This way, we can maintain the security of application and make sure that database will not contain any dirty data. As for example, the function vF_regDescription below will check the description with the defined regular expression and also provide proper error message.

```
export const vF_regDescription = {
  regExp: new RegExp(/^[A-Za-zäöåÄÖÅ0-9\(\)\s\/,.:-]*$/),
  hint: "A-ö big and small letters, numbers and some punctuation characters allowed",
  errorMessageFunction: (fieldName) =>
    genericErrorMessageFunction(
      fieldName,
      "has wrong format. A-ö big and small letters, numbers and some punctuation characters allowed",
    ),
};
```

6.4 Back-end architecture and design

In this project, back-end is mainly responsible for providing Application Programming Interface (API) endpoints for front-end program, processing and applying different business logics, retrieving, and manipulating data in database. Several back-end technologies have been used to implement the needed functionalities in this project, which can be viewed from a package.json file mentioned in Appendix 4. The structure and management of different modules can be viewed from the overall project structure mentioned in Appendix 5.

TypeScript has been used as programming language, which provides mush richer and wider features over JavaScript. Various custom data types have been defined in accordance with the data model and enforced into the application. Similarly in front-end part, npm has been used to install and manage different dependencies in the project. In this section, I am going to explore about different used technologies, structures, and design of the back-end in more detail.

6.4.1 Application configuration, routing, and handling HTTP methods

The index.ts file serves as the main entry point of the application, where application configures, loads, and registers all different routes and listening to back-end server happens. The code from index.ts file is mentioned below in Figure 6.

```
1   import bodyParser from 'body-parser';
2   import cors from 'cors';
3   import dotenv from 'dotenv';
4   import express from 'express';
5   import routes from './routes/index.js';
6   import logger from './utils/logger.js';
7
8   const app = express();
9
10  dotenv.config({});
11
12  app.use(cors());
13  app.use(bodyParser.urlencoded({ extended: true }));
14  app.use(express.json());
15  app.use(`${process.env.BE_API_URL_PREFIX}`, routes);
16
17  app.listen(process.env.BE_SERVER_PORT, () => {
18    logger.log('info', `Backend starting on port ${process.env.BE_SERVER_PORT}`);
19  });
```

Figure 6. Back-end entry point, content of an index.ts file

To facilitate the process of creating API endpoints, routing, and handling configurations, Express.js has been used in the program. Express is a fast, unopinionated, minimalist web framework based on Node.js http module and components, providing flexibility and high customization for developers (Azat Mardan 2014). In Figure 6 above, an instance of Express.js application has been created and

represented by app object which has methods to define routes, use of middleware, and needed configurations.

For example, the app object allows the use of Cross Origin Resource Sharing (CORS) policy to handle response in request from a different port, which browsers by default enforces the rule for security reasons. By configuring the app with cors middleware, now the requests coming from other protocols can receive response from a back-end server protocol. Another third-party middleware body-parser has been configured to use in the application, which parses the incoming request bodies and makes it available into body property of request object.

Similarly, express.json has been used, which is a built-in middleware function of express, which parses incoming requests with JSON payload (Express 2017). After parsing the request body, it will be available for different route handlers. In addition to these, a Node.js module dotenv has been installed and configured to enable the loading and usage of environment variables into the application. Finally, the application will listen to the back-end server in the specified port defined in .env file along with a call back function. The .env file contains configuration settings, values that can vary depending upon the development, production, or testing environment, and sensitive information of application.

In this project, back-end offers large number of API endpoints which are defined within different respective files inside routes module. The management of defining various routing is done in differentiable and manageable practice, which has made defining and understanding different routing parts easier. All different routes have been loaded using middleware inside index.ts file of routes module. The file structure and management of different routes is shown in Figure 7 below.

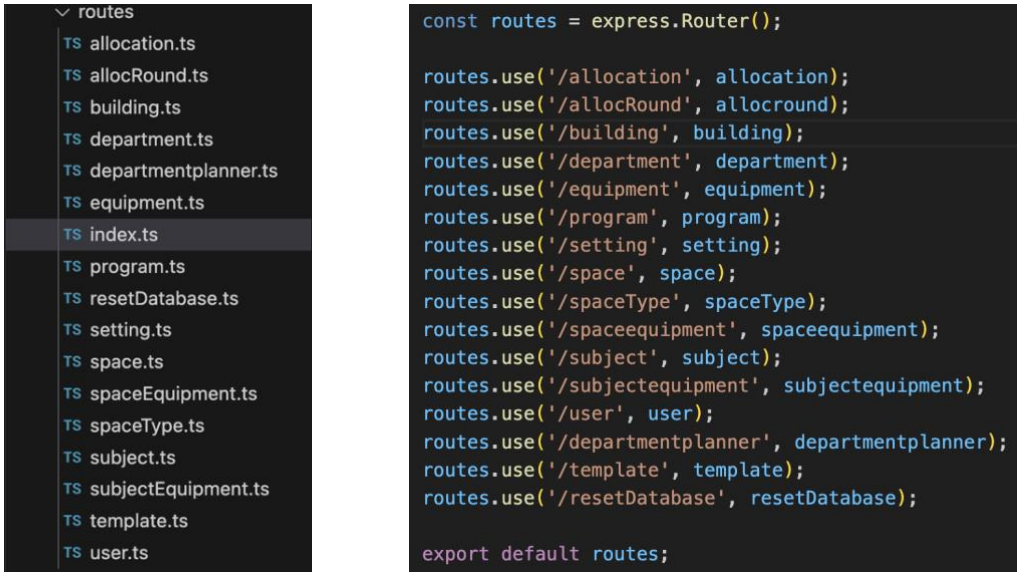


Figure 7. Structure and loading application routes

A new router object has been created with express.Router() function, which has been used to define routes for different paths and middleware. Each route files consists of various CRUD functionalities and defined in the order of get many, get one, create, update, and delete with methods get, post, put, and delete respectively of router object. After passing through all the application-level middleware, the request is finally available to route handlers in their respective paths, where it undergoes various needed security checks before performing any database operations to maintain security and integrity of data. In this project, most of the database operations are performed in routes level. To represent this, below is a slice of code from building route where it defines a routing path to get all buildings from database in Figure 8.

```
building.get('/',
  [authenticator, admin, statist, planner, roleChecker, validate],
  (req: Request, res: Response) => {
    db_knex('Building')
      .select()
      .then((data) => {
        successHandler(req,res,data,'Successfully read the buildings from DB',);
      })
      .catch((err) => {
        dbErrorHandler(req,res,err,'Error trying to read all buildings from DB',);
      });
  },
);
```

Figure 8. Definition of route to retrieve all buildings

6.4.2 Log messages and debugging

Various levels of logging message facilitates the debugging process for developers. Logging messages into console is a de facto technique of debugging in web application development. The log method of global console object in Node.js ecosystem can be used to log message into console during back-end development. However, there are other technologies as well for message logging. As for example, functionality for logging messages have been implemented with a popular Node.js logging library called Winston. A custom format for logging messages has been created, which has made message logging process flexible and extensible. In addition to this, logs related to different event types are transported for storage into separate file location in the project, which has make the debugging process easier and faster.

6.4.3 Testing API endpoints with Postman and VS Code REST client

We need to make sure that each and every API endpoints that we develop in server works well enough so that we don't leave any surprises to the front-end program. For this purpose, Postman was initially used to test API endpoints, which is widely used application for designing, testing, and documenting web APIs. But later the development team decided to use REST client which is a VS Code plugin for testing. The decision to use REST client for testing was quite wise, since it increased the productivity by providing convenient way to interact with APIs and test them without having to leave the development environment.

6.4.4 Authentication, authorization, and data validation

Each incoming request from front-end undergoes through different middleware functions that performs authentication and authorization checks before they are allowed to retrieve and manipulate with data. As for example, a middleware function 'authenticator' defined in Figure 9 below extracts the token sent in header of request and verifies with the help of jsonwebtoken and passes request to next handlers in a successful case. If the token is not found or mismatched, then a proper error message will be given as response. The back-end server has also set the token expiration time in .env file and responds accordingly, if it gets expired.

```
10  export const authenticator = (req: Request, res: Response, next: NextFunction,) => {
11    const authHeader = req.get('Authorization');
12    const token = authHeader?.split(' ')[1];
13    if (token == null) {
14      authenticationErrorHandler(req, res, 'Login TOKEN not found in headers');
15      return;
16    } else {
17      try {
18        const verified = jsonwebtoken.verify( token, process.env.SECRET_TOKEN as string,);
19        const currentTime = Math.floor(Date.now() / 1000);
20        const iat = typeof verified === 'object' ? verified.iat ?? 0 : 0;
21        if (currentTime - iat > Number(process.env.TOKEN_EXPIRATION_SECONDS)) {
22          authenticationErrorHandler(req, res, 'Token Expired');
23          return;
24        }
25        req.user = verified as User;
26        req.areRolesRequired = 0;
27        req.requiredRolesList = [];
28        next();
29      } catch (err) {
30        authenticationErrorHandler(req, res, 'Login token found but NOT valid');
31      }
32    }
```

Figure 9. A middleware function authenticator for token validation

In addition to the authenticator middleware, which is the first level middleware applied in route levels for retrieving all items, admin, statist, planner, roleCheker, and validate are other middleware

functions for checking the specific role and validating incoming data. The application has use cases for different user profiles as admin, statist, and planner. So these middleware functions will check if the request is coming from admin, or statist, or planner to grant authorization for different services based upon each roles.

The request data has been initially validated with different validators and they are applied as first level middleware functions in route levels for respective operations. The validators are defined using express validators, which has made the validation process efficient. Express validators can be used to validate and sanitize requests, offering tools to check if the request is valid or not and if they meet the defined validation rules in express applications (Express-validator 2023). In this project, application receives data from front-end application in many different field names and it is necessary to perform the security check and sanity of those data.

As for example the createIdValidatorChain function below in Figure 10 performs validation check if it is a number or if it is empty, where validateIdObl is an array performing validation against fieldname 'id'. Similarly like this, many validation arrays has been defined and integrated into a single validator chain to perform checks for requests carrying multiple fieldnames into the application.

```
export const createIdValidatorChain = (fieldName: string,): ValidationChain[] => [
  check(`${fieldName}`)
    .matches(/^[0-9]+$/)
    .withMessage(`${fieldName} must be a number`)
    .bail()
    .notEmpty()
    .withMessage(`${fieldName} cannot be empty`)
    .bail(),
];

export const validateIdObl = [...createIdValidatorChain('id')];
```

Figure 10. Custom validator function for id validation

6.4.5 Response handlers

Based upon the request, a back-end server must respond accordingly. Sometimes, receiving desired response might become challenging as there can be server issues or database malfunctions. There might be errors in request as well for e.g. missing or invalid token, or even unauthorized access. To handle similar situations, different response handlers has been defined in the application to send a proper response status code, error message or content in case of successful requests. A proper response status code is necessary especially during development phase as it provides valuable information for developers.

In general level, HTTP status codes have following meaning according to the standard defined by Internet Engineering Task Force. (Fielding, Nottingham & Reschke 2022).

- 1XX – Informational messages
- 2XX – Successful responses
- 3XX – Redirection messages
- 4XX – Client error responses
- 5XX – Server error responses

In this project, 2XX and 4XX are used mostly in response status code.

The back-end server has defined response handlers for handling situations like successful requests, database errors, request errors, authentication error, validation error, and authorization error. As for example, a handler functions for successful case and request error case is shown in Figure 11 below, which are applied in promise chain in route levels accordingly.

```
export const successHandler = (req: Request,res: Response,data: unknown,message: string,) => {
  const logMessage = routePrinter(req) + logMessagePrinter(message, successMessage);
  logger.http(logMessage);
  const body = typeof data === 'number' ? { returnedNumberValue: data } : data;
  res.status(200).send(body);
};

export const requestErrorHandler = (req: Request,res: Response,message: string,) => {
  const logMessage = routePrinter(req) + logMessagePrinter(message, requestErrorMessage);
  logger.error(logMessage);
  res.status(400).send(responseMessagePrinter(requestErrorMessage, logMessage));
};
```

Figure 11. Response handler functions for success and error cases

6.5 Data model and database operations

In this project, MariaDB has been used as a database system, which is a relational DBMS.

After having a thorough understanding of customer requirements, the development team has defined suitable data model, which is a conceptual representation of structure and organization of application data. Many different entities, attributes, constraints, and their relationships has been defined to store, retrieve, and manipulate data. A full representation of data model is shown in Figure 12 below. Based on this defined data model, various required queries have been written in the project.

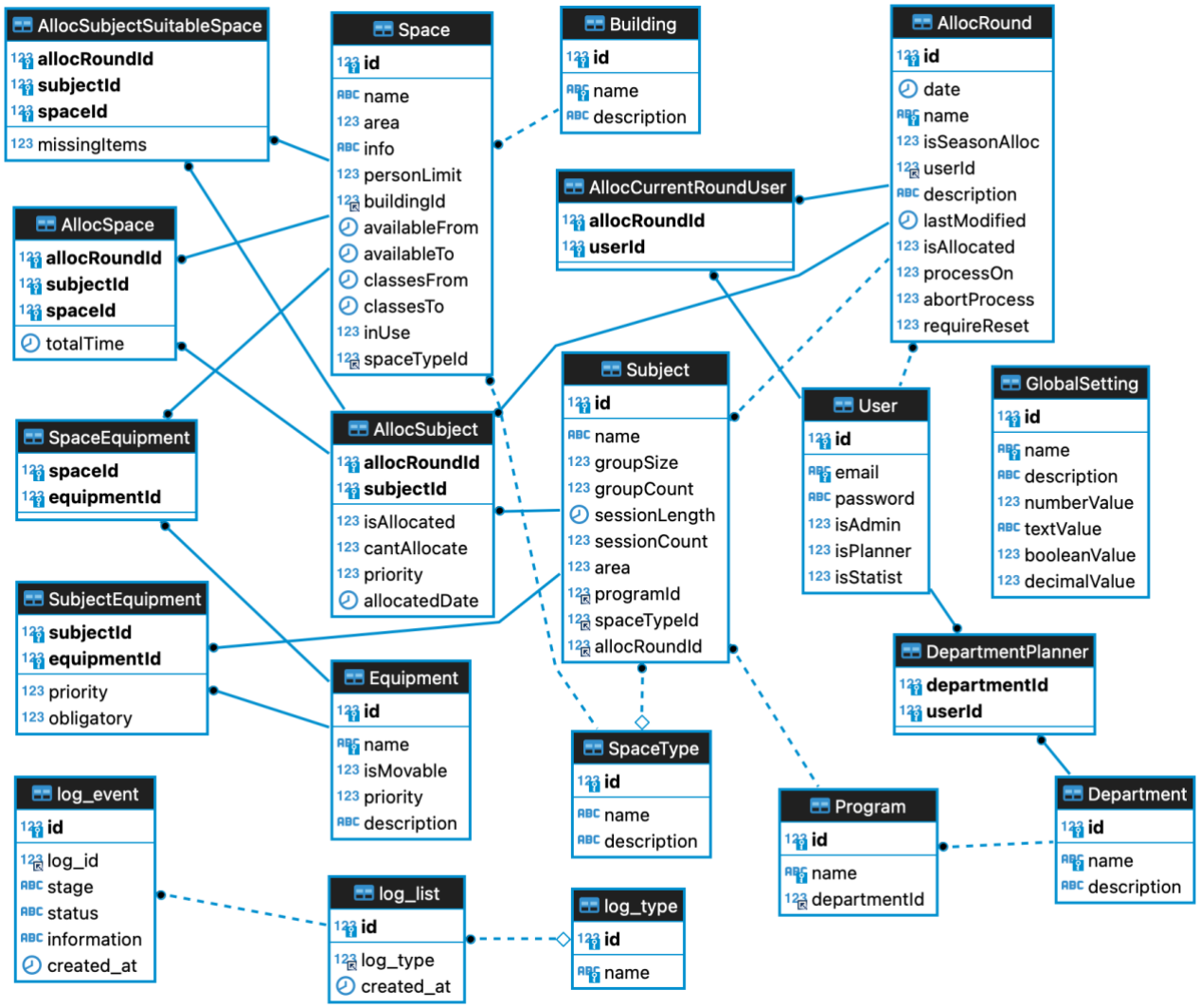


Figure 12. An overview of data model: entities, attributes, constraints, and their relationships

6.5.1 Database schema definition, creation, and implementation

Based on the defined data model, specific data types has been defined according to the attributes for each entities. Also, primary and foreign key constraints has been defined to enforce relationships between entities, maintain uniqueness, data integrity and facilitate data retrieval and manipulation process. A total of 20 different entities have been defined in the data model along with its respective attributes. All schemas has been defined with data definition language to create tables and needed constraints. As for example, a SQL statement for creating DepartmentPlanner table along with primary and foreign key constraints is shown in Figure 13 below.

```
CREATE TABLE IF NOT EXISTS DepartmentPlanner (
    departmentId    INTEGER     NOT NULL,
    userId          INTEGER     NOT NULL,

    PRIMARY KEY (departmentId, userId),

    CONSTRAINT FOREIGN KEY (departmentId) REFERENCES Department(id)
        ON DELETE CASCADE
        ON UPDATE NO ACTION,
    CONSTRAINT FOREIGN KEY (userId) REFERENCES User(id)
        ON DELETE CASCADE
        ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 13. SQL statement for creating DepartmentPlanner table

In addition with create table statements, some of the test data have been inserted into database using SQL insert statements. Some procedures and functions have also been defined to perform specific tasks. The SQL statements related to the creation of tables, dropping tables, inserting test data into table, creating procedures and functions has been defined into respective files under single folder. All of the created SQL statements has been executed into DBeaver platform, which is a database admin client tool, to implement database schema in local MariaDB database server.

6.5.2 Database setup and connection

A local MariaDB database has been used during the development phase of the application. The user credentials for database creation and setup have been created during the setup process and stored into .env file of back-end server for the connection. Optionally, the cloud database could have been accessed via secured shell by creating tunnel, which has been hosted into CSC cloud ubuntu virtual machine. CSC is a Finnish research institute of information technologies and services. To establish

a connection into database, Knex.js has been used and configured to use MySQL client with connection details. Database connection was established using Node.js MySQL driver as well during the initial phase of development, but this approach was ultimately not followed in the development.

6.5.3 SQL query builder – Knex.js for database operation

Knex.js has been used for generating database queries in server, which is a JavaScript SQL query builder. Instead of writing raw SQL queries, Knex.js generates it based upon chain of JavaScript code using Knex.js methods. A simple use case of Knex.js is presented in Figure 14 below, where a SELECT SQL query has been constructed behind the scenes using select method of knex object db_knex. The result of the query has been handled later in promise chains accordingly.

```
building.get('/',[authenticator, admin, statist, planner, roleChecker, validate],
  (req: Request, res: Response) => {
    db_knex('Building')
      .select()
      .then((data) => {
        successHandler(req, res, data, 'Successfully read the buildings from DB',);})
      .catch((err) => { dbErrorHandler(req, res, err, 'Error trying to read all buildings from DB',);});
  },
);
```

Figure 14. Implementing knex.js query method in building route

7 Analysing the architecture of the sample project

In this section, I am going to analyse the architecture and design principles that has been used in this project based on February 2024 code version. Analysing system's architecture and design principles is crucial for ensuring quality of system, which can also identify possible areas for further improvement. To facilitate the analysis process, I am going to look the overall project from many different angles of software development and write my thinking and perspectives by reflecting to the theoretical base that I have made.

7.1 Selection of technological stacks and project management

Based on my development experience, the selected technological stacks in the project well-suited for full stack web application. Programming languages, JavaScript and TypeScript served well for different functionalities in front-end and back-end part respectively. The use of TypeScript in back-end part allowed to use more advance features over JavaScript specially in defining data structures and generics. The relational DBMS MariaDB, which is compatible with MYSQL, provided an efficient way in creating database and query execution techniques. The choice of React as front-end library, which has high community support, helped in creating user interfaces in an efficient manner. Material UI on the other hand, provided good layouts, styles, and responsiveness across multiple devices. The use of Express.js in back-end, eased the process of creating REST API features. Postman and VS Code Rest client well supported the testing of REST API endpoints. The selected technological stacks used in project served quite well for the purpose.

Trello was used as the project management tool for the project, which helped in creating a nice workspace for project team members. It allowed to create tasks, assign it to team members, and monitor the progress of each task, in different sprints. There are other project management platforms and tools as well, however based on my experience specially during the development of project, the selected tool supported well from project management point of view.

7.2 Application development methodology

Scrum was adapted as a development methodology in this project. The application was developed in an iterative way breaking down into smaller but manageable components and improving it further in each sprint. This methodology facilitated flexibility and changeability during the application development cycle. As the project was progressing, new requirements were added, and changes were made frequently. Furthermore, this methodology helped in maintaining customer collaboration during the development of project. Minimum viable product was developed early enough and hence the customer had an opportunity to visualize the application outlook in advance. By referring to the

agile manifesto mentioned above in agile model (see sub-chapter 3.2), which prioritizes customer collaboration, early working solution, changeability, people and interactions, the chosen methodology well suited for the sample project's development.

7.3 Architectural characteristics analysis

The architectural characteristics of sample project can be analysed from different angles. This is a simple web application that is going to be used by limited number of users and hence a huge workload is not expected. But there can be massive data in long run, which will be saved in MariaDB database system, possibly somewhere in cloud, which has an ability to handle increasing workload and performance. There is well-communication between front-end, back-end, and database layer during the development phase and it should behave the same during production as well. Different error handling mechanisms have been applied for different type of user requests to perform the system reliably. In production phase also, despite of network error, the application aims to serve reliably.

The system fulfils all the requirement functionalities and performs as intended. User requests are immediately listened and answered by system. There is small delay in calculating the allocation of spaces, otherwise the application has adequate performance efficiency. The application has strong security methods with user authentication and authorization techniques in place. Inputs are validated strongly both in frontend and backend before they are allowed to interact with database. The program codes are well-organized in all part of development by breaking them into corresponding modular structures. Features and functionalities are utilized and reused as much as possible and hence proves to be a maintainable system.

The back-end and database part are quite extensible to develop new features. While we can develop new features in most of the front-end part easily but in some places, it might be a bit challenging as there is data flow from component to components and some changes or new added features might affect existing ones. However, the application is quite extensible to other compatible tools and technologies for Node project in developing new features and functionalities.

The system is quite observable from the architectural point of view. The implementation of logging messages in front-end has provided useful information during development phase through browsers' console. The built in feature of browser for e.g. network activity has helped to observe the system activities in more detail. On the other hand, the logging messages in back-end console, transportation of different log events to respective file locations has helped to observe and understand the system state more strictly.

By reflecting to the theoretical knowledge about architectural characteristics gathered in software architecture section, the application is proven to be scalable, reliable, efficient, extensible, maintainable, secured, and observable (4.3).

7.4 Design principles and coding styles

The implementation of design principles and coding styles in the project mostly follows the general concepts and very rarely adheres to the object-oriented principles. This is because of the nature of programming language used in the project as well. The project strongly adheres to “Don’t Reinvent the Wheel” principle (5.7). There is good use of third-party libraries in both front-end and back-end for several needed functionalities. From creating stylish layouts, printing custom log messages to console, authentication and authorization management, database queries, creating REST endpoints, and so on, we have used already established and developed functionalities available through different libraries. This has made the development faster and helped in concentrating to other needed requirements.

Modularity and separation of concerns are strongly and widely followed in all part of development, establishing the fundamental design principles of project (5.1, 5.2). The development of whole project is divided into three major parts: front-end, back-end, and database. Development of each part is further divided into multiple different modules. As for example, common styles of applications are defined within styles module, navigation is managed within routers module, different components and views are defined within components and views modules respectively. Functionalities related to communication with back-end server has been defined under ajax module, while various validation rules are defined under validation module.

Similarly, the development of back-end part has been divided into multiple different modules (5.1). Various back-end routing has been defined under routers module whereas some functionalities related to database operations are defined under services module. Validation rules and user authorization techniques has been defined within different modules. Response handlers for different type of requests are managed in a single place and separated from other part of application. The database configuration has been separated from application configuration. In addition to this, different SQL scripts related to creation of tables, insertion of data, dropping tables, creating of procedures, and dropping, has been defined separately (5.2).

The overall program is designed in a simple but structured manner (5.4). The code in all part of development is divided into respective modular structures, which has made the development process efficient. Each module consists of code only related to one nature and has facilitated the maintaining process. The naming convention for files and folders, variables, functions are mostly

done in a uniform and standard way. These design principles and coding patterns has provided the strong guidelines in organizing the whole project and implementing each specific solution.

7.5 Support for team collaboration and contribution

The use of Git and GitHub in the project simplified the development team collaboration and contribution to the project. The rich features of Git enabled team members to work together and contribute to the project independently from different angle of interest and expertise without affecting each other's development. GitHub allowed to create pull requests, make review, and finally merge into the main branch of source code repository. These tools, provided a nice platform and techniques to collaborate in the same codebase within several team members, providing everybody a chance to contribute.

8 Suggestions for architectural improvements

Even though the development of project was smooth, directed by its predefined architecture, and design principles, I have discovered some issues, reflecting to my theoretical base that I have made and from my own experience during the development of project. In this section, I am going provide some suggestions that might be useful for further improvement of this project and for future development projects as well.

8.1 One-source documentation with frequent update

The used tools and technologies, chosen architecture and design principles can be viewed by exploring the project's codebase, however, customer requirements, restrictions and limitations were difficult to find out. This made a bit confusing for new team members, who joined the project in later stages. The database descriptions, installation and configuration description, backend services description, and REST API endpoint's documentation was there in the project, but it was not updated in later stages. On the other hand, REST API endpoint's documentation and backend services description was in database section instead of backend section. The feedback gathered during the project cycle were scattered in different places, for e.g. on course teacher's own computer or in Teams channel or in students' own notes. Documentation is also an important aspect of software development and hence plays an important role in understanding and maintaining the project in long run. I would suggest a better way of handling whole project documentation within a single source, which should be updated as project progresses and requirements updates.

8.2 Consistent team communication

Besides having an efficient way to contribute and collaborate, communication in the development team was poorly done during the development time when I participated. Everybody was contributing and learning but most of them did independently. In my opinion frequent communication across all team members is quite necessary to have good team spirit and overall success of project. By doing so, the development team could notice motivation, weakness, and expertise of each team member and provide relevant support if necessary. This can also help in identifying a 'go to guy' in the project for tackling immediate problems, ideas, and support. Furthermore, if somebody is behind in some tasks or doesn't have any clue on what and how to do, then they should immediately ask for help inside the team. There might be somebody in the team who can provide relevant support. This further helps in building a better working environment within a team and minimizes the project development time.

8.3 Uniform coding patterns and guidelines

There has been some misunderstanding or lacking uniform coding patterns and guidelines across the project. As for example, some functions are defined as normal functions instead of arrow functions both in front-end and back-end, which has violated the standard guidelines of using arrow functions. Also, some of the REST endpoints in some routes has been defined in different order and not following the standard format of get many, get one, post, put, and delete which was defined during the development. Some functions which have high reusability has been defined in wrong place. For example, in back-end part, handleErrorBasedOnErrno function has been defined inside building route. This function has a capability of providing generic solution across various routes and could have been defined in suitable place for example inside responseHandlers.

Furthermore, imports of local and external modules across different files are mixed up. If we could place the external imports first and local imports in second place, separated by one line space consistently across all files, it will improve the readability of code and distinguish them well. There are also some unused imports in different files which could have been removed after finding out that it is not necessary anymore.

8.4 Testing

While the back-end REST API functionalities were tested using Postman and VS Code Rest client, we were all the time loading browsers to test any developed front-end feature and checking for log messages and errors in console. Some technical capabilities for system testing are done with Selenium library, however the testing is quite limited. In my opinion, we should have implemented many test cases that would test system's functionalities in an elegant way both in front-end and back-end part. By referring to the architectural characteristics of software applications, testability is an important character that defines the quality of system and hence it needs to be addressed properly.

8.5 Prioritizing reusability

The project initially had quite a lot of redundant code, but with series of improvements, we were able to develop more reusable code which has increased the overall code quality, consistency, and maintainability. However, there is still room for improvements in reducing the redundant code for e.g. input data validation in front-end part. There, we have used similar code, which has similar chain of conditional statements across different validation files. We can define some generic functions that can be reused multiple times across various validation files. Eventually, this will reduce the overall code length and helps in maintaining the project.

8.6 Database operation within service module

Most of the database operations has been carried out within the route level in current implementation. This practice has mixed up handling of different routes, database operations, and response handlers in same place. The functions inside routes primarily should focus on handling application routing and HTTP requests and responses. There has been little practice of defining functionalities related to database operations in service module. Service layer in back-end is mainly responsible for having functionalities related to business logics and data manipulation. Keeping database operations and business logics separate from routing promotes separation of concerns, which is one of the fundamental design principles. This practice helps in understanding programs better by making codebase more modular and facilitates the maintaining process.

8.7 Data management in front-end

Currently application data has been managed with react hooks in front-end. Data has been passed from parent components to child components using props. Handling the flow of data by passing it from components to components is easy in small applications but it gets complicated if the application size becomes larger. The use of useContext react hook has supported the data accessing mechanism a bit, but its usage in the application has been quite limited. Based on my experience from other projects, the use of state management library for example, Redux, can be more beneficial for this project as it centralizes the state management by defining the single global application state.

Debugging and testing becomes easier when we have a single source of state management in an application with the use of Redux. The data sharing between components becomes easy and we can achieve smooth communication between components regardless of component hierarchy. This also helps in separating user interface creation process and application logics, thus promotes separation of concerns and modularity of an application.

9 Thesis Reflections

Finally, we have come to the end of this report and here I am going to share about my thesis work experience, learnings, and challenges that I have faced. Through this research work, I have now realized that software architecture and design principles are quite extensive areas and has an important fundamental role in shaping the development of quality software.

The main goal of this thesis was to study the sample project´s architecture and try to find out areas for improvement. During the development of this project, we were able to perform refactoring and even developed new features very quickly in an existing codebase, which was developed by other team members. There were many developers working in the same project at the same time, but we worked independently without breaking each other's code. The debugging process throughout the application was quite efficient. I realized all of this happened very smoothly and efficiently and wanted to find out underlying supporting factors and raised some common questions.

I have gathered information from various sources regarding software architecture, characteristics, and design principles. I have used some e-books, articles, and research papers to study the relevant topics however, most of my information sources references to blogs written by software professionals. This was because, I was able to find the recent information in my searched topics and they were thoroughly understandable. In addition to this, I have studied the sample project's architecture in more detail. This process helped me to understand architecture and design principles from both theoretical and practical point of view.

Now with the completion of this thesis work with the collection of theoretical information, studying project's architecture, evaluating, and providing suggestions for improvements, I have now fulfilled my objectives and have provided common answers to my questions. I have tried to evaluate the architecture of project by reflecting to the established theoretical base, however some of the suggestions for architectural improvement comes from my own development experience.

I have been always interested in coding but never wrote a report by thoroughly analysing program code. This thesis provided me an opportunity to write a long report in a structured and professional manner. I gained skills on sourcing information from various sources, integrating them into a report, and adhering to proper referencing guidelines. I also gained valuable knowledge in different software development methodologies and phases, various architecture types, architectural characteristics, and many well-practiced design principles.

By thoroughly studying the project's architecture, I have now honed my ability in analysing code, program structure, and overall project development from the lens of architecture and design principles. The knowledge that I have gained in software architecture and design principles throughout this thesis work will be beneficial in improving my code writing practices and project development. Furthermore, this knowledge will be valuable for me specially during job interviews, because I can now tackle a particular problem in better way than before.

While I am completing this report within the planned time duration, I have noticed little deficiency in my time management skills. This thesis work is my personal project, and I should have maintained consistent progress throughout the duration. However, I have observed that my activity levels have been fluctuated, which in my opinion, has hindered the delivery of better result. I have realized that improving my time management skills with consistent efforts is essential for my own personal growth and future career aspirations as well.

9.1 Future research

During my thesis work, I primarily focused in finding information from various sources and thoroughly delving into the sample project's architecture and coding patterns in an iterative manner. This provided me a good base to perform the analysis and provide few suggestions for further improvement of project. However, I could have done this research work also by conducting interviews with software professionals in addition to finding and collecting information by myself.

Latest practices and challenges related to software architecture and designs could be discovered by collecting the data obtained from interview. The data obtained from interviews would provide an additional reference for analysing the architecture of project in addition to theoretical base. By doing so, I could possibly provide better suggestions for architecture improvement. Additionally, it could also give thesis a better quality.

Finally, during this thesis work process, I had an opportunity to listen to other people's thesis work presentations, which were mostly related to artificial intelligence in software industry. Referencing to those presentations, I have a feeling that artificial intelligence will have a significant role and cannot be avoided in software development field. Now, I have also cultivated a curiosity of integration and adaptation of artificial intelligence in information systems and its potential impact on software architectures and design principles, which will be good topic to explore in future.

Sources

Ahmad, A. 2023. Essential Software Design Principles. URL: https://www.designgurus.io/blog/essential-software-design-principles-you-should-know-before-the-interview. Accessed: 1 April 2024.

Ali Khan, S. M. 2023. Waterfall Model Used In Software Development. URL: https://www.c-sharpcorner.com/article/waterfall-model-used-in-software-development/. Accessed: 26 February 2024.

Alexandrian, R. 2019. Non Functional Requirements: Maintainability. ArDIGITAL. URL: https://argondigital.com/blog/product-management/non-functional-requirements-maintainability/. Accessed: 21 April 2024.

Ashanin, N. 2018. Quality attributes in Software Architecture. Medium. URL: https://medium.com/@nvashanin/quality-attributes-in-software-architecture-3844ea482732. Medium. Accessed: 21 April 2024.

Begg, C. & Connolly, T. 2015. Database Systems A Practical Approach to Design, Implementation, and Management. Pearson.

Chacon, S. & Straub, B. 2014. Pro Git. Apress. URL: https://archive.org/details/progit-en/page/30/mode/2up. Accessed: 22 April 2024.

Code Reliant 2023. Modularity: A Pillar of Reliable Software Design. URL: https://www.codereliant.io/modularity-a-pillar-of-reliable-software-design/. Accessed: 20 April 2024.

Codecademy 2024. Back-end Web Architecture. URL: https://www.codecademy.com/article/back-end-architecture. Accessed: 19 April 2024.

Coursera 2024. Front-End vs. Back-End Developer: Understanding the Differences. URL: https://www.coursera.org/articles/front-end-vs-back-end. Accessed: 20 April 2024.

Express 2017. API Reference. URL: https://expressjs.com/en/4x/api.html#express.json. Accessed: 13 March 2024.

Express-validator 2023. express-validator. URL: https://express-validator.github.io/docs/. Accessed: 20 March 2024.

Fernandes, S. 2020. You Ain't Gonna Need It — The YAGNI Principle. Medium. URL: https://medium.com/@senfernandes/you-aint-gonna-need-it-the-yagni-principle-38e2b1166505. Accessed: 5 April 2024.

Fielding, R. Nottingham, M. & Reschke, J. 2022. HTTP Semantics. URL: https://httpwg.org/specs/rfc9110.html#overview.of.status.codes. Accessed: 20 March 2024.

Gabby, T. & Villa, M. 2022. The Basics of Package.json. URL: https://nodesource.com/blog/the-basics-of-package-json/. Accessed: 6 March 2024.

GitHub 2024. About GitHub and Git. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git. Accessed: 18 March 2024.

Gladun, S. 2024. What Is the Software Development Life Cycle (SDLC) and How Does It Work? URL: https://agilie.com/blog/what-is-the-software-development-life-cycle-sdlc-and-how-does-it-work. Accessed: 26 March 2024.

Godbolt, M. 2016. Frontend Architectures for Design Systems. O'Reilly. URL: https://learning.oreilly.com/library/view/frontend-architecture-for/9781491926772/ch01.html. Accessed: 21 April 2024.

Harris, C. 2024. Microservices vs. monolithic architecture. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith. Accessed: 30 March 2024.

Heath, M. 2011. Essential Developer Principles #2–Don't Reinvent the Wheel. URL: https://markheath.net/post/essential-developer-principles-2dont. Accessed: 6 April 2024.

Hu, C. 2023. An Introduction to Software Design. URL: https://books.google.fi/books?hl=en&lr=&id=hxbNEAAAQBAJ&oi=fnd&pg=PR6&dq=software+design+principles&ots=hKwe-jy00n&sig=2MuaN23SJuh2KGIwRK8Bg4A7DO0&redir_esc=y#v=onepage&q=software%20design%20principles&f=false. Accessed: 1 April 2024.

Koper, K. 2024. The 7 Stages of Software Development. URL: https://www.netguru.com/blog/stages-of-software-development. Accessed: 25 April 2024.

Kokhan, S. 2023. Quality Attributes in Software Architecture. URL: https://www.linkedin.com/pulse/quality-attributes-software-architecture-serhii-kokhan/. Accessed: 30 March 2024.

Krysik, A. 2023. SDLC Guide: Requirement Analysis in Software Engineering. URL: https://stratoflow.com/requirements-analysis/. Accessed: 25 April 2024.

Krishna, A. 2023. What is SOLID? Principles for Better Software Design. URL: https://www.freecodecamp.org/news/solid-principles-for-better-software-design/#:~:text=The%20SOLID%20principles%20are%20a,understand%2C%20modify%2C%20and%20extend. Accessed: 1 April 2024.

Learntek 2019. SDLC (Software Development Life Cycle). URL: https://www.learntek.org/blog/sdlc-phases/. Accessed: 25 April 2024.

MDN 2023a. Window: localStorage property. MDN. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. Accessed: 11 March 2024.

MDN 2023b. Using the Fetch API. MDN. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch. Accessed: 11 March 2024.

Mardan, A. 2014. Express.js Guide. URL: https://pepa.holla.cz/wp-content/uploads/2016/08/Express.js-Guide.pdf. Accessed: 4 March 2024.

Martin, R. C. 2014. Agile Software Development, Principles, Patterns, and Practices. URL: https://dl.ebooksworld.ir/motoman/Pearson.Agile.Software.Development.Principles.Patterns.and.Practices.www.EBooksWorld.ir.pdf. Accessed: 1 April 2024.

Mete, F. 2021. Fundamentals of software architecture: Part I. Medium. URL: https://federicomete.medium.com/fundamentals-of-software-architecture-part-i-5de40012609a. Accessed: 30 March 2024.

Mia-Platform 2023. Observability in Software Engineering – Metrics, Logs, Traces. URL: https://mia-platform.eu/blog/observability-software-engineering/#:~:text=In%20software%20engineering%2C%20observability%20is,performance%20of%20intricate%20software%20systems. Accessed: 30 March 2024.

Natesan, N. 2019. How to implement Design Pattern – Separation of concerns. Software intelligence Plus. URL: https://www.castsoftware.com/pulse/how-to-implement-design-pattern-separation-of-concerns. Accessed: 20 April 2024.

Niculae, C. 2017. Software product development phases. URL: https://www.areusdev.com/software-product-development-phases/. Accessed: 25 April 2024.

Parra Rosales, I. J. 2023. The Evolution of Software Development Cycles: A Journey from the Beginnings to the Present. URL: https://medium.com/@josueparra2892/the-evolution-of-software-development-cycles-a-journey-from-the-beginnings-to-the-present-de1496f4c148. Accessed: 20 February 2024.

Panchal, S. 2022. Monolithic V/s Distributed Architectures. Medium. URL: https://sumeetpanchal-21.medium.com/monolithic-v-s-distributed-architectures-ae5796568d95. Accessed: 30 March 2024.

Plutora 2022. Understanding the DRY (Don't Repeat Yourself) Principle. URL: https://www.plutora.com/blog/understanding-the-dry-dont-repeat-yourself-principle. Accessed: 20 April 2024.

Riyani, N. 2023. Performance, Scalability, and Reliability: The Pillars of Robust Software. URL: https://www.linkedin.com/pulse/performance-scalability-reliability-pillars-robust-software-riyani/. LinkedIn. Accessed: 30 March 2024.

Richard, M. & Ford, N. 2020. Fundamentals of Software Architecture. O'Reilly. URL: https://learning.oreilly.com/library/view/fundamentals-of-software/9781492043447/preface01.html. Accessed: 23 March 2024.

Richards, M. 2022. Software Architecture Patterns. O'Reilly. URL: https://learning.oreilly.com/library/view/software-architecture-patterns/9781098134280/copyright-page01.html. Accessed: 29 March 2024.

Risener, K. 2022. A Study of Software Development Methodologies. URL: https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=1105&context=csceuht. Accessed: 25 March 2024.

Schwaber, K. & Sutherland, J. 2017. The Scrum Guide.

Sharma, L. 2022. 10 nonfunctional requirements to consider in your enterprise architecture. RedHat. URL: https://www.redhat.com/architect/nonfunctional-requirements-architecture. Accessed: 21 April 2024.

Shylesh S 2017. A Study of Software Development Life Cycle Process Models. pp. 1–1. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2988291. Accessed: 21 April 2024.

Swimm Team 2024. 6 Software design principles used by successful engineers. URL: https://swimm.io/learn/system-design/6-software-design-principles-used-by-successful-engineers. Accessed: 1 April 2024.

Team Split 2023. The seven phases of software development life cycle. Split. URL: https://www.split.io/blog/software-development-life-cycle-phases/. Accessed: 25 April 2024.

Watkins, E. 2024. Navigating the Digital Frontier: Defining Characteristics of Modern Software Architecture. iplocation. URL: https://www.iplocation.net/navigating-the-digital-frontier-defining-characteristics-of-modern-software-architecture. Accessed: 21 April 2024.

Wikipedia 2024. Sibelius Academy. Helsinki.

Yost, M. 2018. A Brief History of Software Development. URL: https://medium.com/@micahyost/a-brief-history-of-software-development-f67a6e6ddae0. Accessed: 15 February 2024.

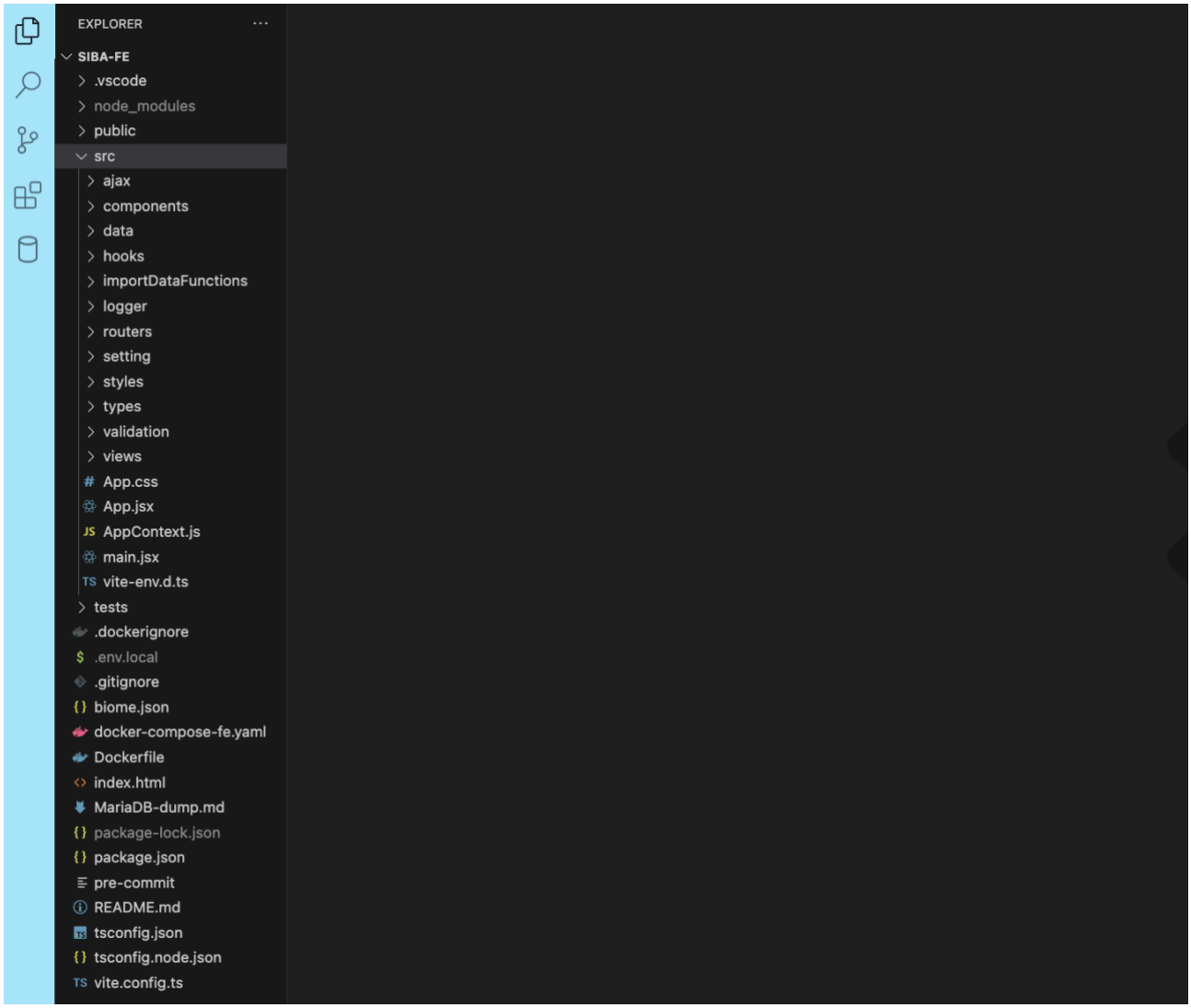
Young, D. 2013. Software Development Methodologies. URL: https://www.researchgate.net/publication/255710396_Software_Development_Methodologies. Accessed: 20 February 2024.

Appendices

Appendix 1. Technological stacks used in front-end, a package.json file.

```
{
  "name": "siba-fe",
  "version": "0.1.0",
  "private": true,
  "type": "module",
  ▷ Debug
  "scripts": {
    "clean": "rimraf dist",
    "start": "vite --open",
    "build": "tsc && vite build",
    "preview": "vite preview",
    "precommit": "sh ./pre-commit"
  },
  "pre-commit": [
    "precommit"
  ],
  "nano-staged": {
    "*.{js,jsx,ts,tsx}": "biome check --apply"
  },
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@fortawesome/fontawesome-svg-core": "^6.4.2",
    "@fortawesome/free-solid-svg-icons": "^6.4.2",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@mui/icons-material": "^5.14.18",
    "@mui/material": "^5.14.18",
    "@ramonak/react-progress-bar": "^5.0.3",
    "axios": "^1.6.2",
    "esbuild": "^0.19.6",
    "exceljs": "^4.4.0",
    "file-saver": "^2.0.5",
    "formik": "^2.4.5",
    "js-file-download": "^0.4.12",
    "luxon": "^3.4.4",
    "papaparse": "^5.4.1",
    "react": "^18.2.0",
    "react-csv": "^2.2.2",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.19.0",
    "selenium-webdriver": "^4.15.0"
  },
  "devDependencies": {
    "@biomejs/biome": "1.5.3",
    "@types/luxon": "^3.3.4",
    "@types/node": "^20.9.2",
    "@types/react": "^18.2.37",
    "@types/react-dom": "^18.2.15",
    "@vitejs/plugin-react": "^4.2.0",
    "nano-staged": "^0.8.0",
    "pre-commit": "^1.2.2",
    "rimraf": "^5.0.5",
    "typescript": "^5.3.3",
    "vite": "^5.0.0"
  }
}
```

Appendix 2. Front-end file and folder structure



Appendix 3. Routes of Nav component from front-end.

```
<Routes>
  <Route path="/login" element={<LoginView handleLoginChange={handleLoginChange} />}/>
  <Route path="/" element={<LoginView handleLoginChange={handleLoginChange} />}/>
  <Route path="/subject" element={<SubjectView />} />
  <Route path="/subject/:subjectIdToShow" element={<SubjectView />} />
  <Route path="/allocation" element={<AllocRoundView />} />
  <Route path="/roomresult" element={<RoomResultView />} />
  <Route path="/programresult" element={<ProgramResultView />} />
  <Route path="/equipment" element={<EquipmentView />} />
  <Route path="/building" element={<BuildingView />} />
  <Route path="/department" element={<DepartmentView />} />
  <Route path="/space" element={<SpaceView />} />
  <Route path="/space/:spaceIdToShow" element={<SpaceView />} />
  <Route path="/program" element={<ProgramView />} />
  <Route path="/allocation/addAllocRound" element={<AddAllocRound />} />
  <Route path="/settings" element={<SettingsView />} />
  <Route path="/users" element={<UserView />} />
  <Route path="/alloc-fail/:allocId" element={<AllocationSubjectFailureView />} />
  <Route path="*" element={<NotFoundView />} />
  <Route path="/forget-password" element={<ForgetPasswordView />} />
  <Route path="/reset-password/:id/:token" element={<ResetPasswordView />} />
</Routes>
```

Appendix 4. Technological stack used in back-end, a package.json file.

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  Debug
  "scripts": {
    "start": "tsc-watch --project . --outDir ./dist --onSuccess \"node ./dist/index.js\"",
    "devStart": "nodemon ./dist/index.js",
    "production": "node ./dist/index.js &",
    "production2": "npm2 start dist/index.js",
    "test": "echo \"Error: no test specified\" && exit 1",
    "precommit": "sh ./pre-commit"
  },
  "pre-commit": [
    "precommit"
  ],
  "nano-staged": {
    "*.{js,ts}": "biome check --apply"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "express-validator": "^6.15.0",
    "jsonwebtoken": "^9.0.2",
    "knex": "^3.0.1",
    "mariadb": "^3.2.2",
    "mysql": "^2.18.1",
    "nodemon": "^3.0.1",
    "winston": "^3.11.0"
  },
  "devDependencies": {
    "@biomejs/biome": "1.5.3",
    "@types/bcrypt": "^5.0.2",
    "@types/cors": "^2.8.16",
    "@types/express": "^4.17.21",
    "@types/jsonwebtoken": "^9.0.5",
    "@types/mysql": "^2.15.24",
    "@types/node": "^20.9.0",
    "@types/react": "^18.2.37",
    "@types/react-dom": "^18.2.15",
    "concurrently": "^8.2.2",
    "nano-staged": "^0.8.0",
    "pre-commit": "^1.2.2",
    "rimraf": "^5.0.5",
    "tsc-watch": "^6.0.4",
    "typescript": "^5.3.3"
  }
}
```

Appendix 5. Back-end file and folder structure