**SOUTHAMPTON SOLENT UNIVERSITY**

Text, logo

Description automatically generated

BSc (Hons) Computing

**Building maintainable web applications using React – An evaluation of architectural patterns used in Canvas LMS**

Author: 10152420

Assessment Title: SIP Phone Configuration Management System

# Abstract

Online application development, specifically React and Canvas LMS architectural patterns was the focus of this research. This study tested if these fundamental principles of architecture could be used to build easy-to-maintain web apps using React. The study assessed React's capabilities for developing architectural patterns used in Canvas LMS. The study emphasised on quantitative data and used surveys for collecting data. One hundred developer professionals and industry experts were surveyed. The survey collected relevant information regarding Canvas LMS React and architectural pattern use. Regression and correlation analysis were used to analyse variable associations in the study. The study found a strong positive correlation between Redux and MobX state management frameworks. These frameworks also improved React code modularity, simplicity and structure. React has demonstrated that it is capable of building maintainable web apps. This study shows Canvas LMS's extensive use of React and architectural patterns. This recommendation can help developers construct and manage web apps.

Table of Contents

[Abstract 2](#_Toc154086987)

[CHAPTER 1: INTRODUCTION & BACKGROUND 5](#_Toc154086988)

[1.1 Background 5](#_Toc154086989)

[1.2 Problem Statement 6](#_Toc154086990)

[1.3 Aims and Objectives 6](#_Toc154086991)

[1.4 Research Question 7](#_Toc154086992)

[1.5 Scope of research 7](#_Toc154086993)

[1.6 Dissertation structure 7](#_Toc154086994)

[CHAPTER 2: LITERATURE REVIEW 9](#_Toc154086995)

[2.1 Introduction 9](#_Toc154086996)

[2.2 React performance and capability in building maintainable web applications. 9](#_Toc154086997)

[2.3 Advantages and disadvantages of using React for building architectural patterns used in Canvas LMS. 11](#_Toc154086998)

[2.3.1 Advantages 11](#_Toc154086999)

[2.3.2 Disadvantages 11](#_Toc154087000)

[2.4 Use of React in building architectural patterns used in Canvas LMS. 12](#_Toc154087001)

[2.5 Theoretical framework 13](#_Toc154087002)

[2.6 Conceptual Framework 14](#_Toc154087003)

[2.7 Conclusion 15](#_Toc154087004)

[CHAPTER 3: METHODOLOGY 16](#_Toc154087005)

[3.1 Adopted Methods 16](#_Toc154087006)

[3.1.1 Research philosophy 16](#_Toc154087007)

[3.1.2 Research approach 17](#_Toc154087008)

[3.1.3 Research design 17](#_Toc154087009)

[3.2 Data Collection and research instruments 18](#_Toc154087010)

[3.3 Data Analysis 19](#_Toc154087011)

[3.4 Professional, Legal and Ethical issues 20](#_Toc154087012)

[3.5 Project management 20](#_Toc154087013)

[CHAPTER 4: DESIGN AND IMPLEMENTATION 22](#_Toc154087014)

[4.1 Implementations 23](#_Toc154087015)

[CHAPTER FIVE: RESULTS 26](#_Toc154087016)

[5.1 Introduction 26](#_Toc154087017)

[5.2 Demographics 26](#_Toc154087018)

[5.3 Correlation Analysis 27](#_Toc154087019)

[5.4 Regression analysis 29](#_Toc154087020)

[5.5 Reliability analysis 31](#_Toc154087021)

[5.5 Discussion 32](#_Toc154087022)

[5.6 Chapter summary 34](#_Toc154087023)

[CHAPTER SIX: CONCLUSION 35](#_Toc154087024)

[CHAPTER 7: RECOMMENDATION 37](#_Toc154087025)

[References 39](#_Toc154087026)

# CHAPTER 1: INTRODUCTION & BACKGROUND

## 1.1 Background

In recent years, there has been a huge change in the improvement of web applications. The formation of maintainable and adaptable arrangements requires detailed attention. Given the variety of JavaScript libraries and frameworks accessible, React with its component-based architecture and declarative syntax has become one of the most famous in comparison to the other libraries or framework (Ivanova and Georgiev, 2019). Considering online learning management systems (LMS) like canvas, the issue is that it needs to fabricate applications that not only answer the continually changing learning enviorment yet additionally adjust with goals of viability and extensibility required to develop complex systems used in architectural patterns of Canvas LMS (Colby, 2020).

The growth of complex web applications has made maintainability a big factor. Large-scale codebases demand work, and badly designed applications may suffer from duplication of code, difficulties in adding new features or bug fixes (Rahman, 2021). In this environment, the advent of Java-based frameworks such as React has opened up new prospects for developing maintainable web applications. Its component-based paradigm breeds reusability, modularity and better developer experience which would seem to result in more maintainable code. Canvas LMS, a widely used open-source learning management system (LMS), provides an interesting example of analyzing the architecture patterns in React applications (Sharma, 2021). Canvas has a very rich feature set, and also many users; it is thus an intricate evolving web application. This recent move to React gives an opportunity for reflecting on the maintainability of adopted architectural patterns.

Within the React ecosystem, several architectural styles have developed. Each style provides a different way to organize and structure code. Redux and MobX are examples of common patterns. These patterns are designed to solve problems like state management, data flow and communication between components. Ideally these patterns help make code much more organized, readable and modular (Ventura, 2021). Yet choosing and carrying out suitable patterns for a given application can be one of the most difficult tasks. Application size, complexity of features involved, and other such aspects all need to be taken into account. Furthermore, the degree to which these patterns are effective with regard to maintainability is not generally agreed upon (Bogner et al., 2020). In particular for large and ever-changing applications such as Canvas LMS, this effectiveness has yet to be proven.

## 1.2 Problem Statement

In the field of web application development, React is a highly recognized framework which provides for animated and high-performance UI (user interfaces). But as the complexity of web applications increases, so does the difficulty in maintaining them. With online technologies constantly evolving, developers of interactive and effective applications have many choices (Szymkowiak et al., 2021). In this diverse JavaScript world, the popular React library focuses on its view layers and implements a high-speed virtual DOM. Although usage is cost-free, developers must pick the appropriate technology for their projects. The question is whether React can be used to build high-performing, scalable and stable architectural patterns used in Canvas LMS (Satrom, 2018).

Although React has become the nearly universal standard for modern Web development, there is a serious lack of knowledge in regard to systematic evaluation of architecture patterns within applications specifically from maintainability perspective (Banijamali et al., 2020). Although there are numerous studies on React and a myriad of architectures, the lack of research that tackles these patterns specifically in their application to systems like Canvas LMS makes it clear that further evaluation is needed. The objective of this research was to fill the gap between theory and practice by taking an extensive evaluation on Canvas LMS's architecture, really focusing on how React capacities influence maintainability (Lundberg, 2023). The study accentuation on giving practical and logical bits of knowledge to developers who used React-based architectural patterns for Canvas LMS.

## 1.3 Aims and Objectives

The aim of the proposed study is to assess React's capabilities for developing architectural patterns used in Canvas LMS.

* To evaluate the performance of React in building maintainable web applications.
* To identify the advantages and disadvantages of using React for building architectural patterns used in Canvas LMS.
* To explore the use of React in building architectural patterns used in Canvas LMS.

## 1.4 Research Question

* How effectively can React be used to develop and assess the maintainability of architectural patterns within the Canvas LMS environment?

## 1.5 Scope of research

This research examines React and architectural trends in Canvas LMS. The purpose is to thoroughly assess architectural choices for improving Canvas LMS, a popular education platform (Takala, 2023). The study examined Canvas LMS architectural patterns and their effects on web application maintainability. In Canvas LMS, component architecture, state management, data stream, and other React development topics must be thoroughly examined (David, 2020). The study compares architectural patterns' maintainability strengths and shortcomings. A detailed review of how architectural decisions affect codebase adaptability, analysis ease, and application ability to meet shifting requirements is required (Márquez and Astudillo, 2018). Additionally, this study clearly outlines its scope and impact. The research's insights and recommendations can benefit React application developers, especially in education. This article targets individuals implementing React for Canvas LMS (Peña‐Ayala, 2018). This study shed light on architectural choices in online application development, particularly educational technology.

## 1.6 Dissertation structure

* Chapter 1: Introduction: It introduced the research problem, objectives and scope.
* Chapter 2: Literature Review: This chapter examined React framework web application maintainability and architectural patterns used in Canvas LMS.
* Chapter 3: Methodology: It provides detail the research design, data collection, analysis methods and ethical considerations.
* Chapter 4: Design and Implementation: This chapter discusses design choices, implementation details, integration of architectural patterns in Canvas LMS.
* Chapter 5: Results: This chapter provides analytical findings while correlating results with research objectives.
* Chapter 6: Conclusion: This chapter summarises key findings.
* Chapter 7: Recommendations: This chapter provides practical guidelines for developers, suggest industry best practices and outline potential areas for further exploration.

# CHAPTER 2: LITERATURE REVIEW

## 2.1 Introduction

This chapter firstly discusses the performance and capability of the react in building of maintainable web application. It further discusses its advantages and disadvantages especially in building architectural patterns which are used in Canvas LMS. Further, the chapter discusses the use of react in building architectural patterns used in canvas LMS. Lastly, the theoretical framework and conceptual framework is provided.

## 2.2 React performance and capability in building maintainable web applications.

React's prominence in building viable web applications is irrefutable. The part-based engineering of React, virtual DOM and unidirectional data stream empower developers to make secluded, reusable, less complex, and adjustable code. React code redundancy and complexity are a vital worry in Canvas LMS (Flovén, 2020). In contrast, components of React advance modularity, the sheer volume of code inside an enormous application can prompt copied functionalities and multifaceted part connections. This makes code harder to explore and refactor, affecting viability. To address this, developers can use code-dividing strategies to separate the application into more modest pieces stacked just when required (Islam et al., 2022). Also, libraries like React.memo can be utilised for part memoisation, decreasing pointless re-delivers and supporting execution.

Another critical point is React state management. The application state is managed in a manageable and centralised manner by libraries such as Redux and MobX. Managing the global state across complex components becomes an issue in Canvas LMS. Redux also brings clarity and predictability with its unidirectional data flow and a single source of truth (Kudiabor 2020). However, its standard code and steep learning curve can be obnoxious to learners, in contrast to MobX which gives a more direct reactive methodology, whose adaptability may sometimes prompt misuse and accidental secondary effects. Which library to pick relies upon the necessities of a Canvas LMS improvement group and how much state management is required (Robinson et al., 2021).

As well as arranging code and utilising libraries, React's modularity is essential to maintainability. Code coherence and reusability improve when components are isolated into more modest, more engaged units with evident limits. Modularity gives the Canvas Learning Management System (LMS) a few capacities and capabilities. Canvas LMS is more successful and adaptable because of its secluded design. This method encourages code cleanliness and developer collaboration. Canvas LMS should consider React's performance. The virtual DOM is efficient, but nesting components and maintaining complicated states can cause performance concerns. These issues can cause performance constraints. Memoisation, code splitting, and lazy loading can improve performance. Strategically using these methods may increase software efficiency and effectiveness. Developers can reduce duplicate computations by memoising expensive function call results and retrieving them when needed. Code splitting lets developers break up their codebase for faster loading and better user experience. Lazy loading reduces initial loading times and optimises performance by loading components and resources only when needed. Include these strategies in their development process and employ profiling tools to discover performance bottlenecks and guide optimisation (Kainu, 2022). Careful consideration is needed to balance performance and maintainability within Canvas LMS's requirements.

Furthermore, React's unidirectional data stream guarantees an unmistakable and unsurprising state management instrument. Utilising a single source of truth for the application state upgrades its maintainability by diminishing the probability of bugs and irregularities emerging from complex data connections (Le, 2021). Developers can follow and investigate all the more easily, smoothing out the support cycle and limiting the risk of presenting accidental aftereffects during updates or adjustments. Canvas LMS fills in as a certifiable contextual investigation for assessing React's viability in building viable web applications. The Learning Management System's many-sided design requires a structure that can flawlessly deal with the complexity of instructive stages (Adkins et al., 2020). React's capacity to deal with the state and productively update the UI adjusts well to the prerequisites of such systems, where continuous updates and associations are vital. By taking apart the architectural patterns utilised in Canvas, LMS can recognise best practices and likely traps, offering essential bits of knowledge for developers looking to upgrade the maintainability of their web applications (David and Syriani, 2023).

## 2.3 Advantages and disadvantages of using React for building architectural patterns used in Canvas LMS.

### 2.3.1 Advantages

Utilizing React to apply Canvas LMS architectural patterns has many advantages. It improves on the making of web applications that are simpler to keep up with and construct. React's clear and coordinated component-based component building strategy is one of its core strength (Nguyen Nhat, 2018). Developers can make an efficient and reusable codebase by dismantling the UI into measured components with novel capabilities. Isolating a system into its segments works on its creation and support. Changing a component of an element does not influence the whole substance, making it simple to keep up with and continually modern (Vasilakis et al., 2018).

React's virtual DOM system is one more fundamental advantage for building Canvas LMS architectural patterns. The virtual DOM restricts the prerequisite for customary updates to the actual DOM, smoothing out execution by unequivocally conveying simply the parts influenced by a change of state (David, 2020). This efficiency is particularly crucial for a Learning Management System like Canvas, where continuous communications and updates are inescapable. The better show ensures a smooth client experience and all the while diminishes the probability of execution-related issues that could obstruct the overall maintainability of the application (Chung et al., 2023).

React's unidirectional data flow simplifies state management and provides a consistent framework for handling data across the application. This method helps detect and fix mistakes and ensures that the codebase can be handled and updated in the future (Evergreen, 2023). Developers can better track and understand data flow, reducing application faults and inconsistencies. This benefit is crucial in a sophisticated system like Canvas LMS. Strong state management is essential in handling a wide range of user interactions and massive data operations.

### 2.3.2 Disadvantages

While React offers areas of strength for a for building maintainable web applications, its utilization in creating architectural patterns for Canvas LMS isn not without certain limitations. One surprising drawback begins from the natural complexity that can arise in React applications as they scale (Thomas et al., 2018). The specific part-based approach, while priceless for affiliation and reusability, can sometimes provoke astounding dominance hierarchies and associations between parts. Concerning Canvas LMS, a tremendous and diverse educational platform, managing this complexity transforms into an earnest idea. The versatile interdependencies between parts may conceivably confuse or further complex the overall system and, consequently, impact the maintainability of the application over the long term (Seaborn et al., 2021).

React for Canvas LMS also requires certain expertise and experience. React's component-based architecture has many advantages, yet it requires a change in perspective in UI design which can only be adhered by expereienced developers (Derakhshanmanesh et al., 2019). For projects with tight cutoff times or deadlines, gathering developers who are skilled might be a vital issue. React's ability to manage data and application state can have positive and negative effects. The framework lets developers use local state, the context API, and external state management libraries to manage application state (Coronado et al., 2022). This technique is flexible enough to satisfy many project goals, but it can generate problems if not handled correctly. A consistent and systematic approach to information management in Canvas LMS is crucial (Estill, 2019). Different job techniques may impair teamwork and maintenance troubleshooting.

## 2.4 Use of React in building architectural patterns used in Canvas LMS.

Canvas LMS, a significant and vigorous learning management system, is significantly beneficial because of its accentuation on flexibility and personalisation. The complex cluster of structures and user interactions (UI) requires solid and strong design, which React effectively offers. Its natural capacities make it an ideal decision for making sensible components inside Canvas (Peng et al., 2019). The component-based architecture is a vital piece of React's supportiveness. Isolating the LMS into specific accumulated components is joined with Canvas' deliberate complexities. Each course, errand, or client profile is treated as an independent element in React, which works on the clarity and segment of code hence making it a suitable option to build Canvas LMS. Developers can investigate and change individual components without directly affecting the general system (Kruk et al., 2018). This modularity advances joint effort by empowering many groups to work simultaneously on various platform bits. It also works on the most common way of rectifying bugs and changing the platform's elements.

Unidirectional data flow is vital to React. User interactions with Canvas and API requests cause complex changes to numerous components. Controlled flow provides essential predictability (Paul and Nalwaya, 2019). Redux provides an authoritative and centralised application state repository. This prevents program-wide inconsistent states and ensures components receive relevant changes. Despite its repetitive code, the centralised technique controls state management. This is important in complex systems like Canvas (Franklin and Duran, 2023). React excels at architectural patterns outside its components. Canvas used the Container-Presentational paradigm to separate user interface and business logic efficiently. This method helped isolate the application's essential functionalities from its visual presentation. The team used this paradigm to ensure that user interface changes would not affect business logic. Presentational components merely render; containers manage state and fetch data. This strategy clarifies roles and promotes code reuse (Lacerda et al., 2020). Programmers use Higher-Order Components (HOCs) to share capabilities amongst components efficiently. This strategy streamlines development and reduces code duplication, making it more efficient and understandable.

However, React's versatility can introduce challenges when applied to a tremendous platform like Canvas. Code redundancy might happen if modularity is not fastidiously controlled. Strategies such as memoisation and code parting are significant for advancing memory use and improving execution (Allioui and Mourdi, 2023). Moreover, managing worldwide states in a convoluted system, for example, Canvas, requires vast consultation. In spite of the fact that libraries, for example, Revival, offer association, extreme reliance on them can bring about bulky state trees and complexity. Striking a suitable balance between incorporated organisation and component independence is significant. It is vital to specify that React's exhibition is a basic calculation of the progress of Canvas (Cornito, 2021). It is pivotal to consider enhancing component refreshes, utilising slow stacking for highlights that are not utilised, and assessing execution bottlenecks. Improving data-getting methods and limiting unnecessary re-renders can enormously upgrade client experience in the asset-escalated setting of a Learning Management System (LMS) (Pham, 2023).

## 2.5 Theoretical framework

The Software Architecture Theory stresses the meaning of planning and coordinating software systems in a manner that aligns with the general objectives of maintainability, versatility, and flexibility. Regarding this study, the Software Architecture Theory gives a calculated focal point to examine and assess the architectural patterns utilized in Canvas LMS. This perplexing web application works with the React framework (Dragičević and Bošnjak, 2019). The central views of the Software Architecture Theory underline the requirement for a very organized and measured architecture to upgrade software systems' life span and maintainability. By utilizing this theory to look at Canvas LMS, the review attempts to sort out how the architectural patterns, impacted mainly by React, help or hurt the web application's general maintainability (Soares-Robinson, 2018). The theory directs the assessment of how the picked architectural patterns line up with standards like modularity, division of worries, and versatility, which are all crucial components in making maintainable web applications.

The study of Software Architecture Theory helps detect architectural compromises. This study examines how Canvas LMS architecture decisions affect application development and maintenance. According to Messinger and Öqvist (2023), architectural choices affect the development and maintenance of the Canvas LMS. The theory allows for formal analysis of Canvas LMS deployment, adaptation, and architectural pattern issues. Ralph (2018) proved software architecture theory and solidified software design notions. The material provides a complete foundation for determining software system standards. These standards help build and maintain sustainable, manageable web apps. This speculative framework examined Canvas LMS architectural patterns and showed how they help build a strong React web app.

## 2.6 Conceptual Framework

**React Capabilities**

React code redundancy and complexity

React state management libraries (Redux and MobX).

React Code organisation and modularity

**Maintainability of Architectural patterns used in Canvas LMS**



Dependent variable

**Independent variables**

## 2.7 Conclusion

In a nutshell, the component-based architecture is a vital piece of React's supportiveness The component-based structure of React, its use of virtual DOM and unidirectional data flow make it much better suited to code management and speeding things up. On the other hand, there are also several negative aspects. Code redundancy is one concern here and a learning curve for implementation may be another problem which needs to be addressed carefully. The Recovery and design principles applied to Canvas LMS are a demonstration of React’s flexibility. Developers can use code-dividing strategies to separate the application developers to strike a balance with React's versatility.

# CHAPTER 3: METHODOLOGY

## 3.1 Adopted Methods

### 3.1.1 Research philosophy

Research philosophy is the foundational beliefs, concepts, and norms that guide research. It includes the principles and attitudes that shape the research approach and methodology. Research philosophy informs judgements and supports study integrity and rigour by offering a firm foundation. Research design, system, and procedures affect the researcher's perspective (Abu-Alhaija, 2019). Positivism is a research philosophy that thinks observation and analysis may reveal data. It stresses objectivity, quantifiability, and generalizability. Positivism holds that accurate knowledge and appraisal can reveal significant information, according to Turyahikayo (2021).

Positivism helps create sustainable React web apps and evaluate architectural patterns in Canvas LMS. Positivism emphasises objectivity and logic (Aldaeej, 2021). Positivism helped academics objectively evaluate React's capabilities in Canvas LMS when studying architectural patterns. This evaluation included presentation, production, and viability. Experimental observation collects quantitative data to evaluate architectural patterns' sustainability effects (Li et al., 2021). Besides, positivism's emphasis on quantifiable data upholds the foundation of generalisable standards. In web application improvement, especially with React, the objective is frequently to recognise best practices that can be applied comprehensively (Sovacool et al., 2018). Positivism's obligation to speculation permits findings from the study to be extrapolated to other web advancement projects utilising React, giving significant bits of knowledge to the more extensive community.

The decision of positivism in this study is valid by the requirement for an organised and objective assessment of React's viability in building maintainable web applications, an errand innately lined up with exact perception and quantifiable measurements (Abdullah et al., 2019). The study includes evaluating explicit results, quantifiable and logical measurements, and the effect of architectural patterns on maintainability — all of which positivism obliges through systematic estimation and investigation (Gillani, 2021).

### 3.1.2 Research approach

Kapur (2018) defines the research method as the plan or strategy that guides the researcher in data collection, analysis, and interpretation to answer research questions. Different methods are used to conceptualise and draw conclusions from a study. The deductive research method collects and analyses information to test a theory. A general idea or hypothesis is followed by specific facts or observations (Stephens et al., 2018). Reasoning is advantageous due to its structure, ability to examine hypotheses, and potential for good internal validity. Preexisting ideas or hypotheses that need methodical testing work best.

For this study on React, deductive research is appropriate to construct sustainable web apps and Canvas LMS architectural patterns. Initially, it allowed researcher test organised hypotheses on React's ability to build maintainable web apps (Li, 2019). The study examines architectural patterns and maintainability. Logical methods work for systematically examining these theories. The deductive approach is ideal for testing theories with empirical evidence (Pearse, 2019). The deductive method helps online application developers collect data to evaluate software architecture and assumptions. This method uses current principles to ensure thorough testing. This project examines how React is used in Canvas LMS. React's efficacy is objectively assessed in this deductive investigation (von Grafenstein et al., 2022). The research is based on architectural pattern maintainability theories. Canvas LMS data is evaluated to test these hypotheses.

### 3.1.3 Research design

Research design is a framework for putting together and assessing data in a research project (Asenahabi, 2019). Quantitative research utilizes numerical data and statistical analysis to foster decisions about a phenomenon (Mohajan, 2020). This study utilizes composed surveys to procure numerical data for statistical analysis. Quantitative research has objectivity and flexibility in looking at a more extensive population, unambiguous circumstances and logical results and correlations. This research strategy works on statistical values, variable assessment, and theory testing (Schutt, 2019). WWeb apps that are easy to maintain are built with React, however Canvas LMS is built on architectural patterns which are complex and require logical understanding to comprehend. The researcher utilized React to inspect its effect on system maintainability (Filz et al., 2021). The quantitative methodology was used for systematic assessment because of code redundancy and other quantifiable elements.

Quantitative research designs provide informed speculation based on findings. The study examined the viability of utilising React to build maintainable web apps. The researcher collected numerical data to provide summarised findings that describe experiences beyond contextual analysis. This study used a quantitative research technique because it suited its requirements. React's maintainability must be assessed objectively using numerical data and statistical analysis (Becker et al., 2018). This approach formalises and comprehensively evaluates React's maintainability.

## 3.2 Data Collection and research instruments

Data collection is the calculated assembling of relevant data for a research examination. This includes carefully gathering and archiving data to respond to questions or test speculations. Primary data gathering includes gathering data straightforwardly from people or original sources for research. The research emphasised on providing insights to effectively use React for web applications and development of Canvas LMS designs. Researcher gathered data for the study by means of surveys. Standardised close ended questions were used to obtain data from a specified population.

Advantages of primary data collection, particularly through surveys, consolidate the ability to adjust research examinations to the specific research targets, procure real-time and numerical data and to assure the relevance of data to the study novel requirement to maintain its reliability (Bentley et al., 2019). Surveys provide a coordinated method for managing quantitative data, engaging the assessment of variables and the examination of patterns. In this review, surveys were directed to gather data clearly from developers related with building web applications using React inside the Canvas LMS systems. The benefits of considering primary data collection through surveys in this study included customization of study to target nuanced perspectives related with architectural patterns. This data is utilized to numerically assess React's showcase and maintainability in Canvas LMS for solid outcomes (Kulkarni, 2022). This invigorated system made sure that data matched research questions and objectives. Surveys were utilised to measure Canvas LMS React and architectural feasibility. A systematic examination of how React impacts maintainability was possible using the quantitative research procedure and study objectives (De Rosis et al., 2020).

Convenience sampling is a type of sampling that is classified as non-probability sampling technique in which rather than using a random or systematic technique of selection, researchers chose participants based on their willingness and availability to participate (Khan, 2020). It is a helpful and time-effective methodology, frequently decided because of down to earth contemplations, like simple involvement of relevant participants. In this study on building maintainable web applications utilizing React and assessing architectural patterns in Canvas LMS, convenience sampling was utilised. The choice to utilize convenience sampling was driven by the openness of participants inside the developer community (Zhang et al., 2020). 100 surveys were performed with experienced experts and industry specialists from the developer community. This strategy considered a timely collection of data from people promptly accessible and able to share their bits of knowledge, giving logical viewpoints on the utilization of React and architectural patterns in real-world web improvement situations.

## 3.3 Data Analysis

Data analysis comprises evaluating, organising, manipulating, and using data to get insights, support decision-making, and find insights. Research data analysis seeks to uncover patterns and trends (Huang et al., 2018). This helps with study findings and conclusions. Statistical tests analyse numerical data from smaller samples to gain insights into large populations (Bruce et al., 2018). These tests help researchers understand data relationships, variations and trends. Statistical tests produce accurate results, help understand significant populations, and guide data-driven decision-making.

In this study, the researcher utilised statistical tests to survey React web application advancement and Canvas LMS architectural example assessment data. Statistical testing were utilized to look at what architectural patterns mean for maintainability (Uttley, 2019). The study used regression analysis to look at the interaction of independent and dependent variables. This presented the opportunity to evaluate numerical variables affecting React web application assurance in Canvas LMS. Correlation analysis examined the strength and relationship of variables. But it was also necessary to disseminate correlations and recognize patterns in architectural styles (Pereira et al., 2018). The study placed an emphasis on using SPSS to guide statistical tests. Statistical analysis was done with SPSS because of its simple interface and broad range of statistical activities. It was utilised to precisely and proficiently perform regression and correlation analysis on the quantitative data (Abu-Bader, 2021). These tests were performed well with SPSS to break down test results, practical adequacy, and participant's responses to acquire reliable and significant data (Bhattacharya et al., 2023).

## 3.4 Professional, Legal and Ethical issues

Professional, legal, and ethical aspects were prioritised meticulously in this study. The involved participants were acknowledged and respected for their time and knowledge. Data protection and privacy issues were considered to avoid legal issues. Participants were informed about the purpose of the research, and informed consent was obtained beforehand. It was informed that the responses will be exclusively utilised for research purposes. Ethical concerns involved protecting study participants' privacy and anonymity purpose. For this, data was securely maintained and stored in a password-protected cloud storage, and access was only accessible to the researcher and supervisor. The data was securely maintained and used only for research purposes. Following these steps ensured that the study provided reliable and trustworthy information.

## 3.5 Project management

In this research, systematic project management was followed. The project started with a thoroughly planned structure, from establishing objectives and research questions to point-by-point planning for each step. Another essential function for task circulation was assuring the efficiency of research, preserving an attainable schedule and completing tasks on time. When designing a survey by collecting data or making statistical evaluations, the study first ensured that the questions developed were reviewed and accepted by the supervisor and that the statistical techniques were discussed before the actual execution of the results. The members remained on track, and convenient overviews were conducted utilizing common communication channels like emails. In-advance gatherings with the boss were held to assist remain on track, illuminate problems or issues emerging amid the investigate consider, and make particular changes (Holmlund et al., 2020). A comprehensive and efficient information administration framework was outlined to guarantee secure capacity and bolster of collected data. The accentuation was set on versatility and time administration to handle conceivable factors or delays that might emerge amid the work (Javed, 2023). With this agile approach, they guaranteed that the extend remained on course whereas too considering moral.

# CHAPTER 4: DESIGN AND IMPLEMENTATION

### **4.1** Implementation**:**

A key player within the e-learning ecosystem, Canvas may be a learning administration framework (LMS) that employments computer innovation and internet-based information delivery to carefully alter conventional classroom encounters. Since its initiation within the late 1990s, learning administration frameworks (LMS) have advanced from recording course information and overseeing computerized fabric to giving progressed learning situations with a wide run of highlights and smooth network with other applications.

By permitting schools to customize advanced learning situations to meet their particular prerequisites, Canvas LMS streamlines instruction and moves forward learning. With more than 30 million clients spread over 4000 educate around the world, Canvas offers solid third-party application interaction and flexibility, empowering educate to tailor the stage with fundamental highlights. Canvas, made by Instructure, Inc., is utilized in both higher instruction and K–12 settings.

 React is a front-end programming framework that improves maintainability and reusability through modular components, first announced by Canvas in 2016. Reusable React components were made available to the open-source community by Instructure through the instructure-ui library on GitHub. The study object for this thesis will be a particular area of the Canvas LMS, with an emphasis on the platform's flexibility and ongoing development in response to technology breakthroughs.

An openly available API is a component of Canvas LMS that is necessary for providing dynamic data to the platform's client side. Through the use of a "access token," authentication enables communication between web apps and the API. The client-side application obtains access by adding/api to the Canvas instance URL, for example, https://<canvasinstance>.instructure.com/api/. However, applications with different domains have difficulties due to the same-origin policy, which limits default access and is implemented in XMLHttpRequsts and the Fetch API. The thesis suggests employing a proxy server for outside apps looking to access APIs in order to solve this. There are two types of APIs that Canvas LMS supports: GraphQL and REST. Interestingly, the documentation focuses on GraphQL development for future work, which could prevent some functionality from being backported to the REST API.

In recent decades, software—a crucial technology—has evolved from specialized tools used in many sectors to solve problems to becoming an industry in and of itself. It is important because it makes information easier to obtain and changes the way we find, produce, exchange, and use information. The increasing need for information technology-based business processes puts commercial enterprises under more competitive pressure, which forces software to constantly change and adapt. This constant development is driven by customer requests, mistake fixes, and the need to adapt to new situations.

There are similarities between the software development process and conventional building methods. Determining the nature of the problem is the first stage in software development, just as choosing a home style comes before constructing. Similar to home architectural planning, design is essential in both fields. In order to ensure that software is implemented, tested, and debugged successfully, planning is essential. The outcome of a project is frequently established before it is ever built, and knowledge of the subject improves one's capacity to make wise planning decisions. Like building buildings, sophisticated software depends on inflexible methodologies and solutions to provide dependable outcomes. When creating complex software systems, these defined solutions—referred to as patterns—are crucial. In general, the ideas and procedures involved in conventional building construction are mirrored in the creation and production of software.

Software patterns, in Brad Appleton's words, are an attempt to create a common knowledge base within the software community that would help developers deal with problems that come up again and again when developing software. These patterns create a common vocabulary for exchanging knowledge and firsthand accounts of similar issues and how to resolve them. Formally recording these solutions and their relationships results in the creation of an extensive body of knowledge that defines efficient structures that satisfy user requirements. Essentially, patterns allow software developers to share knowledge by facilitating the reuse of solutions to recurring issues and improving the understanding of sound architectures, which eventually results in the production of high-caliber software that meets end-user needs.

There are several categories for patterns. Software architecture, module organization, and data flow management are the main focuses of architectural patterns. Certain architectural patterns are frequently used in online applications. Issues with the software system are dealt with at the component level using component patterns, also known as design patterns. The concept of design patterns in software was first introduced by the book "Gang of Four," which borrowed the idea from other fields like architecture.

Producing software of superior quality benefits software companies as well as end customers. For the company, this means less time spent on rework, which frees up developers to concentrate more on developing new applications because there are less bugs, less maintenance requirements, and fewer customer support issues. Users gain from an improved software experience, which increases their productivity when carrying out tasks. Achieving high quality requires making sure the problem's design and source code adhere to strict guidelines.

Before any code is created, the design, which specifies how the program architecture, data structures, and components should be built, is essential for ensuring software quality. Software architecture is the high-level part of design that outlines the main components, relationships, and data flow of the system. It provides a framework for preliminary choices, making it easier to think through potential system configurations. One of the most targets of high-quality program is to manage framework complexity. Typically fulfilled by compositionally isolating the program into littler, autonomous segments, a prepare known as seclusion.

In "Program Building:

A Practitioner's Approach," Roger S. Pressman puts out rules that bolster well-designed components, recognized engineering styles, and an developmental execution methodology. It is imperative to utilize recognizable engineering designs and styles, and the components ought to have well-designed highlights. Secluded plan advancement ought to streamline testing and support strategies. When combined, these rules offer assistance create computer program that's of fabulous quality.

The accentuation moves to component-level plan, which investigates the complicated structure and intelligent of components, after the engineering system of a framework has been set up. This approach brings the framework one step closer to its real execution.  Cohesion, in which components should be as autonomous as feasible, carrying out single functions and requiring as little contact with others as possible, is crucial for producing high-quality component design. This is in accordance with Dijkstra's Separation of Concerns design philosophy, which isolates system features in order to simplify maintenance and lower complexity.

A key component that divides the system into manageable parts is its modularity. According to Myers, software's conceptual manageability depends on its modularity. A high-quality software design should ideally have less coupling, or the connecting between units. A certain amount of coupling is required for the system to operate, but too much connection between modules makes things more complicated and more difficult to test, maintain, and deploy.

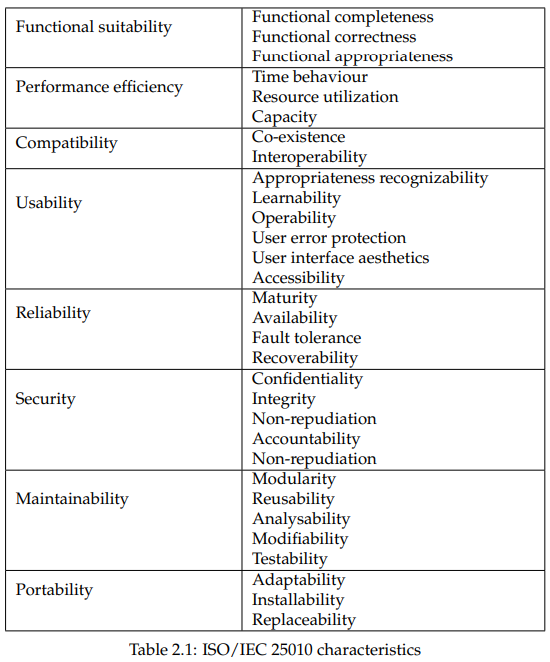
Software engineering goals and business goals are tightly related, particularly when the business is the software product. Improvements need objective measures of the quality of the source code and design models. Because software quality varies depending on the situation, it must be clearly defined and trade-offs between conflicting features must be made in order to meet organizational objectives.

Software attributes might be internal (focused on engineering) or external (conscious of the user). Software engineers are concerned with internal characteristics, which impact ease of modifications, functionality addition, mistake repair, and performance enhancement. External qualities include user experience and desired functionality. External quality is influenced by internal quality; unreliable software that is not well-maintained and designed lowers user pleasure. While admitting their implicit influence on exterior qualities, the main emphasis of this thesis is on internal quality attributes.

The ISO/IEC 9126 standard was made to supply fundamental program quality criteria. These six perspectives are:

usefulness, unwavering quality, ease of use, effectiveness, viability, and compactness. These characteristics work as diagonal pointers of framework quality, encouraging the foundation of a common understanding of the objectives and needs of a framework. A more later standard, ISO/IEC 25010, took the part of ISO/IEC 9126 in 2011. Its scope was broadened by including eight item quality characteristics and thirty-one sub-characteristics. ISO/IEC 25010 may be a successor standard that looks for to recognize related quality qualities which will be utilized to characterize necessities, fulfillment criteria, and related measurements. The eight qualities recorded in ISO/IEC 25010 are:

Useful Reasonableness, Practicality, Unwavering quality, Convenience, Security, Compatibility, and Compactness. In expansion to giving enlightening for assessing and upgrading numerous perspectives of program frameworks, both benchmarks are basic in making a intensive information of program quality.



In the software development process, objective analysis and quality control depend on quantitative assessments of program quality. These metrics make it easier to evaluate crucial aspects of quality and to examine how modifications to the system affect overall quality. A metric is a quantitative assessment of a system's ownership of a certain attribute, according to the IEEE Standard Glossary of Software Engineering Terminology. Metric-derived indicators provide useful data to improve software systems. Measurements are made on the system, and metrics are customized according to the quality qualities that the system prioritizes.

The characteristics of successful software metrics that Ejiogu suggests are as follows: simplicity, empirical and intuitive persuasiveness, consistency and objectivity, uniform usage of units and dimensions, independence from programming languages, and the ability to provide high-quality feedback. Good metrics should be simple to understand and calculate, consistent in their output, independent of programming language specifics, and provide guidance for improving software quality. They should also be in line with intuitive concepts of quality qualities.

Measurements are a means of creating high-quality software by iteratively improving products on the basis of metrics that are generated. As the quotation highlights, "There is no true way to know if you are progressing if you do not measure. Measuring progress and guaranteeing continual improvement in software development are essential.

When working with huge codebases, when human computation of certain code metrics becomes problematic, automated static code analysis becomes vital. Since many metrics are expressed as mathematical formulas, static code analysis tools can automate them. Problematic code can be found by defining clear code metric limits and integrating automated analysis into the build process. This procedure makes it possible to report on the quality of the code, which facilitates in-depth analyses and possible system redesigns.

Javascript source code metrics are automatically calculated by Plato, a specialized analysis tool. It produces thorough reports and results visualizations in addition to computing metrics. Plato computes complexity data by using the escomplex package.

Conversely, Escomplex is a complexity analysis tool for Javascript abstract syntax trees (ASTs) that supports both the JSX syntax extension used in React and contemporary Javascript. A Javascript parser is used to create the AST, and then static code analysis is carried out to extract complexity information. The productivity and precision of evaluating and upgrading code quality are made strides by this robotized strategy.

The ISO/IEC 25010 standard characterizes practicality as a significant include of program quality that incorporates five sub-characteristics:

testability, seclusion, reusability, analyzeability, and modifiability. The degree to which a framework is composed of free parts that do not meddled as well much with one other when one of them changes is measured by its seclusion. The capacity of an resource to be utilized in other frameworks is measured by its reusability. Analysability measures how well a item can be analyzed with imperfections or the impact of arranged enhancements. Modifiability surveys how well and effectively a framework can be changed without making unused imperfections. Testability ponders how well and rapidly test criteria may be made and utilized.

Practicality, in its easiest shape, is the capacity of program to alter to novel settings, correct bugs, and involvement upgrades in response to alterations within the environment, utilitarian needs, or necessities. Measures of post-development upkeep operations, such the sum of time required for adjustments or blunder redresses, offer important data. Viability models utilize factors like as estimate, cohesiveness, coupling, and complexity to expect a system's viability all through improvement. Diminishing complexity is significant for moving forward viability since it permits for more focused on and viable enhancement, adjustment, and rectification activities when program is separated into littler, more sensible components.

**Metrics:**

A exhaustive appraisal of practicality measurements was carried out through observational examinations within the distribution "Computer program Measurements for Foreseeing Viability" composed by Marc Frappier and colleagues. "Computer program Building Measurements and Models" and "Characteristics of Computer program Quality," two foundational distributions within the field of program building, served as the premise for the evaluation criteria. The investigate highlights the significance of measures having both "Objectivity and Algorithmic" characteristics. The term "algorithmic" shows that a degree must be appropriately calculated utilizing an calculation and ought to not vacillate with regard to time, area, or spectator. Subjective estimations can deliver conflicting discoveries, hence objectivity is basic to guaranteeing reliable and dependable comes about.

The ponder looked at exploratory information that scholastics have discharged to affirm these measures and looked into the relationship between the measurements and practicality pointers. Out of the nineteen measurements that were assessed, fourteen were found to be suitable for execution. A closer examination was conducted on six measurements:

Lines Of Code, Cyclomatic Complexity, Module Coupling, Module Quality, Plan Complexity of Card and Agresti, and Computer program Science Exertion. These measurements were calculated in a few ways:

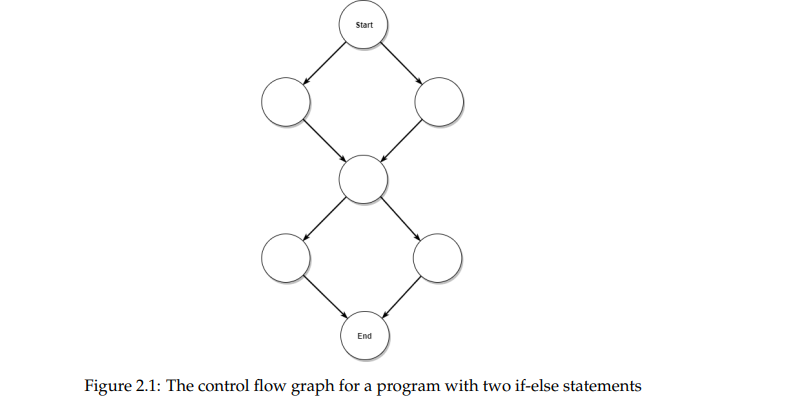
a few were calculated physically, whereas others made utilize of apparatuses for inactive code investigation. This careful examination gives shrewd data on choosing markers that precisely predict software practicality.

Lines Of Code

Lines Of Code (LOC) could be a measurement that checks the number of lines in a program, overlooking clear and comment lines, to decide its measure. Decreased LOC is seen to be a sign of superior viability. Inactive code examination devices may be utilized to computerize the computation of LOC. In spite of the fact that LOC offers a basic way to gage framework estimate, a tall LOC score as a rule demonstrates a difficult-to-maintain framework.

A prevalent degree for assessing the basic complexity of a program's control stream is cyclomatic complexity. The program is spoken to as a chart with hubs that speak to program explanations and edges that appear the control stream between explanations, which is set up by conditional explanations. The chart is computed on this representation.

E ´ N + 2 ^ ρ is the equation for cyclomatic complexity (M), where E signifies the number of edges, N the number of hubs, and ρ the number of program exit focuses. A program comprising of two if-else explanations, for illustration, would deliver a control flow graph, and the cyclomatic complexity degree might be computed fittingly. This degree is valuable for deciding how complicated the control stream structure of a program is, which makes a difference with viability assessment.



Thomas McCabe set up the Cyclomatic Complexity degree in 1976, and it has since appeared to be a valuable instrument with a wide run of employments in program advancement. McCabe emphasized its significance in prognosticating imperative points of interest with respect to the constancy and upkeep of program frameworks through computerized source code examination. All through the program extend, these measurements give progressing input, which makes a difference to oversee plan endeavors. They give comprehensive bits of knowledge into program modules, supporting within the recognizable proof of any unsteady zones all through the testing and upkeep stages.

Cyclomatic Complexity has been well approved and appears a outstanding affiliation with practicality. Program practicality is emphatically related with a lessening in cyclomatic complexity. Interestingly, certain examinations within the consider by Marc Frappier and colleagues uncovered that programs tended to have less botches when their Cyclomatic Complexity score was higher. This apparently conflicting result was clarified by the hypothesis that bigger modules—which are regularly connected to higher levels of complexity—were modified more carefully than littler ones. This realization bolsters the complex interface between cyclomatic complexity and computer program quality by advertising a comprehensive perspective on how it influences mistake rates and practicality.

**Coupling and cohesion:**

A key idea in software engineering, coupling quantifies the degree of interdependence between modules in a system by measuring the strength of interactions between them. This reliance may be directly observed between modules, with global data, or with the outside world. Less parameters and public methods indicate loosely connected modules, which are better since they are more flexible and make testing and reuse easier. Systems that are closely linked, meaning that modifications to one module would necessarily affect other modules that are connected, can be difficult to manage and adjust.

One particular metric of coupling that may be manually calculated using a checklist is module coupling, which was first proposed by G.J. Myers. Content coupling, common coupling, control coupling, and other links between modules are among the six types of interconnections that Myers sees as increasing coupling. Despite the absence of correlation data, the software engineering community has come to adopt this statistic widely.

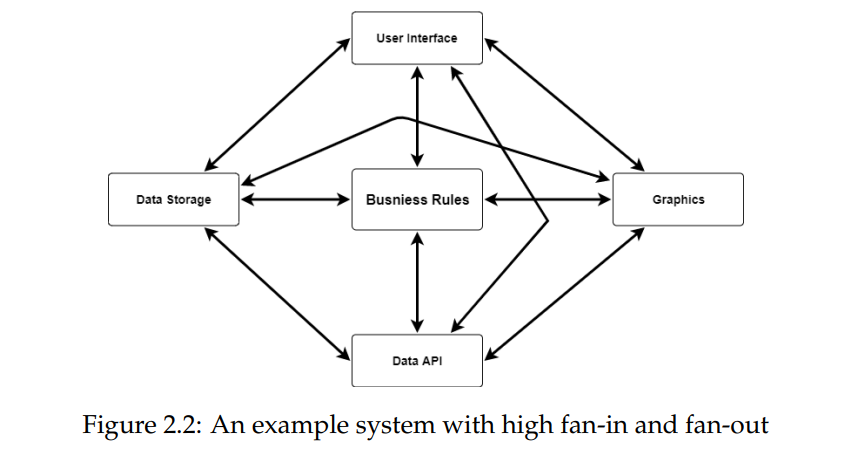
Conversely, cohesiveness quantifies the degree to which components inside a module are firmly connected. Strongly cohesive modules with tightly linked pieces are better in terms of maintainability, understandability, and reusability. Systems with cohesive modules often have less coupling since they depend on fewer other modules. According to Stevens, Myers, and Constantine, coherence is quantified by a checklist that includes kinds including functional cohesion and coincidental cohesiveness. In order to evaluate how connected the components are to one another inside a module and to create software systems that are both modular and maintainable, the term "cohesion" is essential. Comprehending coupling and cohesion offers noteworthy points of view on framework design, helping in accomplishing an balance between interconnecting and independence for the leading conceivable computer program quality and viability.

A careful assessment of the complexity of methods interior modules and their linkages inside a framework is given by Card and Agresti's Plan Complexity degree, which considers strategy calls, parameter passing, and data use. This degree is based on the full number of modules within the framework, the number of fan-outs, and the input/output factors from each module. As a result, it offers an assessment of a system's coupling and cohesiveness, making it an vital device for surveying the complexity and viability of frameworks.

This estimation, which was to begin with displayed by D. N. Card and W. W. Agresti, illustrates its flexibility by empowering assessment of the complexity and practicality either some time recently to framework establishment or after it has been put into put. Since of its normalized nature, it gives a reasonable normal complexity calculation for numerous frameworks, permitting for cross-system and estimate comparisons.

Framework complexity may moreover be caught on through the utilize of basic measurements related to module interconnects, such as fan-in and fan-out. Fan-in is the number of modules that call a certain module, whereas fan-out is the number of modules that the module calls. These measures shed light on how perplexing module interlinking could. Be a exceedingly connected module is demonstrated by a tall fan-out value, while an independent, self-sufficient module that's straightforward to preserve is proposed by a moo or zero fan-out. On the other hand, a tall fan-in esteem indicates a module that's broadly utilized by others, illustrating compelling framework reusability.

By the by, as Figure 2.2 outlines, a framework with high values for both fan-in and fan-out demonstrates a complex interconnection of modules, making it troublesome to preserve or alter when changes are presented to one module. This highlights the importance of striking a adjust between interdependency and autonomy in framework plan.



**JavaScript and ECMAScript:**

High-level programming dialect JavaScript was to begin with made to compose basic browser-based rationale. It has presently created into a adaptable device for making complex client-side and server-side online applications. It finds utilize in desktop apps created utilizing Electron and portable applications created with React-Native, in expansion to web advancement. JavaScript may be a programming dialect that works with browsers' Record Question Demonstrate (DOM) and doesn't require unequivocal compilation. Instep, it depends on the have environment for highlights like organizing and design rendering. JavaScript has first-class capacities that are fundamental for utilitarian programming, counting higher-order capacities, and underpins basic, useful, and event-driven programming approaches. It takes after the ECMAScript standard set up by Ecma Universal; eminent upgrades, such as syntactic sugar for course affirmations and module sharing, are included in ECMAScript 2015, which moves forward the language's capacity to make perplexing applications. With the ES6 lesson, JavaScript advertised a more conventional strategy for object creation and legacy, though being prototype-based.

Each major browser features a diverse JavaScript motor, which shifts depending on the have environment and translates JavaScript. Strikingly, the Node.js runtime environment and Chrome and other Chromium-based browsers are fueled by Google's V8 motor, which is broadly utilized. Utilizing V8, Node.js gives a valuable runtime for JavaScript to be executed exterior of the browser; this is often valuable for server-side errands and creating web apps. The Hub Bundle Supervisor (NPM),  included with Node.js, makes dependency management and project setup easier via the package.json configuration file. NPM gives developers access to a huge library of third-party packages, which improves their skills in many areas, most notably the creation of React applications.

**Web Application:**

A web application, sometimes known as a "WebApp," is a type of network-centric software that started off simple with static pages and has now evolved to include computation and dynamic content. By entering a URL, users may view webpages by pointing their browser to a web server. Web applications have historically relied on servers to get static files instead of having client-side architecture. But improvements in browser capabilities resulted in a shift of functionality to the client-side.

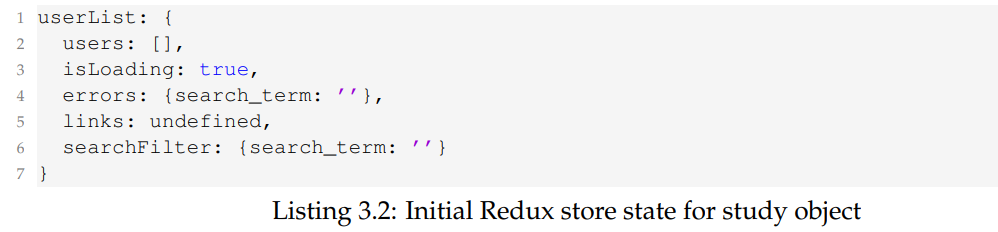
Powell defined WebApps as a fusion of software development, publishing, and several domains. They need both non-technical and technical work in design, usability, architecture, and functionality. WebApps, in contrast to traditional programs, are known for their rapid iterations, continuous evolution, short time-to-market, and emphasis on high maintainability for prompt updates and error repairs. Rich online experiences were once made possible by technologies like Adobe Flash and Microsoft Silverlight, but they have faded and been replaced with JavaScript-based alternatives. An early library called jQuery made interactive web pages possible. Building JavaScript apps and producing uniform HTML, CSS, and JavaScript for browsers has become the new paradigm. Web engineers rely heavily on frameworks and libraries to reduce the amount of code required by meeting time-to-market requirements and guaranteeing high-quality solutions for intricate features.

**Initial Implementation:**

The create-react-app tool was utilized to generate the development environment for the study's aim, as elaborated in Section 2.6. After that, all created source files were deleted, with the exception of index.html and index.js, which are the entry points for the program. The users' administration page's source code was first implemented by adding it to the application's src folder.

The Redux architectural pattern is used by the application. Using the redux-thunk middleware to handle asynchronous operations, the createStore function from the redux library is used to create the Redux store. Two input parameters are needed for this method: a reducer and the store's starting state.

Listing 3.2 shows the starting state, which consists of an object named userList with five fields: users, isLoading, errors, links, and searchFilter.



The other variables hold local data, but the program retrieves distant user data from the Canvas LMS API. Since they were unnecessary, a few fields pertaining to toggling between the user, account, and course pages were eliminated from the original version. As the only reducer, the rootReducer modifies the store according to actions that are sent using the Flux pattern. The actions are as follows: UPDATE\_SEARCH\_FILTER (modifies the search filter), SEARCH\_TERM\_TOO\_SHORT (handles a too-short search term), LOADING\_USERS (updates the isLoading state), GOT\_USER\_UPDATE (updates a single user), and UPDATE\_USERS (updates the state with new users).

The Redux store is added to the UsersSearch component as a property, enabling the user interface to access it. The UsersPane component, which is essential for handling store data, receiving updates, and initiating actions, receives this store after that. Using the setState() function of React.Component, UsersPane retrieves and stores the state of the Redux store. Lifecycle methods that deal with store subscriptions and unsubscribing include componentDidMount and componentWillUnmount.

Two actions are sent out after mounting and subscribing: applySearchFilter, which starts the first user fetch, and updateSearchFilter, which synchronizes the searchFilter state with query parameters from the URL.

Using the UsersStore object, the applySearchFilter action creator starts asynchronous user fetching from the Canvas LMS API and maintains the search filter state. It carries out the following duties:

obtains the search filter's current condition.

Verifies the search term's validity by commencing user fetching if it is valid or sending an action if it is too short.

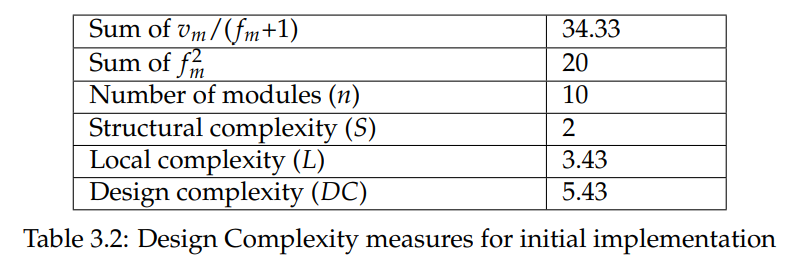
When a search term is too short, the UsersToolbar component receives a payload with an error message and an action called displaySearchTermTooShortError of type SEARCH\_TERM\_TOO\_SHORT.

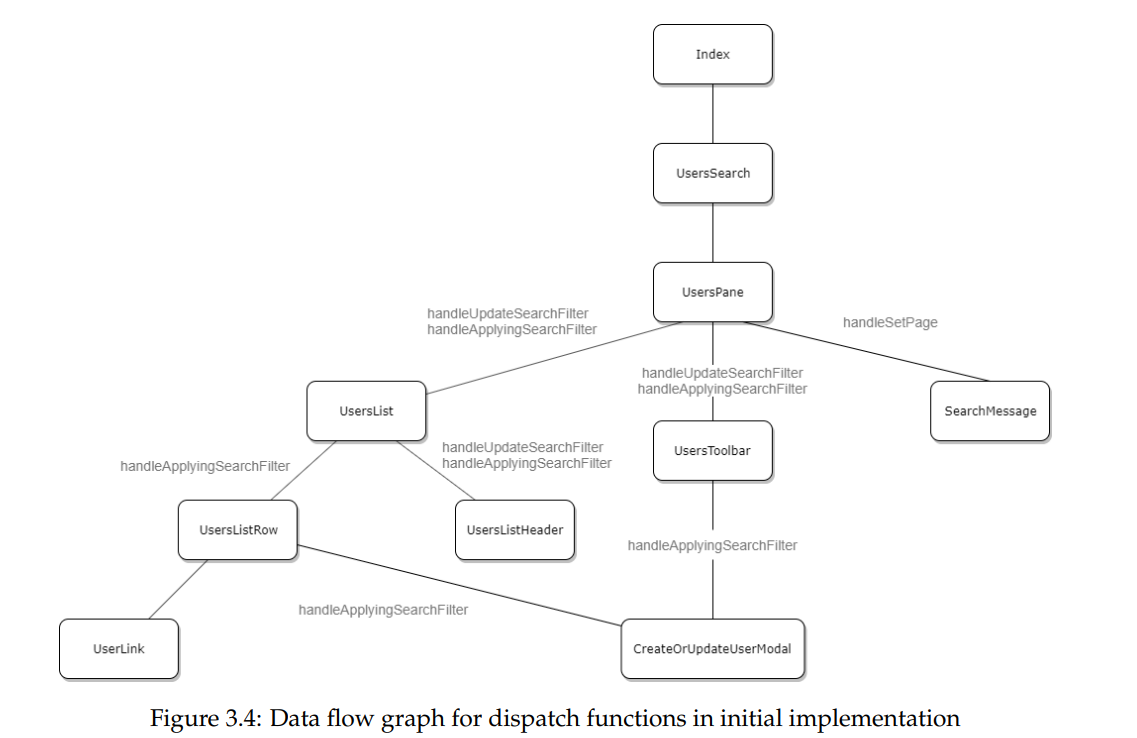
An action named loadingUsers with the type LOADING\_USERS and an empty payload is sent if the search term is legitimate. The asynchronous user fetching is then started, and the fetched users are sent to an action named gotUserList of type GOT\_USERS along with a payload that contains the users and an XMLHttpRequest object with pagination links.

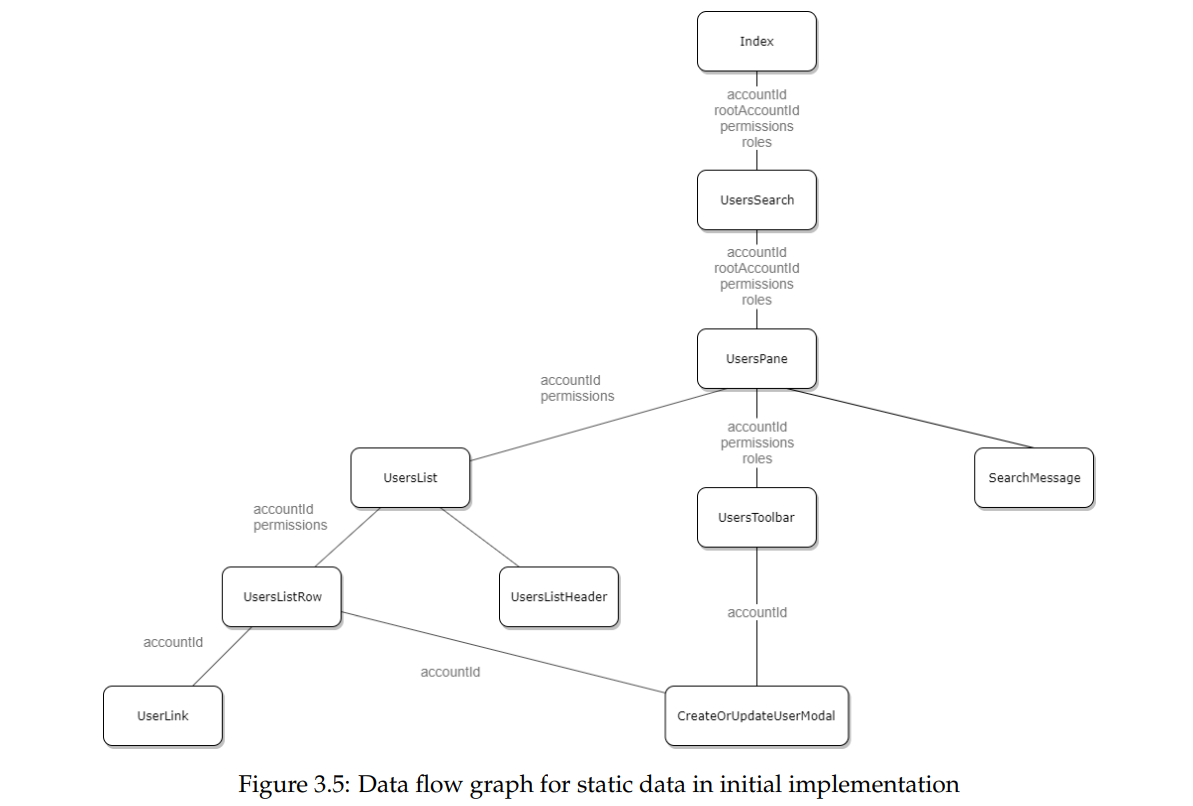
The UsersStore object, which was made using the createStore method, is in charge of fetching users. The UsersStore gains essential functionality from the createStore constructor, which may be improved with custom attributes. It lists methods such as normalizeParams for filtering and adding default parameters, and getUrl for getting URLs. The Ajax method of jQuery, enhanced for error handling, authentication, and JSON response processing, is used in the load function of UsersStore. The study object's superfluous functionality was eliminated for simplicity.

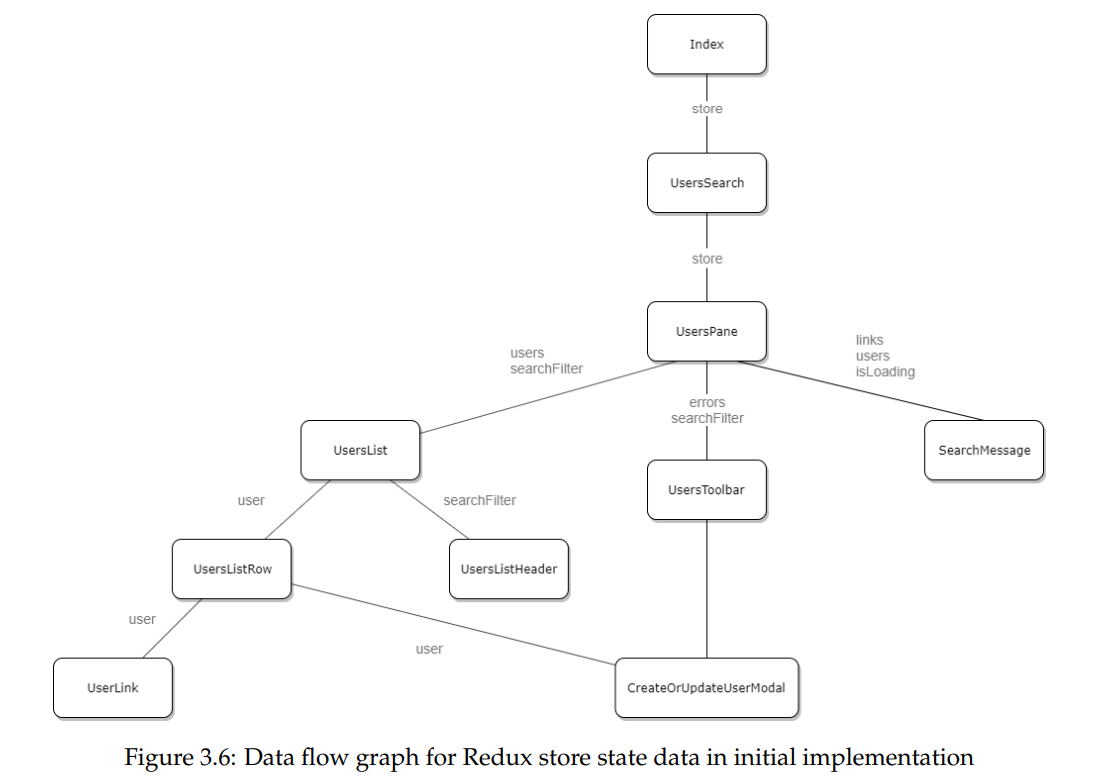
Metrics that were chosen for their applicability in assessing the application's design and Plato compatibility were used to measure maintainability. The metrics used are as follows: Lines of Code, which indicates source code size and overall complexity; Cyclomatic Complexity, which gauges source code complexity; Software Science Effort, which assesses overall complexity and maintainability; and Design Complexity of Card and Agresti, which measures structural and local complexity.

The results of the first implementation are shown in Table 3.2 for Design Complexity. Manual computations were performed using the data flow diagrams (Figures 3.4, 3.5, and 3.6). Figure 3.5 shows the flow of static data, Figure 3.6 shows the flow of data saved in the Redux store state, and Figure 3.4 shows the flow of actions sent to the Redux store in the component tree. Three diagrams illustrating this data flow analysis allowed for a thorough comprehension and informed changes to the application's architecture during the Design Science process.

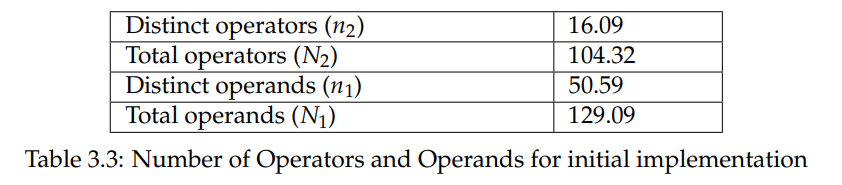


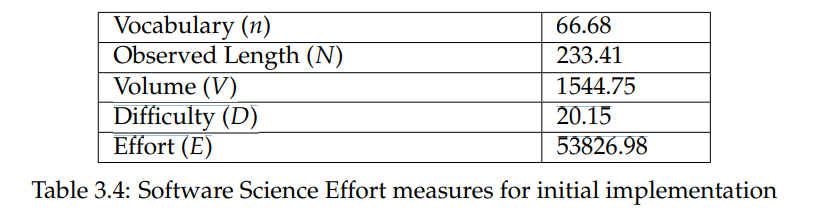




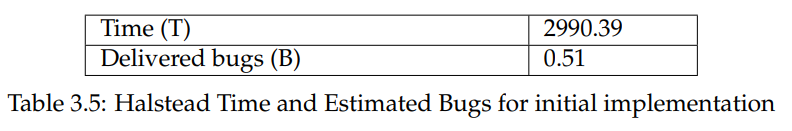


Using Plato and a custom script for per-file averages, the Software Science Effort measurement produced separate and total counts for operators and operands, as shown in Table 3.3. After that, average values for Software Science Effort were computed, as shown in Table 3.4. These values included metrics like Vocabulary, Observed Length, Volume, Difficulty, and Effort.

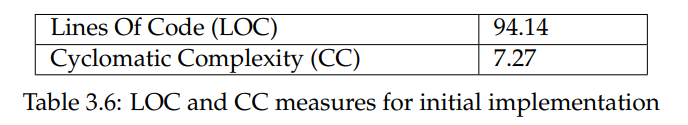




The amount of provided problems and implementation time were then estimated using the Software Science Effort findings, as shown in Table 3.5. Time (T) is determined at 2990.39 seconds, and Delivered bugs (B) is computed at 0.51.



Furthermore, Table 3.6 displays average Lines of Code (LOC) and Cyclomatic Complexity (CC) measures that were obtained through the use of Plato. A Cyclomatic Complexity of 7.27 and a LOC value of 94.14 were obtained from the first implementation.



# CHAPTER FIVE: RESULTS

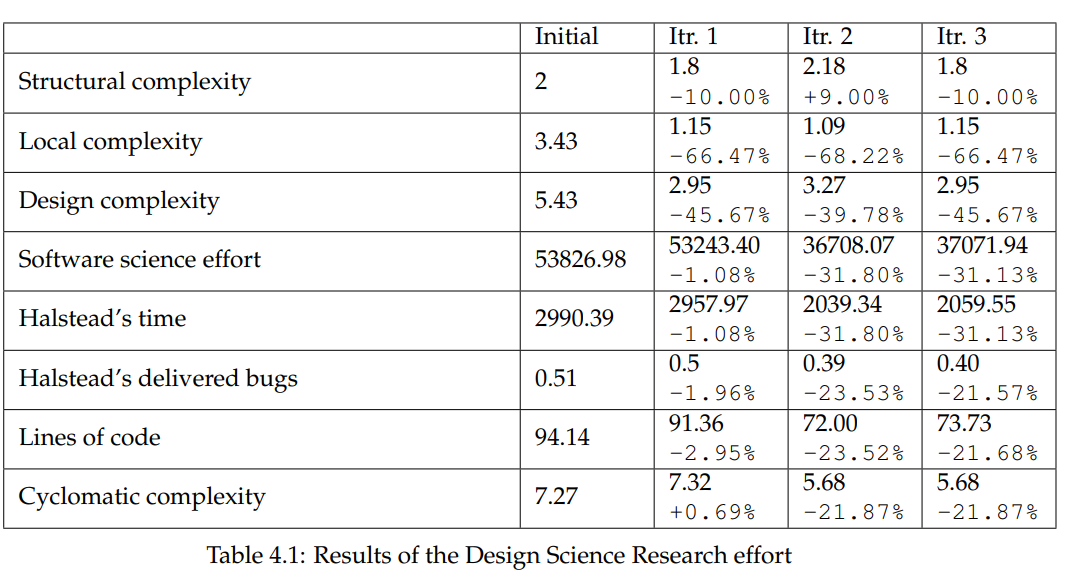
### 5.1 Introduction

A systematic redesign of the program using architectural patterns to lower the complexity of the source code and design resulted in an improvement of overall maintainability. Using the react-redux library in Iteration 1 resulted in a significant 45.67% reduction in Design Complexity; nevertheless, source code complexity remained high. In response to this issue, Iteration 2 introduced the Canvas LMS GraphQL API, which improves data collecting and updating flexibility through the use of targeted queries. After replacing the Redux store with Apollo Client and InMemoryCache, there was a significant improvement of 23.52% in lines of code, 21.87% in cyclomatic complexity, and 31.80% in software science effort as compared to the original version.

Iteration 2's usage of the React Context API to share static data between components resulted in greater coupling, which was resolved in Iteration 3. Coupling was reduced by 10% by storing static data in the Apollo Client Cache; nevertheless, additional queries and type definitions resulted in a little increase in source code complexity.

Cohesion showed the most gain, increasing by an astounding 66.47% in the final implementation. By storing data in a global Apollo Client Cache that is accessible via React Hooks, the initial problem of many input and output variables going through the component tree was addressed, resulting in decreased coupling and a 45.67% improvement in Design Complexity.

The results of the Design Science Research endeavor are compiled in Table 4.1, which also shows percentage changes from the first implementation for each metric. Please refer to Sections 3.4 and 3.5 for more information and thorough calculations.



### 5.2 Demographics

This chapter explores the results and the technique used, and then it talks about the ethical and social aspects of the thesis as well as source critique. The architectural patterns of React were implemented and evaluated using the iterative Design Science Research methodology. The first method used props-drilling in a unidirectional data flow, which had a detrimental effect on cohesion since it exchanged a lot of variables between input and output across the component tree. React-redux was added in the second iteration, which improved cohesiveness and decreased coupling. The complexity of the source code grew with the inclusion of Redux and react-redux, though. This was fixed in the third iteration by replacing Redux with Apollo Client and Context API. This led to significant gains in Cyclomatic Complexity, Software Science Effort, and Lines of Code. In the end, the architecture was improved by using Apollo Client Cache as a central source for data management, which provided better overall maintainability by giving equal weight to all metrics. Iteration 2 may be better depending on priorities in some cases because of somewhat less cohesiveness and simpler source code. The findings demonstrate a significant difference in maintainability between Redux, the most often used React architectural pattern, and Apollo Client, the second most popular option. It's interesting to note that Apollo Client is more maintainable than Redux, even though Redux has long been the go-to framework for complex React apps. This disparity may result from Redux's historical forerunner status and extensive use in intricate React applications. Redux's weaker maintainability may also be caused by its propensity to be combined with REST APIs on the backend. The popularity of Apollo Client may increase as GraphQL technology becomes more widely used and integrated across different APIs, suggesting a possible change in the recommended architectural patterns within the React ecosystem.

### 5.5 Discussion

The study's major focus is on the iterative Design Science Research method, which is used to assess and improve a React application's maintainability through the use of various architectural patterns. The study started by looking into React's built-in design, which makes use of props-drilling and unidirectional data flow. Because of the significant transmission of input and output variables via numerous levels in the component tree, this first design had a detrimental influence on cohesiveness.

Well known libraries like react-redux, Apollo Client, and Setting API were included in afterward modifications to unravel issues found within the unique design. Unwavering quality measures appeared outstanding picks up, particularly in Plan Complexity, Computer program Science Exertion, Lines of Code, and Cyclomatic Complexity. A noteworthy increment in source code complexity was apparent within the switch from Redux and react-redux to Apollo Client and Context API, highlighting the need of choosing the correct libraries depending on wants of the venture.

The study's discoveries demonstrate that building designs utilized in Respond apps have a enormous impact on practicality, with Apollo Client appearing more proficiency than Redux, which is regularly utilized. The repercussions of specialized changes are moreover examined, with the note that Apollo Client might ended up a more favored choice to Redux as GraphQL gets to be more well known. Within the conclusion, the consider highlights the iterative nature of plan changes and offers smart direction on how to form well-informed building choices to move forward the in general practicality of Respond apps.

### 5.6 Chapter summary

This chapter clarifies how a few structural designs may be utilized to make strides a Respond application's viability utilizing an iterative Plan Science Inquire about approach. The paper starts with an investigation of React's inborn design, centering on one-way information stream utilizing props-drilling and indicating out cohesiveness issues. In arrange to illuminate the issues raised, encourage adaptations join and assess well-known libraries such as react-redux, Apollo Client, and Setting API. The results appear outstanding changes in measures related to practicality, particularly in Apollo Client's case where it outperforms Redux. The repercussions of specialized changes are secured within the chapter's conclusion, wherein Apollo Client's conceivable rise to noticeable quality in response to GraphQL's developing utilization is expected. The iterative plan changes highlight how important it is to form customized building choices for Respond apps, advertising supportive counsel on how to maximize viability.

# CHAPTER SIX: CONCLUSION

In summary, this inquire about systematically explores and evaluates structural designs inside the system of making React-based viable online apps, with a specific accentuation on Canvas LMS. Through an broad prepare of Plan Science Investigate, the consider gives vital unused data on how different building choices influence measurements related to viability. The iterative strategy appears how the application's plan has created whereas highlighting the preferences and drawbacks of each unused engineering design.

The comes about highlight how critical it is to alter building designs to suit certain application prerequisites. Astoundingly, critical picks up in practicality measurements are watched when well known libraries like react-redux and Apollo Client are utilized rather than props-drilling. The differentiate between both libraries highlights how, indeed with Redux's widespread use, Apollo Client may be a more sensible option—especially when considering the selection of GraphQL. This ponder includes to our understanding of React application design and offers helpful exhortation to engineers trying to find the finest ways to form and oversee adaptable, adaptable web apps. Future headways can be guided by the evaluation's discoveries, which can offer assistance choose engineering designs that are particularly planned to progress the viability of web apps that are React-based.

# CHAPTER 7: RECOMMENDATION

Taking after a careful examination of structural designs in Respond web application advancement, a number of recommendations are made for designers and other partners working on related ventures.

To begin with off, Apollo Client appropriation ought to take priority over Redux selection given the famous viability benefits, especially with regard to GraphQL APIs. The discoveries appear that Apollo Client not as it were makes information getting more effective, but it moreover makes a difference to play down source code complexity, which makes codebases simpler to oversee. In addition, the iterative engineering alter presentation handle suggests that a cautious evaluation of each pattern's impact on cohesion, coupling, and complexity measurements is fundamental. The trade-offs of libraries and systems ought to be considered by designers, and those that best suit the specific needs of the application ought to be chosen.

The consider too highlights how pivotal it is to ceaselessly alter and progress building choices as the extend advances. Ceaselessly assessing the chosen designs agreeing to measurements like Lines of Code, Cyclomatic Complexity, and Program Science Exertion ensures that the plan remains in line with the objective of making strides viability.

At long last, the research's recommendations—which are particularly custom fitted to the Canvas LMS context—are implied to help designers in making well-informed choices and advance the advancement of solid and viable Respond web apps.

# References

Abelson, H. and Sussman, G.J., 1996. Structure and Interpretation of Computer Programs. 2nd ed. MIT Press.

Appleton, B., 2000. Patterns and Software: Essential Concepts and Terminology. [Online] Available at: http://www.bradapp.com/docs/patterns-intro.html [Accessed 20 Nov. 2019].

Boehm, B.W., 1978. Characteristics of software quality. TRW series of software technology. North-Holland Pub. Co.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., 1996. Pattern-Oriented Software Architecture - Volume 1: A System of Patterns. Wiley Publishing.

Canvas.instructure.com, 2019. Canvas LMS REST API Documentation. [Online] Available at: <https://canvas.instructure.com/doc/api/> [Accessed 20 Nov. 2019].

Canvas.instructure.com, 2019. GraphQL API - Canvas LMS REST API Documentation. [Online] Available at: <https://canvas.instructure.com/doc/api/file.graphql.html> [Accessed 20 Nov. 2019].

Card, D.N. and Aggresti, W.W., 1988. Measuring Software Design Complexity. J. Syst. Softw., 8(3), pp.185–197.

Community.canvaslms.com, 2018. InstUI: Instructure’s Style Guide 2.0 | Canvas LMS Community. [Online] Available at: https://community.canvaslms.com/community/ideas/blog/2018/11/16/instui-instructure-s-style-guide-20 [Accessed 20 Nov. 2019].

Conte, S.D., Dunsmore, H.E., and Shen, V.Y., 1986. Software Engineering Metrics and Models. Benjamin-Cummings Publishing Co., Inc.

Carmean, C.D.B. and Wagner, E., 2009. The Evolution of the LMS: From Management to Learning. The ELearning Guild Research.

Dijkstra, E.W., 1982. Selected Writings on Computing: A Personal Perspective. Springer-Verlag.

Ecma-international.org, 2019. ECMAScript 2015 Language Specification – ECMA-262 6th Edition. [Online] Available at: https://www.ecma-international.org/ecma-262/6.0/ [Accessed 20 Nov. 2019].

Ecma-international.org, 2019. Welcome to Ecma International. [Online] Available at: <https://www.ecma-international.org/> [Accessed 20 Nov. 2019].

Ejiogu, L.O., 1991. Software Engineering with Formal Metrics. QED Information Sciences, Inc.

Frappier, M. and Mili, A., Software Metrics for Predicting Maintainability. [Online] Available at: http://www.dmi.usherb.ca/~frappier/Papers/tm2.pdf.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc.

GitHub, 2019. escomplex/escomplex. [Online] Available at: <https://github.com/escomplex/escomplex> [Accessed 20 Nov. 2019].

GitHub, 2019. facebook/create-react-app. [Online] Available at: https://github.com/facebook/create-react-app [Accessed 20 Nov. 2019].

Graphql.org, 2019. GraphQL: A query language for APIs. [Online] Available at: <https://graphql.org/learn/> [Accessed 20 Nov. 2019].

Halstead, M.H., 1977. Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc.

IEEE, 1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990.

Instructure.com, 2019. About Canvas | Edtech Learning Platform | Instructure. [Online] Available at: <https://www.instructure.com/canvas/about> [Accessed 20 Nov. 2019].

Iso.org, 2019. ISO/IEC 25010:2011. [Online] Available at: <https://www.iso.org/obp/ui/#iso:std:isoiec:25010:ed-1:v1:en> [Accessed 20 Nov. 2019].

Iso.org, 2019. ISO/IEC 9126:1991. [Online] Available at: https://www.iso.org/standard/16722.html [Accessed 20 Nov. 2019].

Krasner, G. and Pope, S., 1998. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. Journal of Object-oriented Programming - JOOP, 1.

Leff, A. and Rayfield, J.T., 2001. Web-application development using the Model/View/Controller design pattern. In: Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference, pp.118–127.

McCabe, T.J., 1976. A Complexity Measure. IEEE Trans. Softw. Eng., 2(4), pp.308–320.

McCabe, T.J. and Watson, A.H., 1994. Software Complexity. CrossTalk, 7(12), pp.5–9.

McConnell, S., 2004. Code Complete, Second Edition. Microsoft Press.

MDN Web Docs, 2019. About JavaScript. [Online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About\_JavaScript [Accessed 20 Nov. 2019].

MDN Web Docs, 2019. Same-origin policy. [Online] Available at: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin\_policy [Accessed 20 Nov. 2019].

Myers, G.J., 1978. Composite/structured Design. Van Nostrand Reinhold.

Powell, T., Jones, D., and Cutts, D., 2019. Web site engineering: beyond Web page design.

Pressman, R., 2010. Software Engineering: A Practitioner’s Approach. 7th ed. McGraw-Hill, Inc.

Reactjs.org, 2019. Getting Started – React. [Online] Available at: https://reactjs.org/docs/getting-started.html [Accessed 20 Nov. 2019].

Reactjs.org, 2019. Integrating with Other Libraries – React. [Online] Available at: https://reactjs.org/docs/integrating-with-other-libraries.html#integrating-with-modellayers/ [Accessed 20 Nov. 2019].

Stevens, W.P., Myers, G.J., and Constantine, L.L., 1974. Structured Design. IBM Syst. J., 13(2), pp.115–139.

V8.dev, 2019. Documentation · V8. [Online] Available at: <https://v8.dev/docs> [Accessed 20 Nov. 2019].

Vaishnavi, V. and Kuechler, B., 2004. Design Science Research in Information Systems. Association for Information Systems.

Yourdon, E. and Constantine, L.L., 1979. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. 1st ed. Prentice-Hall, Inc.



**Ethical Approval Form**

* This form must be completed, signed and submitted with the Project Proposal.
* No work may be carried out on the project until the form has been submitted.
* Late submission will result in a penalty.
* Failure to submit the form will result in an automatic fail for the module. You may also be subject to disciplinary action.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Section 1 TO BE COMPLETED BY STUDENT** | | | | | | | |
| Name of Student: | |  | | | | | |
| Student No: | |  | | | | | |
| Course: | |  | | | | | |
| Module: | |  | | | | | |
| Project Title: | |  | | | | | |
| Summary of Proposed Project: | | | | | | | |
|  | | | | | | | |
| Planned Start Date: | |  | | Planned End Date: |  | | |
| **DECLARATION BY STUDENT:**   * I confirm that I have read and understood the Research Ethical Guidelines and agree to abide by them in conducting my project. * I confirm that I understand the importance of adhering to the Research Ethical Guidelines and I am aware of the penalties for breaching them. * I agree to notify my academic supervisor if there is a change to my project and/or further ethical approval is needed. | | | | | | | |
| **To the best of my knowledge, I confirm that:**   * There is no risk to any participants * There is no risk to me * There is no risk to the institution or QA in terms of liability or reputation | | | | | | | |
|  | I undertake to report all data and findings in a responsible way | | | | | | |
| **Name:** |  | | **Signature:** |  | | **Date:** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Section 2 TO BE COMPLETED BY SUPERVISOR** | | | | | | |
| Name of Supervisor: | | Dr Ajmal Gharib | | | | |
| **DECLARATION BY SUPERVISOR:**   * I undertake to review and approve any questions that the student intends to use for data collection, including interview questions and questionnaire items. | | | | | | |
| ON THE BASIS OF THE INFORMATION PROVIDED BY THE STUDENT, THE PROJECT: | | | | | | |
| X | **DOES NOT** need to be referred to the Faculty Research Ethics Committee for approval. | | | | | |
|  | **DOES** need to be referred to the Faculty Research Ethics Committee for approval. | | | | | |
| If the project needs to be referred to the Faculty Research Ethics Committee for approval, please explain why briefly: | | | | | | |
|  | | | | | | |
|  | On the basis of the information provided by the student, I confirm that the project will contain sensitive or confidential information and should **not** be placed in the public domain. | | | | | |
| **Name:** | Ajmal Gharib | | **Signature:** | A.Gharib | **Date:** | 29/Jan/24 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Section 3 TO BE COMPLETED BY SUPERVISOR** | | | | | |
| **CHANGES TO PROJECT – DECLARATION BY SUPERVISOR:**   * I have reviewed the proposed changes to the project. | | | | | |
| ON THE BASIS OF THE INFORMATION PROVIDED BY THE STUDENT: | | | | | |
|  | I **APPROVE** the revised project. | | | | |
|  | I **DO NOT APPROVE** the revised project. | | | | |
| If the revised project is not approved, please explain why briefly: | | | | | |
|  | | | | | |
| **Name:** |  | **Signature:** |  | **Date:** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Section 4 TO BE COMPLETED BY STUDENT** | | | | | |
| **CHECKLIST FOR STUDENT** | | | | | |
|  | I have fully completed this Ethical Approval Form and have signed where appropriate. | | | | |
|  | I have included a copy of any research instruments I wish to use (interview questions, questionnaires, etc.) in the Appendix of my proposal. If draft versions, I undertake to have the final versions approved by my supervisor before collecting any data. | | | | |
|  | I have included this Ethical Approval Form in the Appendix of my **proposal** so that it may be reviewed by my supervisor. The proposal outlines the research methodology I will use. | | | | |
|  | I have included this Ethical Approval Form in the Appendix of my **dissertation**. My supervisor has completed Section 2 of this Ethical Approval Form and has signed where appropriate. | | | | |
| **Name:** |  | **Signature:** |  | **Date:** |  |