SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

TAREK HELAL

Web Application for Finnish and English Language Learning

Bachelor Thesis
Spring 2024
Bachelor of Engineering - Automation Engineering

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree Program: Bachelor of Engineering, Automation Engineering

Specialization: Machine Automation

Author: Tarek Abdelghafar Hamed Mohamed Helal

Title of thesis: Web Application for Finnish and English Language Learning

Supervisor: Raine Kauppinen

Year: 2024 Number of pages:49 Number of appendices:0

This thesis was completed for PNP Crossing Borders, a dynamic private company located in Seinäjoki, South Ostrobothnia. Established in 2002, the company is a trusted trainer and partner to a wide array of businesses and organizations, both small and large. The focus of the thesis was to transform an existing mobile application, originally developed during practical training at the company to facilitate English and Finnish language learning, into a web-based application. The aim was to develop a web application ready for deployment and to select an appropriate development stack while enhancing the user experience.

Several key improvements were introduced in this web-based version that were not present in the original mobile app. One major enhancement was the inclusion of example sentences, which aids users in learning phrases and sentence structures more effectively. This feature shows example sentences in Finnish with English translations, and in English alone when learning English, and was added to both the word puzzle and dictation pages. Another significant addition was the `StrikesDisplay` component, which gamifies the learning process by providing visual feedback on user performance, making the overall experience more engaging. User ranking and performance analytics were also added to the profile page, offering users a clear view of their progress and ensuring accurate ranking even when users have identical scores.

The development process was conducted with attention to detail. A multi-tier client-server architecture was chosen after exploring various options. The MERN stack was utilized: MongoDB served as the database, Express.js and Node.js managed server-side operations, and React was used for the user interface. This stack provided a robust foundation, ensuring ease of use, fast performance, and effective user feedback with consistent use of JavaScript across all technologies. Several challenges were addressed, such as integrating the frontend and backend, improving the user experience, and implementing secure authentication methods using JSON Web Tokens (JWT).

The outcome is a unique language learning tool deployed on the Render cloud platform. It aligns with the company’s mission to educate and is ready for use. Future expansions and improvements are possible, such as adding more languages and features to enhance the learning experience. User feedback will guide these enhancements, continuously adapting the application to the needs of learners.

[1] Keywords: MERN Stack, JWT, MongoDB, Express.js, React, Node.js

TABLE OF CONTENTS

Table of Figures

Terms and Abbreviations

API	Application Programming Interface, which allows programs to communicate with each other.
BSON	Binary JSON. A binary representation of JSON-like documents.
CORS	A protocol allowing web apps to access servers on different domains.
CRUD	Persistent storage functions: create, read, update, and delete data.
DevOps	Practices to shorten development lifecycle, ensure continuous delivery, and quality.
DOM	Interface for web documents to change structure, style, content.
HTTP	Protocol for transmitting hypermedia documents like HTML over the internet.
JWT	Compact, URL-safe way to transfer claims between parties.
JSON	Text format for structured data, used in web applications.
jQuery	A fast, small, and feature-rich JavaScript library.
MERN Stack	Technologies (MongoDB, Express, React, Node.js) for full-stack web development.
MongoDB Atlas	Cloud database service with automated management features.
NoSQL database	Stores and manages data in non-relational formats.
SDLC	A process for planning, creating, testing, and deploying software.
SOA	Architectural pattern where services are accessed within a network.
UI	Part of software interacting with users via screens and elements.
Virtual DOM	Copy of DOM to compare changes with original DOM.

1 Introduction

1.1 Background

Learning languages is essential today for traveling, education, making friends, and exploring different cultures. This thesis aims to create a web application for learning English and Finnish, building upon a previous mobile application developed for the same purpose.

The previous mobile application was developed for a company called PNP Crossing Borders to facilitate and enhance the learning journey for English and Finnish. Inside the application, there were three levels of difficulty: easy, medium, and difficult. In addition, there were also three patterns of learning: puzzle word mode, dictation mode, and definition mode.

Learning Finnish is challenging for many foreigners, while English is essential for global communication. Thus, a need exists for a dynamic and efficient way to learn both languages. A mobile app was created to meet this need. The mobile app was developed only for Android and iOS platforms, which limited the user base. Web-based apps offer better integration and scalability. They can be integrated with different systems and easily scaled up for growth. Additionally, the screen size of mobile devices struggled to display long words in a readable font, a problem that is solved by using a browser on larger screens.

Conducted for PNP Crossing Borders, a Finnish network of trainers, translators, and experts, this thesis sought to provide a tool to help achieve their educational goals locally and globally. A web-based application meets this need by offering enhanced accessibility and functionality, overcoming the limitations of its mobile predecessor.

1.2 Aim of the thesis

The main goal of the thesis is to convert an existing mobile application into a web platform. This aligns with the company’s mission to improve the accessibility and user-friendliness of their language learning platform, thereby enhancing the learning experience for learners of Finnish and English.

The thesis aims to achieve the following objectives:

1. Develop a web-based application that is ready for deployment, ensuring it is robust and reliable with both frontend and backend components to allow deployment without requiring further adjustments.

2. Determine a development stack for the web application by utilizing technologies for frontend, backend, and database, focusing on scalability, performance, community support, and future-proofing to meet current requirements while allowing room for growth and updates.

3. Deliver a good user experience through user interface design that facilitates easy navigation, swift performance, and relevant feedback for users. This is intended to enhance user satisfaction and efficiency in usage.

By accomplishing these objectives, the thesis aims to introduce a web-based language learning tool that effectively supports English and Finnish learners, thus achieving the primary aim of improving accessibility and user-friendliness of the language learning platform.

1.3 Methods

A constructive methodology was employed in this thesis. A new web application was developed using information from existing sources, such as literature on web application architecture and development.

Problems are solved by creating new outputs from old information (Fellows & Liu, 2015, pp. 73-74). Empirical and theoretical knowledge are combined. The product provides new information and experience. The new construction is tested in the target organization to assess its functionality and adoption.

Accessible literature is reviewed to understand the chosen issue (Kothari, 2004, p. 12). Theoretical and practical literature from similar studies is assessed. This review helps define the research problem in context by identifying available data and resources.

The constructive research methodology involved problem identification, acquisition of practical and theoretical knowledge, and solution generation. The goal was to develop a deployable web

application. Testing, theoretical connections, examination of application scope, and novelty assessment were conducted comprehensively.

1.4 Structure of the thesis

The thesis is divided into several parts: the theoretical section, the development process, the results, and finally, the discussion and conclusion. The theoretical section begins with Chapter 2, which focuses on software development models and explores four different models. Chapter 3 examines various software architectures, discussing their importance. Chapter 4 delves into JavaScript frameworks. Chapter 5 discusses the technology stack employed, specifically MERN (MongoDB, Express.js, React, and Node.js), providing an analysis of each component and highlighting their benefits.

Chapter 6 details the development process, starting with the selection of the development model and the reasons for choosing the MERN stack. It covers backend development, including server initialization, controllers, models, and routes. The chapter then moves to frontend development, discussing React setup, components, styling, and data management, and concludes with the deployment of the platform. Finally, Chapter 8 offers discussions and the conclusion of the study.

1.5 Company presentation

The thesis was conducted for PNP Crossing Borders, a Finnish network of trainers, translators, and experts. PNP Crossing Borders assists Finnish businesses and institutions in maximizing efficiency when interacting with international clients, vendors, peers, and collaborators. Based in South Ostrobothnia, the company has operated since 2002 and has established its position as a trainer and partner for small and large companies and organizations.

Since 2002, PNP Crossing Borders has become a premier language trainer and provider of communication and internationalization support services in Western Finland. The company has provided training and coaching to staff members of around 200 companies and specialist organizations. Professionals such as technicians, engineers, designers, sales and marketing personnel, finance experts, management, and customer service staff have been trained to enhance performance and efficiency when dealing with international clients and partners.

In this thesis work, ChatGPT was utilized for grammar checking and adjusting sentence structure to ensure clear and fluent expression. However, all the text in this thesis was written by the author and not generated by AI. All AI-suggested structures were carefully examined against the original sources, and all sources have been properly referenced.

2 Software development models

2.1 The Waterfall model

The Waterfall model is considered a simple and continuous approach for software development (Hallal, 2021, p. 20). It is composed of several cycles; each cycle must be completed before progressing to the next one. The product of each phase acts as the input to the next phase. The phases are presented in Figure 1.

Figure 1. Waterfall model (Hallal, 2021).

The Waterfall model has several benefits (Hallal, 2021, p. 20). The stages are clear and easy to follow. Each stage is thoroughly documented. This model works well for smaller projects with clearly defined requirements. However, there are drawbacks. The product is delivered late in the cycle, which can be problematic. The model is unsuitable for complex or evolving projects because early feedback is not available. It is also ineffective when requirements are likely to change.

2.2 Agile model

The Agile methodology, including frameworks like Scrum, is widely used in IT and non-technical projects (Hallal, 2021, pp. 20-21). This approach breaks down the product into manageable cycles or iterations, usually lasting 2-4 weeks. During each iteration, the development team delivers a functional software version. Use cases are segmented into iterations to achieve a functional segment at the end of each cycle. This iterative process allows for continuous, incremental releases, all rigorously tested. Early issue detection and resolution is facilitated, and stakeholders are actively involved throughout to gather feedback. An overview of the Agile stages is shown in Figure 2.

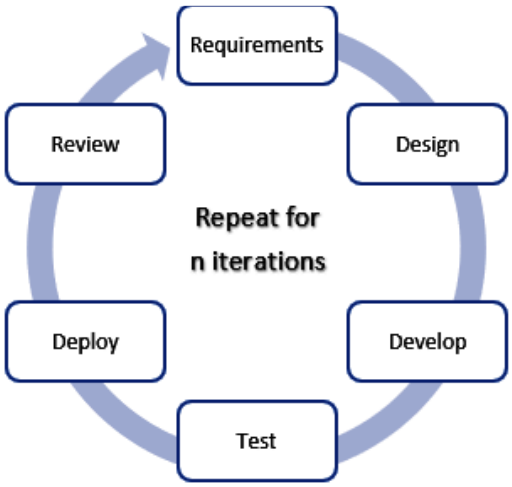


Figure 2. Agile phases (Hallal, 2021).

Agile practices support both the people involved and the final product (Lui & Chan, 2008, pp. 78-79). Key practices include Real Customer Involvement, an Informative Workspace, Shared Code, Short Iterative Cycles, User Stories, Standup Morning Meetings, Refactoring, Pair Programming, Incremental Design, Continuous Integration, and Test-First Programming.

Scrum is a framework used in the Agile process, providing guidelines, milestones, and checkpoints without a fixed process (Heath, 2021, pp. 5–6). It focuses on self-organizing, multi-functional teams to handle complex problems. Software parts are created in brief cycles called Sprints, starting with planning, followed by implementation supported by daily Scrum meetings, and ending with evaluation and planning for the next Sprint (op. cit., p. 42).

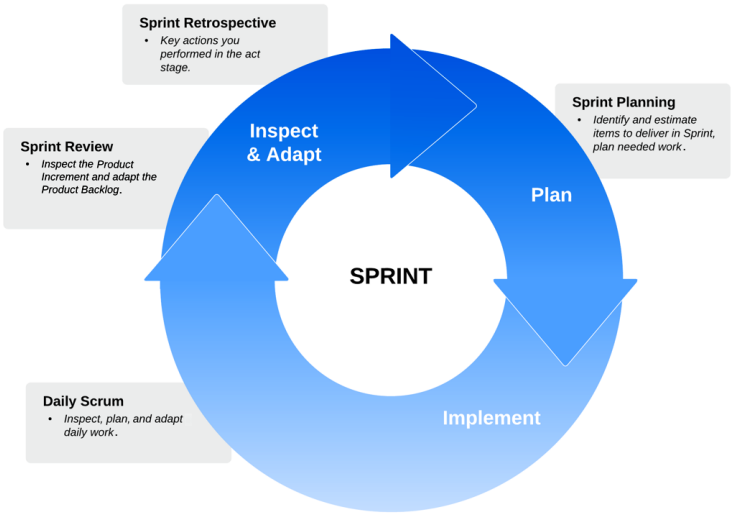


Figure 3. The Sprint (Heath, 2021).

2.3 Spiral model

The Spiral model combines elements of both the Iterative and Waterfall models (Hallal, 2021, p. 21). This model is used for large projects and managed risks early in each iteration. Four main phases are involved: setting objectives, conducting risk analysis, reviewing, and finally developing and testing.

Initially presented for releasing various prototypes throughout different project stages until the final version is achieved, the Spiral model offers continuous generation and improvement of new ideas (Lui & Chan, 2008, pp. 15-16). Software developers might implement the Spiral model based on project specifications, offering flexibility by combining it with other approaches like the Waterfall model to take advantage of both.

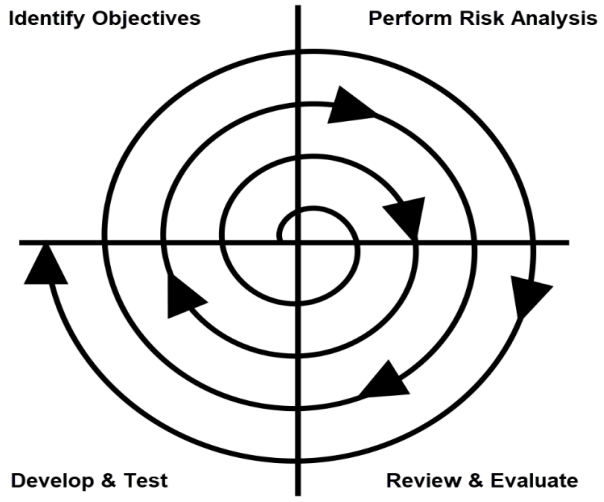


Figure 4. Spiral model (Hallal, 2021).

The advantages of the Spiral model include early identification of risks, prioritization of high-risk parts, high stakeholder involvement, early system visibility through prototypes, and constant feedback from stakeholders (Hallal, 2021, pp. 22-23). However, disadvantages include the requirement for a large budget, making it unsuitable for small projects with minimal risks. controlling the process is complex, and expertise in risk assessment is required to implement the model effectively.

2.4 DevOps model

In a DevOps framework, collaboration between developers and operations teams is emphasized (Hallal, 2021, p. 23). Traditionally, companies allocated resources to teams with specific

responsibilities. A development team focused on designing and constructing the product. An operations team was responsible for setting up the environment and hosting the product. A testing team was tasked with creating test cases, conducting comprehensive Quality Assurance (QA) testing, and providing feedback to the development team.

However, in the DevOps methodology, developers and operations teams are encouraged to collaborate closely as a unified team throughout all stages of the software development life cycle (SDLC) (Hallal, 2021, p. 23). A successful implementation of DevOps ensures uninterrupted feedback, fast deployment, enhances the development process, and automates manual tasks.

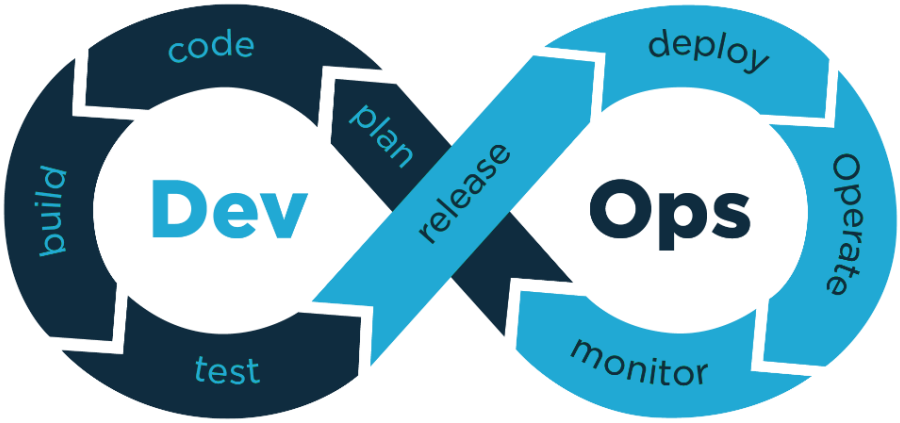


Figure 5. DevOps model (Hallal, 2021).

Advantages of the DevOps model include accelerated delivery of functionalities, enhanced reaction to issues, streamlined operations, decreased bottlenecks, improved communication and collaboration, and increased productivity among team members, freeing up time for innovation (Hallal, 2021, p. 24). However, the disadvantages are that it requires a cultural shift and adoption of new communication methods, which can be challenging in classic environments. Base framework upgrades may be necessary to optimize processes, which can be costly for some organizations, and rapid development may result in significant security vulnerabilities.

3 Software architecture overview

3.1 Architecture requirements

Software architecture represents the various components or subsystems of a software system and their interrelationships (Pillai, 2017, p. 2). The design of a software system provides its structure, characterized by the segmentation of the system into parts, their organization, and their interactions (Martin, 2018, pp. 117-118). This structure aims to simplify development, deployment, operation, and continuous maintenance, ultimately supporting the system's life cycle. Effective architecture enhances understanding of the developed system, reduces overall system costs and increases programmer efficiency.

When dividing an application into parts, the architect assigns specific duties to each component (Gorton, 2011, p. 3). These duties define the tasks each component performs, leading to a collective set of components that collaborate to deliver functionality. Responsibility-driven design, which originated from object-oriented programming, effectively outlines principal components by focusing on behavioral modeling through objects, responsibilities, and collaborations. The most critical aspect of an architect's job is the integrity of architectural design (op. cit., p. 101). Good design is essential, as poor design weaken other efforts. Experienced architects are crucial for effective design.

The starting point for the design phase is the architecture requirements (Gorton, 2011, p. 101). This phase consists of two steps, which are done repeatedly. First, a general plan for the architecture is chosen, based on established architecture patterns. Then, the individual parts of the application are defined, showing how they fit into the plan and what each part is responsible for. The result is a collection of architecture views that illustrate the design, along with a document explaining the design, the main reasons behind key design choices, and the potential risks associated with moving forward with the design.

3.2 Multi-tier architecture

When a user interacts with a web server to browse web pages, two main parts are highlighted: the user's browser (client) and the distant web server (server) (Pillai, 2017, p. 6). These are

the central elements of the system. Other parts, like caching proxies or a remote cache on the server, may exist but are not the primary focus of the architecture explanation.

For small projects, one architecture pattern like N-tier Client-Server might be enough (Gorton, 2011, p. 102). For larger projects, multiple known patterns might be used. The architect decides how these patterns fit together to create the whole plan.

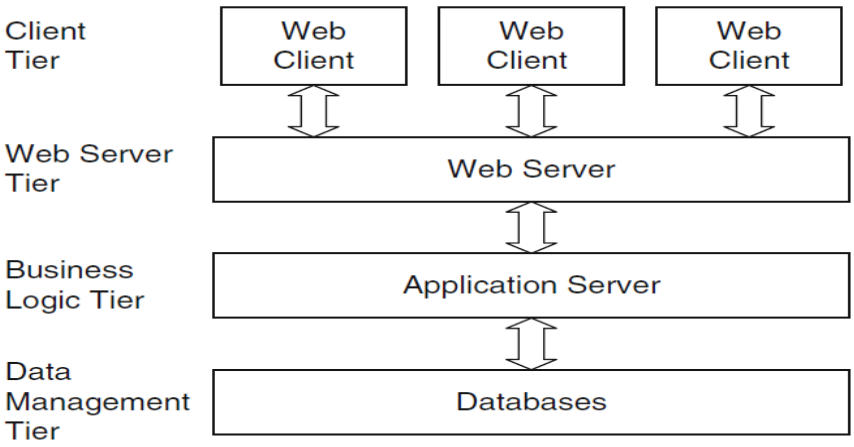


Figure 6. Multi-tier Architecture (Gorton, 2011).

Figure 6 illustrates the layout of the multi-tier architecture pattern for a web application (Gorton, 2011, p. 102). Different tasks, such as displaying content, handling business logic, and managing data, are separated. Communication is synchronous, with messages going back and forth between the layers one at a time. Each layer waits for a response before proceeding. The setup is flexible, allowing configuration of a multi-layered application as needed, whether on one computer or with each part on its own.

Some quality aspects of the N-Tier Client-Server pattern include availability, failure handling, ease of change, performance, and scalability (Gorton, 2011, p. 103). If one server fails, others can continue working, though performance may be affected until the issue is resolved. If a server breaks during client communication, the request may be redirected to another working server seamlessly. The separation of concerns allows modifications to one part without affecting others. Good performance is offered, but the number of simultaneous processes, communication speed between layers, and data transfer amount must be considered. Scalability is supported, as more servers can be added easily to accommodate more users, though the data layer might become a bottleneck.

3.3 Microservice architecture

The microservices architecture allows the solution to be split into different parts (Hallal, 2021, p. 109). Each part operates independently and offers a specific service. The microservices architecture comprises a set of independent services. Each service is self-contained and delivers a specific business function.

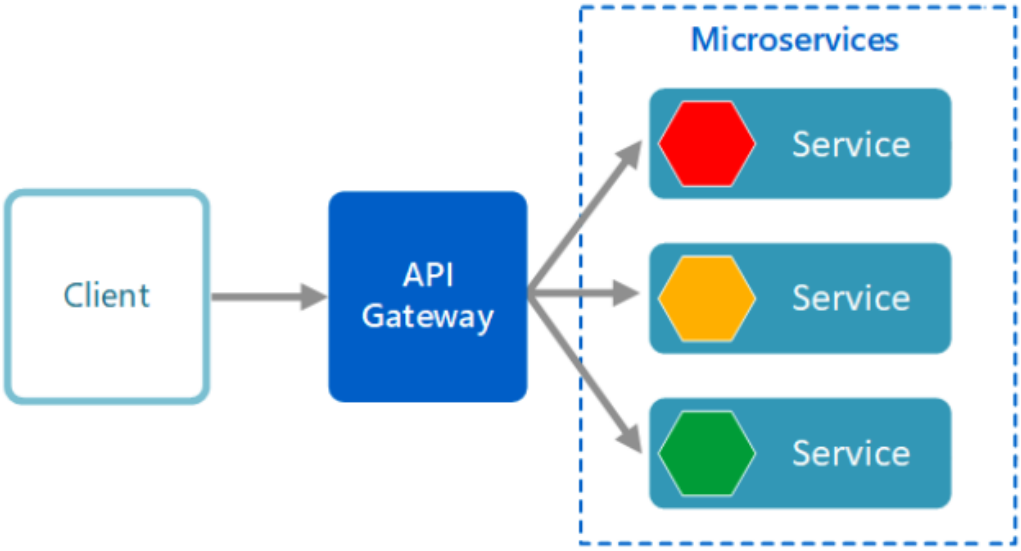


Figure 7. The Microservices architecture design (Hallal, 2021).

This architectural pattern has several features (Hallal, 2021, p. 109). Microservices are small, self-sufficient, and loosely connected. Each service maintains its own codebase and can be managed by small development teams. Each service operates independently and can be deployed on its own. Updating one service doesn't require redeploying the entire system. Services manage their own data access using private databases. The inner workings of each service are isolated and inaccessible to other services. Communication between services is facilitated via well-defined Application Programming Interfaces (APIs). Direct access to the services from the client app is not allowed. Instead, services are accessed through an API gateway, which directs requests to the appropriate services.

3.4 Service-oriented architecture

Service-Oriented Architecture (SOA) enables the use of services accessible within a network (Hallal, 2021, p. 111). Its format is like n-tier architecture, but the presentation layer cannot directly access the business layer; it must go through the services. In SOA, the Service Layer acts as an intermediary between the Presentation Layer and the Business Layer. This setup

allows modification of the Business Layer without impacting the Presentation Layer. The diagram below illustrates SOA within a layered framework:

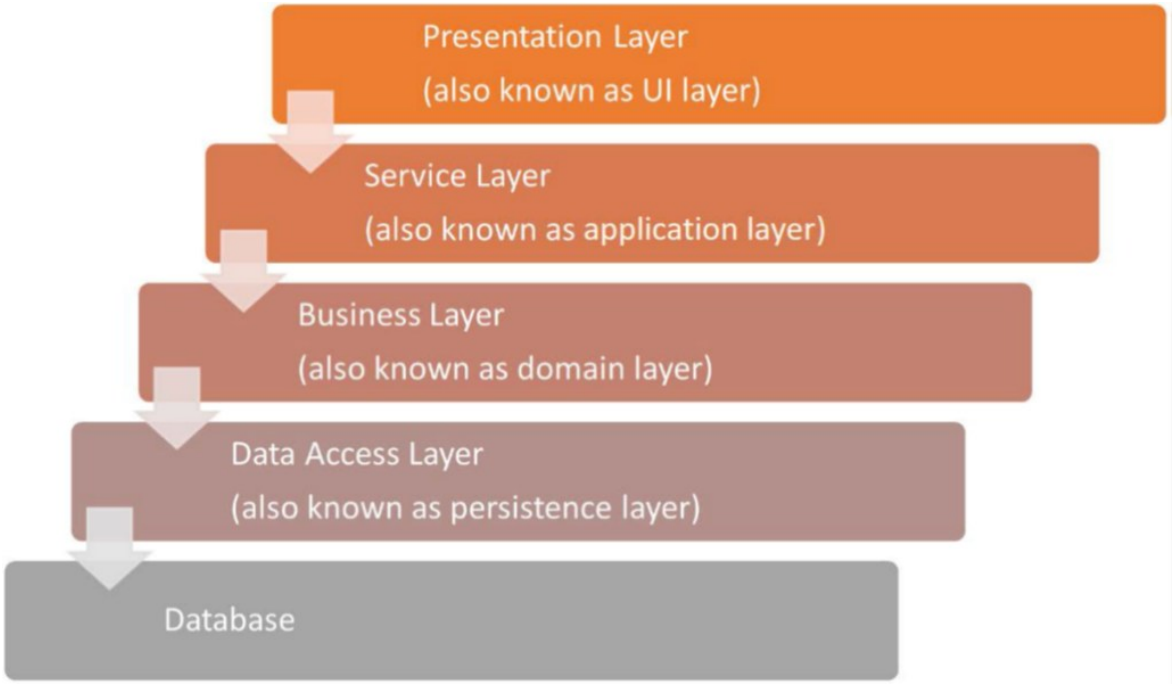


Figure 8. Service-Oriented Architecture (Hallal, 2021).

The main difference between microservices architecture and SOA lies in their scope (Hallal, 2021, p. 112). Microservices architecture emphasizes autonomous, cloud-based services focusing on application-level functionalities. SOA has a broader enterprise scope, where services may not have their own databases and can manage multiple business capabilities. Microservices handle only one business capability per service.

3.5 Serverless architecture

The serverless pattern, or serverless architecture, supports the use of cloud infrastructure and cloud-driven code (Hallal, 2021, p. 113). This framework enables the deployment of solutions on external infrastructure, freeing developers from managing server software and hardware. Applications are broken down into small, self-triggered functions that can be activated and scaled independently, as illustrated in Figure 9.

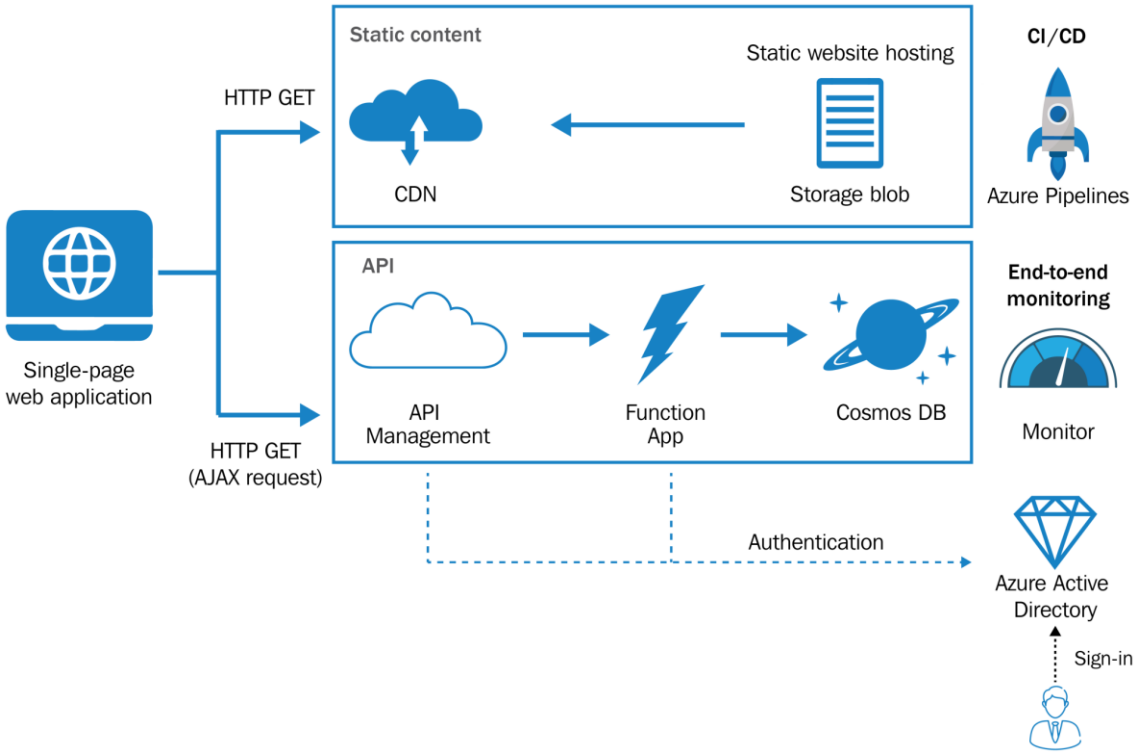


Figure 9. Serverless architecture (Hallal, 2021).

Azure's serverless infrastructure can be utilized for this approach (Hallal, 2021, pp. 113-114). Several components are available within Azure functions: A Content Delivery Network (CDN) enhances response times by caching content. Azure Blob Storage allows the storage of large, unstructured data on Microsoft's platform. Function App operates on an event-driven model, creating self-sufficient functions activated by client HTTP requests. API Management, positioned in front of the functions, serves as an API gateway, separating the frontend app from the backend functions and offering Uniform Resource Locator (URL) rewriting and request management. Azure Cosmos DB provides a NoSQL database solution. Azure AD is used for user authentication. Azure Monitor gathers performance measures and resource usage. Azure Pipelines facilitates Continuous Integration (CI) and Continuous Delivery (CD), automating code building, testing, and deployment.

4 JavaScript frameworks

4.1 Introduction

A framework is defined as a set of tools and procedures structured to address common challenges in project development (Argüelles & Murrugarra, 2018, p. 21-22). These frameworks offer solutions that are comprehensive and tested in various environments, enhancing the reusability of different functionalities to cut down on both time and costs. A JavaScript framework is a collection of components and libraries designed to meet the requirements of web applications. These requirements include routing, data handling via XMLHttpRequest (XML (Extensible Markup Language) HTTP (HyperText Transfer Protocol) Request), managing the Document Object Model (DOM), organizing code into separate functionalities, establishing standard data flows, and defining lifecycles for specific functionalities.

JavaScript frameworks offer benefits like organizing and structuring code in a maintainable manner, enabling modularity, providing proven methods to common problems, and establishing a structured foundation for developers (Argüelles & Murrugarra, 2018, p. 22). They excel in scenarios where significant business logic occurs on the client side. Tasks such as directing users to different pages, setting up the layout of a webpage, checking data, creating tables, and dividing content into separate pages can be handled efficiently without the latency and overhead of additional HTTP calls to the server.

4.2 Comparison of JavaScript Frameworks

JavaScript developers have multiple solutions to problems. Before 2010, options for implementing functionality were limited (Argüelles & Murrugarra, 2018, p. 22). jQuery was the most popular choice and still is today. However, as projects grew more complex, maintaining jQuery became challenging, often resulting in code with mixed functionalities, commonly called "Spaghetti code".

In 2010, Google introduced Angular, a widely adopted JavaScript framework (Argüelles & Murrugarra, 2018, pp. 22-23). Angular provided a comprehensive toolkit and introduced modules, components, routes, and templates, offering a more organized approach to JavaScript code. Following Angular's success, many other JavaScript frameworks emerged. Some gained

popularity through sponsorship by large companies, such as Facebook's React.js. Others, like Meteor and Vue, gained traction through community adoption. Innovation in the technology landscape continues, leading to the emergence of new frameworks. One example is Aurelia, developed by a key engineer from the Angular 2 project. Despite being relatively new, Aurelia has quickly gained popularity as a promising framework.

Angular, uses TypeScript and follows the MVC (Model-View-Controller) framework, making it well-suited for complex projects despite its steep learning curve (Hallal, 2021, p. 185). This complexity has led major companies like Google, Guardian, PayPal, and Nike to adopt Angular. In contrast, React.js, released by Facebook in 2013, is a JavaScript library with a moderate learning curve, known for its strengths in JavaScript usage and server-side rendering. This has contributed to its widespread adoption by large-scale web applications at companies such as Facebook, Twitter, Instagram, Uber, Netflix, and Airbnb. Meanwhile, Vue, created by Evan You in 2014, is known for its lightweight design and ease of learning, following the MVVM (Model-View-ViewModel) framework. Vue is favored by companies like Alibaba, GitLab, and Nintendo. Each framework offers unique benefits and serves different needs in web development.

While Angular and React excel in constructing large-scale enterprise web solutions featuring advanced components and highly dynamic content, coding in React tends to be simpler and swifter compared to Angular (Hallal, 2021, p. 186). On the other hand, Vue stands out for its lightweight nature and ease of learning, while also exhibiting superior performance among the three frameworks. Additionally, Vue's development community is steadily growing, surpassing the growth rates of both React and Angular communities. As per a 2023 Stack Overflow Developer Survey, React ranks as the second most popular framework, following Node.js (Stack overflow, 2023).

React.js boasts an important feature known as server-side rendering (Argüelles & Murrugarra, 2018, p. 25). Unlike conventional JavaScript frameworks where rendering occurs on the client side, React.js enables pages to be processed on the server rather than the client. This reduces the browser's workload in interpreting JavaScript and transforming it into HTML, improving loading times for data-intensive pages. React also supports alternative rendering libraries, such as Inferno.js, which offer enhanced rendering performance.

5 MERN stack

5.1 Technology stack

A technology stack is a set of technologies chosen and used to build web applications, mobile applications, or similar applications (MongoDB, n.d.). A seamless user experience, scalability, and cost-effectiveness are qualities desired in a good technology stack. Typically, a full technology stack consists of a frontend, backend, and database.

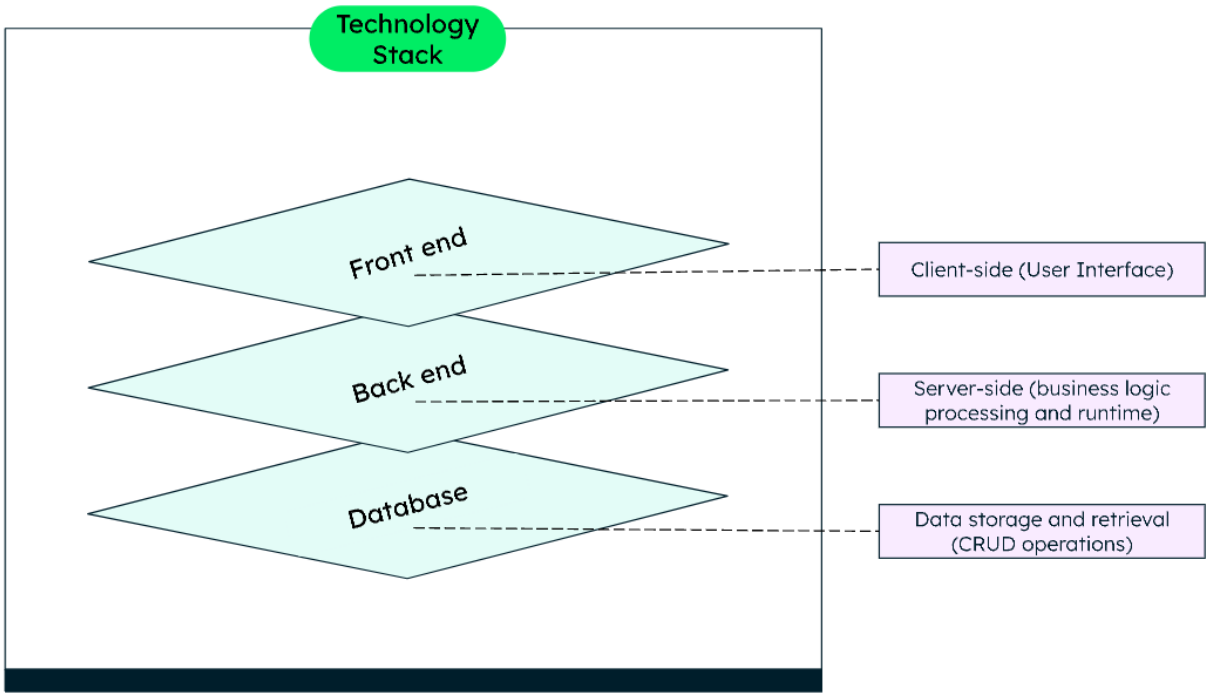


Figure 10. Technology stack (MongoDB).

At the beginning of this introduction to the MERN stack, it's important to emphasize that the MERN stack —consisting of MongoDB, Express, React, and Node.js— is not only a set of technologies but it represents a strong integration that provides a practical and efficient approach to implementing client-server architecture. The MERN stack combines MongoDB, Express, React, and Node to create web applications (Hoque, 2018, pp. 6-7). In this stack, Node and Express handle the web backend, MongoDB acts as the NoSQL database, and React builds the frontend that users communicate with. These technologies are free, open-source, platform-independent, and JavaScript-powered, with robust community and industry support. When integrated, they form a powerful and straightforward full JavaScript stack for web development.

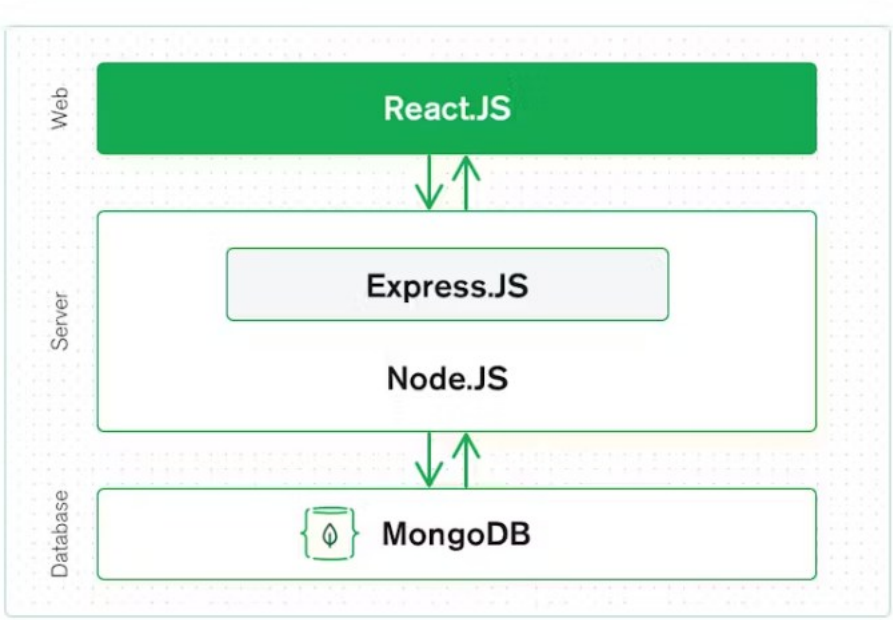


Figure 11. MERN stack (MongoDB).

5.2 MongoDB

5.2.1 Introduction to MongoDB

MongoDB, a well-known NoSQL database, is recognized for its effective implementation and active community, which are essential for support and problem-solving (Mehrabani, 2014, p. 3). It plays a critical role in web applications by facilitating the transfer, processing, and storage of JSON documents from the React.js front end to the Express.js server and MongoDB (MongoDB, n.d.). For cloud applications, MongoDB Atlas streamlines deployment with easy integration using the MongoDB Node.js driver.

CRUD operations (Create, Read, Update, Delete) are performed through HTTP requests using POST, GET, PUT, and DELETE methods (MongoDB, n.d.). MongoDB stores data in BSON (Binary JSON) format, while Express.js handles these requests. React stores data as JavaScript objects on the front end, and JavaScript is used on the backend with Node.js allowing scripting of backend logic. Express converts data between JavaScript and JSON using the`.json ()` method, supporting efficient data flow in web development.

5.2.2 Key Features

MongoDB offers key features such as non-relational data storage, allowing flexible data models and schema modification post-insertion (Mehrabani, 2014, p. 3). High availability is ensured through replica sets, containing multiple data copies. Sharding distributes traffic and enables horizontal scaling by splitting data based on sharding keys. GridFS allows MongoDB to function as a filesystem for data storage and retrieval.

MongoDB's high performance in data persistence is supported by embedded data models, reducing I/O activity, and indexes for faster queries, including keys from embedded documents and arrays (MongoDB Documentation, n.d.). The MongoDB Query API supports CRUD operations, data aggregation, text search, and geospatial queries.

5.2.3 Benefits for Web Development

Database connection in applications is simplified by using drivers provided by the database designer, reducing the amount of code needed for connecting, analyzing, reading, and writing to the database (Phaltankar et al., 2020, p. 370). MongoDB offers drivers for various programming languages, with the Node.js driver being notably popular.

The successful integration of MongoDB with Node.js, a key language for web applications, has driven its growth and adoption (Phaltankar et al., 2020, p. 370). This integration benefits both technologies, resulting in successful implementations across a range of applications, from handheld devices to large web platforms. The ease of integrating MongoDB with Node.js makes it a preferred choice during MongoDB driver presentations.

5.3 Express

5.3.1 Introduction to Express

Express is a key framework for building web applications and APIs using a Node server (Hoque, 2018, pp. 6-7). It enhances Node by providing essential website functionalities. In a Node-based website, Express acts as a routing and middleware web framework, where an application is a sequence of middleware function calls. These middleware functions have

access to the HTTP request and response objects, as well as the next middleware function in the request-response cycle. This allows for flexible middleware insertion, providing high adaptability for development.

Express is unique among web server frameworks for its minimalistic default configuration (McClay, 2017, p. 123). It is a concise yet powerful toolkit for web server functionalities, ensuring it includes only the essentials needed by developers, avoiding unnecessary overhead from additional functions or complexities.

5.3.2 Features and Capabilities

Express JS is known for its clarity and adaptability, offering features that facilitate the development of efficient and scalable web applications (GeeksforGeeks, 2023). It organizes application functionality through middleware and routing while enhancing Node.js HTTP objects to simplify web requests and responses.

Key features of Express JS include routing, middleware, error handling, enhancing request and response objects, and body parsing (GeeksforGeeks, 2023). Routing in Express JS manages HTTP requests by determining the appropriate response to send to the client, and it simplifies this process with built-in methods that allow developers to set up routes directly. Middleware functions act as intermediary processes that are executed between various operations, enhancing security and adding other functionalities. For instance, Express offers pre-built middleware like `express.static()` for serving static files. Error handling is another crucial feature, ensuring smooth program execution even when issues arise. Express simplifies error management through asynchronous handling and global error handling processes. Additionally, the request and response objects in Express JS are enhanced with additional functionalities, such as accessing URL parameters and providing convenient methods for sending responses, like `res.send()`. Lastly, Express JS includes body parsing capabilities, allowing it to parse data sent from the client to the server using built-in modules like `express.json()`.

5.3.3 Server-Side Routing

Express is used for routing and handling incoming requests, sending them to appropriate handlers. Routes are defined using methods such as `app.get()`, `app.post()`, `app.put()`, and `app.delete()`. Middleware functions add flexibility by handling tasks like authentication, logging, and data validation before routing. Express can dynamically extract parameters from URLs and send them to other functions, and it allows data validation before saving it to the database using middleware.

Routing in Express involves how an application responds to requests for resources via HTTP methods (Koroliova, 2018, pp. 15-16). HTTP serves as the foundation for data communication on the web, with each document identified by a URL. Express enables the creation of a server that understands these requests, analyzing them and storing information in two objects: the request object and the response object. The request object contains data about the client's request, including parsed URI (Uniform Resource Identifier) query parameters, which are accessible via `req.query`, and route parameters, accessible via `req.params`. The response object holds the data that will be sent back to the client, with methods available for modifying response headers, sending status codes, and transmitting data.

Route handlers, which are callback functions, accept three inputs: the request object, the response object, and a `next` parameter (Koroliova, 2018, p. 19). This `next` parameter passes control to the next request handler in the sequence, allowing for multiple handler functions to be used within a single route method.

5.4 React

5.4.1 Introduction to React

React, a JavaScript library developed by Facebook, is widely used in major web platforms like Facebook, Instagram, Netflix, and WhatsApp Web (Bugl, 2019, p. 8). It addresses common challenges in creating client-side components for web applications (Richey et al., 2020, p. 2). React focuses on solving design and development problems in complex frontend applications.

React's development fits within the expansion of frontend JavaScript frameworks, offering more than just JavaScript for creating page elements (Richey et al., 2020, p. 2). While methods to

insert DOM elements into web pages existed with early technologies like Prototype and jQuery, React integrates JavaScript and HTML, effectively connecting code with the browser's environment.

5.4.2 Component-Based Architecture

React's component-based architecture allows components to maintain their own state and display, which are combined to create complex user interfaces (Bugl, 2019, p. 10). It uses function components and class components, with function components being simpler and class components previously necessary for handling state and advanced features. React Hooks now enable state management and other features without class components, using familiar JavaScript features (op. cit., pp. 14-15). For example, the Effect Hook in the code below is activated automatically upon the component's mounting and when the `name` prop is altered:

```
function Example ({name}) {
  useEffect (() => {
    // This function is called after the component mounts and whenever the 'name' prop changes.
    fetch(`http://my.api/${name}`)
      . then (/* ... */);
  }, [name]); // The second argument is an array of values that the effect depends on.
}
```

React components define what should be displayed primarily through the `render()` method (Fedosejev, 2015, pp. 64-65). Lifecycle methods handle actions before or after rendering and determine if rendering should occur. The lifecycle stages include mounting, updating, and unmounting. Hooks offer a simpler way to manage component state without relying on traditional lifecycle methods.

5.4.3 Virtual DOM

React uses the virtual DOM, an in-memory representation of the actual DOM elements (Boduch & Derks, 2020, p. 14). When a component re-renders, React compares the new

content with the existing one and updates only the necessary parts of the real DOM. This method ensures efficient UI updates while maintaining clear coding principles

React's ability to swiftly identify differences between the virtual DOM and the real DOM, and subsequently re-render only the updated portions in the real DOM, is efficient (Fedosejev, 2015, p. 19). What's more, for React developers, the beauty lies in not having to think about identifying what needs to be re-rendered. React allows developers to write their code as if they were re-rendering the entire DOM whenever there's a change in the application's state.

5.4.4 State Management

React components use JSX (JavaScript XML) to define UI elements (Boduch & Derks, 2020, p. 42). To be functional, these components need data, represented as the component's state. State is the dynamic aspect of a React component that can change over time. For example, a component's state might start as an empty array and then be populated with data using the `setState()` method, triggering an automatic re-render and invoking the `render()` method.

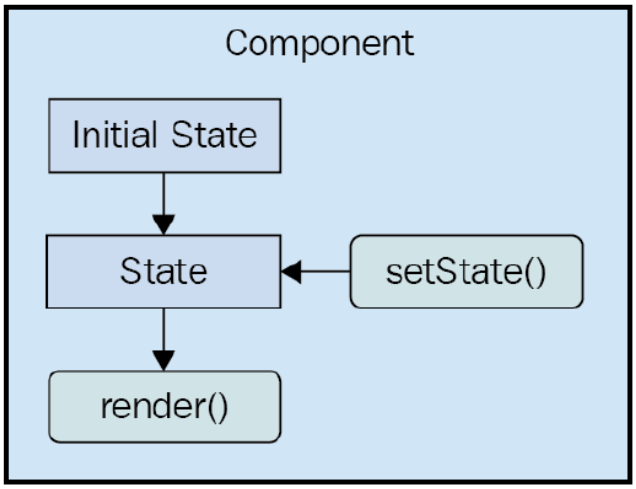


Figure 12. State management (Boduch & Derks, 2020).

Stateful components are those that hold data that changes over time (Chiarelli, 2018, pp. 47-48). Managing component state involves determining what should be considered state and which components should hold it. State should include only the essential data needed to represent information that may change over time. Additional information should be computed within the component's render method. Minimizing state usage is advisable to reduce

complexity, and stateful components should be positioned high in the component hierarchy of the user interface.

5.5 Node

5.5.1 Introduction to Node.js

Node.js simplifies the creation of scalable network programs, addressing the complexities of sophisticated networked applications (Pasquali, 2013, p.7). As web applications grow and handle more data, Node.js helps manage this data efficiently, through a single-threaded model, and event-driven data streams for I/O operations, along with managed concurrency. As network applications expand, the volume of data they handle also increases, leading to challenges in terms of organization and management (op. cit., p.8). This growth in data volume not only overloads the application but further strains the developers. Scaling problems often stem from the inability to accurately predict large system behavior from small system behavior. Node.js addresses these challenges by prioritizing clarity and simplicity, utilizing a single-threaded model with postponed tasks wrapped through callbacks, event-driven data streams for I/O operations, and managed concurrency.

Node, developed as a JavaScript runtime environment, relies on Chrome’s V8 JavaScript engine (Hoque, 2018, p. 7). It reshaped server-side JavaScript usage, extending beyond the browser’s limitations. Node’s event-driven architecture supports asynchronous, non-blocking I/O operations. Its unique approach eliminates waiting times for request handling, making it ideal for building scalable, lightweight real-time web applications that efficiently manage numerous requests. Additionally, Node ships with its default package management system, npm (Node Package Manager). npm provides access to an extensive ecosystem of reusable Node packages created by developers worldwide, currently standing as the largest collection of open-source libraries globally.

5.5.2 Asynchronous Programming

Node.js utilizes JavaScript's event-driven nature to facilitate non-blocking operations within its platform, a characteristic that contributes to its exceptional efficiency (Poojary, 2016, p. 701). JavaScript, being event-driven, allows developers to register code to specific events, which will

then be executed once the event is triggered. This approach enables the seamless execution of asynchronous code without impeding the overall program flow. Originally designed to support browser operations, JavaScript was built around handling browser events. Despite significant evolution over time, the fundamental concept remains the same: Authorizing the browser to transmit HTML user events to JavaScript code. While browsers primarily manage user-generated events like button clicks, Node.js must deal with a diverse array of events originating from various sources.

Given that browsers operate on a single thread, utilizing synchronous programming would freeze all other processes on the page, severely compromising responsiveness and overall user experience (Poojary, 2016, p. 702). Fortunately, this is not the case. Browsers manage JavaScript execution within a single-threaded loop known as the event loop. This loop continuously runs, processing events emitted by the browser and executing corresponding event handlers. After handling each event, the loop moves on to the next event in the queue, ensuring smooth program execution.

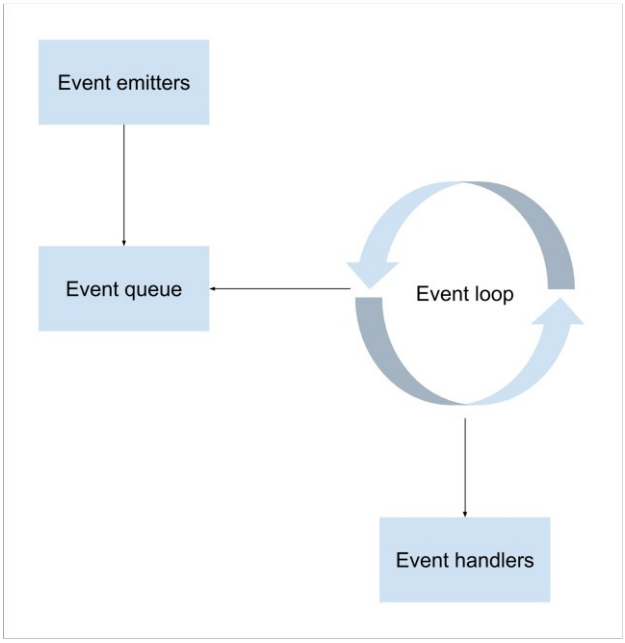


Figure 13. Event loop cycle (Poojary, 2016).

Callbacks play a fundamental role in handling asynchronous operations in Node.js (Acharya, 2023). A callback is a function that will be executed later when a certain task is finished. It gets executed once the asynchronous operation completes. This design allows code execution to continue without blocking, making it suitable for scenarios like handling file I/O or database queries. Promises are a powerful tool for managing asynchronous operations. Essentially, a

promise represents an outcome that will occur in the future either successfully or with an error. These constructs are particularly valuable when handling intricate workflows or coordinating dependencies between various asynchronous tasks.

It is important to note the four states of every promise, as they are widely referenced in terminology (Tsonev, 2014, p. 19). A promise can exist in any of the following states:

- Fulfilled: When the action associated with the promise succeeds.

- Rejected: When the action associated with the promise fails.

- Pending: When the promise has neither been fulfilled nor rejected.

- Settled: When the promise has been either fulfilled or rejected.

6 Development process

6.1 Selection of the Development Model for the Thesis

Choosing a software development life cycle (SDLC) model involves recognizing the unique strengths of each model in addressing development challenges (Hallal, 2021, p. 24). No single model fits all projects or clients; the choice depends on factors such as existing infrastructure, team culture, and client preferences. While the Waterfall model is beneficial for some projects, others are better suited to Agile.

For this thesis, the Agile model was selected due to its iterative development process. Agile enables continuous improvement and refinement based on ongoing feedback, which is advantageous for a single-developer project. This approach allowed for the efficient addition of new features and rapid delivery of a functional application. Agile's iterative nature facilitated frequent testing and adjustments, ensuring that the application continuously aligned with the evolving project goals and effectively met the objectives.

6.2 Reasons for Choosing the MERN Stack

This thesis utilizes a client-server architecture for the web application, making the MERN stack an ideal choice. The MERN stack includes MongoDB, Express.js, React.js, and Node.js. MongoDB, when paired with Node.js, efficiently handles JSON data storage and is supported by MongoDB Atlas, which offers auto-scaling clusters. Express.js, a framework for Node.js, handles HTTP requests and URL routing, while React.js creates interactive user interfaces and manages server communication.

The MERN stack ensures smooth JSON data flow and simplifies both development and debugging. By using JavaScript across all technologies and maintaining a uniform data format, the stack streamlines system comprehension and reduces time spent on data conversions. This integration enhances efficiency and coherence throughout the development process. Additionally, the strong integration of these technologies provides a solid foundation for building robust, reliable, and scalable web applications. By using React to build the user interface, the web application delivers a seamless user experience while also accommodating future growth.

6.3 Preparing Development Environment

Visual Studio Code, a comprehensive source code editor, supports MERN stack technologies (Hoque, 2020, p. 102). Git version control is integrated, and GitHub's cloud-based repositories are used to share the latest code and back it up.

A MongoDB database is deployed using MongoDB Atlas, with the backend code organized in a separate folder. Node.js is installed from its official website to provide the server-side JavaScript runtime (Node.js, n.d.). Required packages are installed with `npm init`, followed by `npm install --save express` for Express.

For frontend development, Vite is used, providing a development server with Hot Module Replacement (HMR) and a build command with Rollup (Vite, n.d.). Vite is installed using `npm create vite@latest app-name -- --template react`, followed by `npm install` to set up dependencies from the package.json file.

6.4 Project Implementation

6.4.1 Application Structure and Functionality

Figure 14 illustrates the word learning application structure. The "Word Learning Game" web application, available in both English and Finnish, featuring several key components. It begins with a login page for user authentication and progress saving. The Word Puzzle Page includes an interactive game where users drag and drop letters across two grids. It offers various difficulty levels and options to check answers, view solutions, retry words, or move on to the next word. Users can reset the game using a reset button and view their scores and levels. Pronunciations, examples, and definitions are accessible. English definitions and examples are retrieved from a free dictionary API, while Finnish meanings and examples are stored internally.

The users can read and listen to example sentences of the word they are learning only after they guess or form it correctly. For example, if the selected language is Finnish, the example sentence will be in Finnish with its English translation. If the selected language is English, the example sentence will be in English only.

The Dictation Page functions similarly to the Word Puzzle Page, where users transcribe words presented audibly, with options to reset the game, view scores and levels, and access example sentences in both written and auditory formats only after answering correctly.

The Definitions Page displays a list of six words, and users must select the word that matches the provided definition. English definitions are retrieved from a free dictionary API, while Finnish definitions are stored internally. This page allows users to learn at their own pace without scoring or leveling.

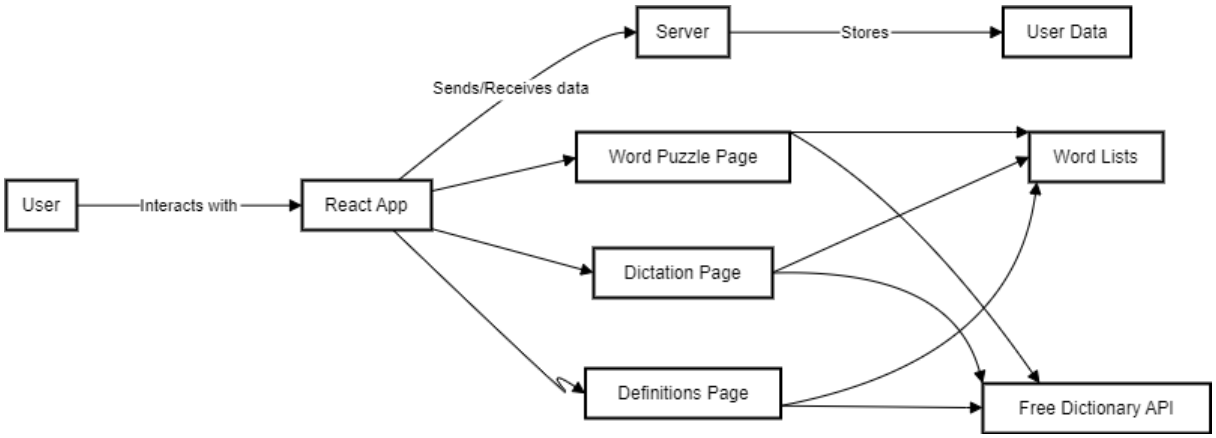


Figure 14. Word Learning Application structure

6.4.2 Architectural Design

The application followed a client-server architecture. Figure 15 illustrates the architectural design diagram. On the client side, React was used, with a component-based architecture to handle user interactions, display data, and manage application state. The client-side architecture was divided into three main layers: the Presentation Layer, the Business Logic Layer, and the Data Access Layer. The Presentation Layer included user interface components like the Word Puzzle Page (interactive word puzzle game), Dictation Page (managed dictation exercises), and Definitions Page (displayed word definitions). Besides these main components, the Presentation Layer also included supportive modules such as ScoreLevelDisplay, SpeakButton, and StrikesDisplay. The Business Logic Layer encapsulated the core functionality and logic of the application. Responsibilities of this layer included handling game logic such as word selection, letter hiding, answer checking, sound generation, and user interactions. The Data Access Layer was responsible for managing data operations and interactions with external data sources. This layer contained word lists and other data sets that were exported and used within the business logic layer to facilitate the game's functionality.

On the server side, MongoDB Atlas was used for user authentication and data storage. The server-side architecture included components for authentication, data storage, and server-side routes and endpoints. Server-side routes and endpoints handled authentication requests and data storage operations. Communication between client and server was established through HTTP requests for user authentication and data retrieval or CRUD operations. The client sent requests to the server, which processed them and returned the necessary data or confirmation.

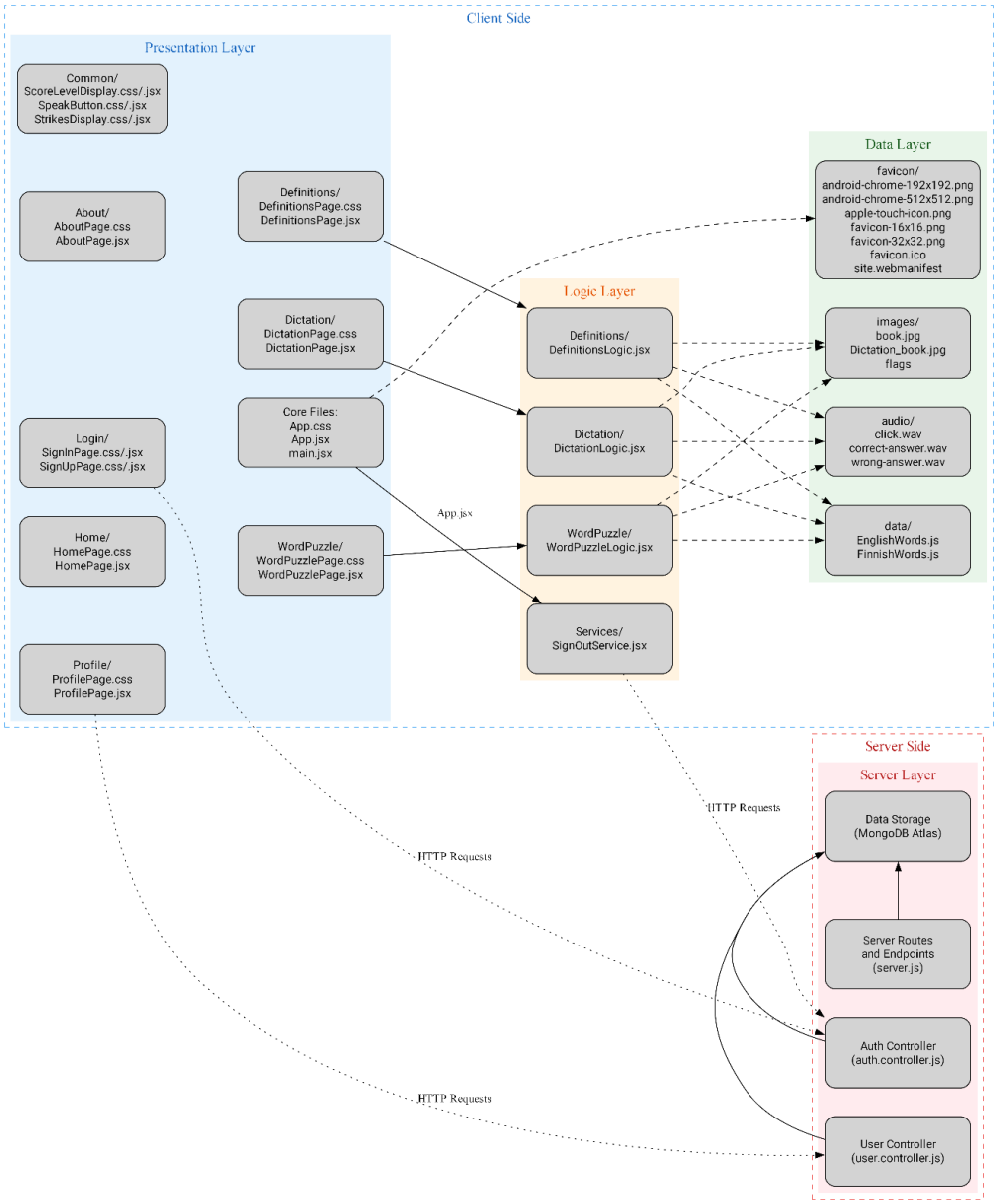


Figure 15. Architectural Design Diagram.

6.5 Backend Development

6.5.1 Server Initialization

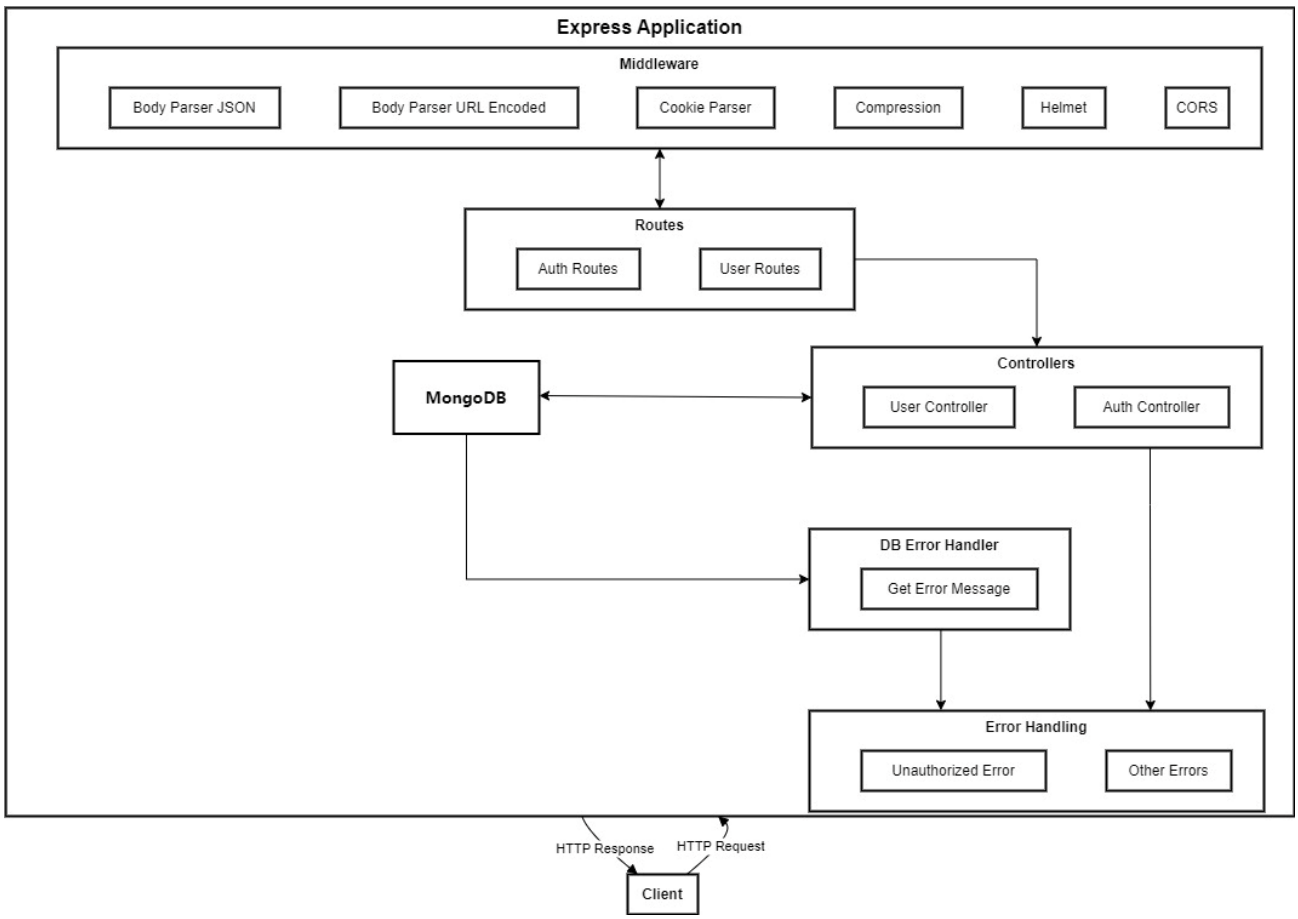
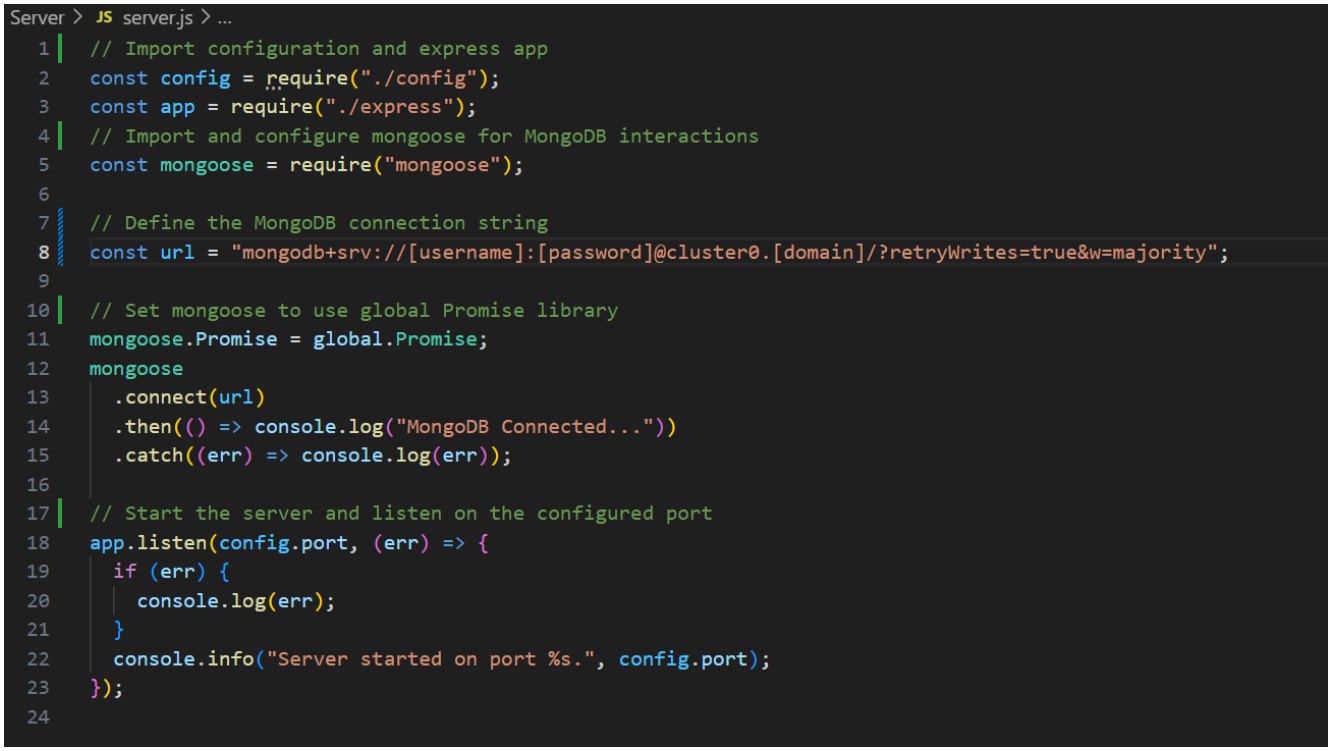


Figure 16. Server Application Architecture and Workflow.

Figure 16 illustrates the server application architecture and workflow, while Figure 17 outlines the server initialization file (server.js) that sets up the backend. The server was started using Express.js, and a connection to the MongoDB database was established through Mongoose. The necessary configurations and the Express app were imported, and Mongoose was configured with the MongoDB connection string, utilizing the global Promise library. The connection to MongoDB was carefully managed, with events handled to ensure stability. Once everything was configured, the server began listening on the designated port to handle incoming requests.



```
Server > JS server.js > ...
// Import configuration and express app
const config = require("./config");
const app = require("./express");
// Import and configure mongoose for MongoDB interactions
const mongoose = require("mongoose");

// Define the MongoDB connection string
const url = "mongodb+srv://[username]:[password]@cluster0.[domain]/?retryWrites=true&w=majority";

// Set mongoose to use global Promise library
mongoose.Promise = global.Promise;
mongoose
  .connect(url)
  .then(() => console.log("MongoDB Connected..."))
  .catch((err) => console.log(err));

// Start the server and listen on the configured port
app.listen(config.port, (err) => {
  if (err) {
    console.log(err);
  }
  console.info("Server started on port %s.", config.port);
});

```

Figure 17. Server.js.

The Express application, shown in Figure 17, was defined in a separate file. Various middleware modules were employed to handle requests and responses. Modules such as Express, body-parser, cookie-parser, compression, Cross-Origin Resource Sharing (CORS), and helmet were imported. These middleware modules were responsible for parsing incoming request bodies, managing cookies, compressing responses, enhancing security, and managing CORS. Additionally, routes for user authentication and other user-related endpoints were established. Error handling middleware was also implemented to catch and manage unauthorized errors and other exceptions, ensuring that clients received appropriate responses.

6.5.2 Controllers

The `user.controller.js` and `auth.controller.js` files managed the application's business logic. The `user.controller.js` handled user data operations such as retrieving profiles, updating information, deleting accounts, registering new users, listing existing users, and retrieving individual user details by ID.

The `auth.controller.js` handled user authentication, defining methods for user sign-in and sign-out. The sign-in method authenticated users, issued JSON Web Tokens (JWT), and sent them

as cookies and JSON. The sign-out method cleared the authentication token stored in the cookie, ensuring subsequent requests were no longer authenticated. Additionally, two methods ensured users were signed in and had the right permissions to access certain routes or resources. The `requireSignin` middleware validated the token, while `hasAuthorization` checked if the logged-in user had the correct privileges. Errors were handled to provide meaningful messages and status codes to clients.

6.5.3 User Model and Routes

In the `user.model.js` file, the user model was defined using Mongoose to interact with MongoDB. A schema was created, specifying required fields such as name and email, along with default values for score and level. Passwords were encrypted using a cryptographic module, with methods provided for password authentication and encryption. Additionally, validations ensured password requirements were met.

In `user.routes.js` and `auth.routes.js`, API endpoints were defined. `user.routes.js` included routes for user operations, such as creating, reading, updating, and deleting users, with middleware to process user IDs in the URL. `auth.routes.js` handled sign-in and sign-out requests, mapping client requests to the appropriate controller functions.

6.6 Frontend Development

6.6.1 React Setup

In the application, React was used to build the user interface, and ReactDOM rendered the App component into the HTML document. The App component was styled with CSS and structured to include a navigation bar and content area. The Router from React Router was utilized to manage navigation within the application.

The navigation bar was designed with interactive links directing users to various pages, such as Home, Sign-In, Sign-Up, Profile, Word Puzzle, Dictation, Definitions, and About. Protected routes required user authentication, which was verified by checking for a JWT token in local storage. The Home page, as shown in Figure 18, was one of these key pages.

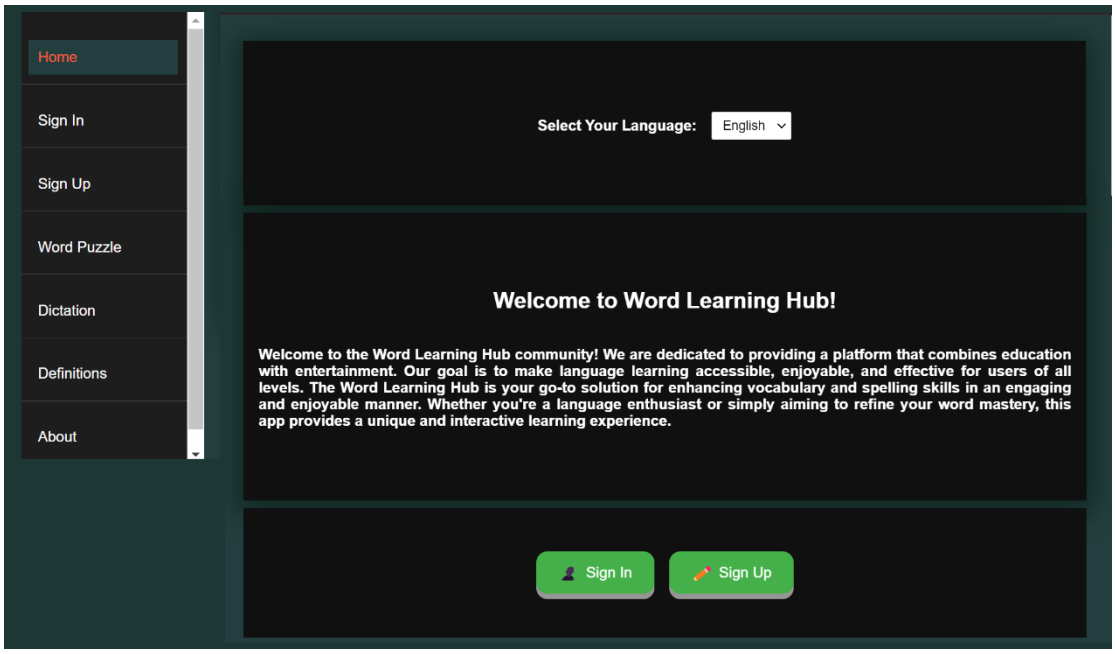


Figure 18. Home page.

The App component managed the state of the menu's visibility and provided functions to handle menu interactions and user sign-out. When the user signed out, the authentication token was removed, effectively logging them out of the application. The routes and their corresponding components were defined within the Router, ensuring that the correct content was displayed when a user navigated to a specific path.

6.6.2 Components

Diagram 19 shows the structure of the "Word Learning Game" web application. The Home Component served as the main landing page, providing information about the app in both English and Finnish. From this page, users could either sign in or sign up, depending on whether they already had an account. The side menu allowed users to navigate to the Sign-In or Sign-Up pages or to access the About Component, which explained the app's features. After signing in, users could also access the Word Puzzle Component (see Figure 20), the Dictation Component, and the Definitions Component (see Figure 21) through the side menu.

The application used Common Components, such as Score-LevelDisplay, SpeakButton, and StrikesDisplay, across different parts of the app. These components supported features like score tracking, strike management, and audio playback.

User authentication was managed by the Login Component, while the Profile Component handled user profiles, allowing users to update their personal information or delete their accounts. The Login Component was directly linked to the Profile Component, providing easy access to user-specific settings.

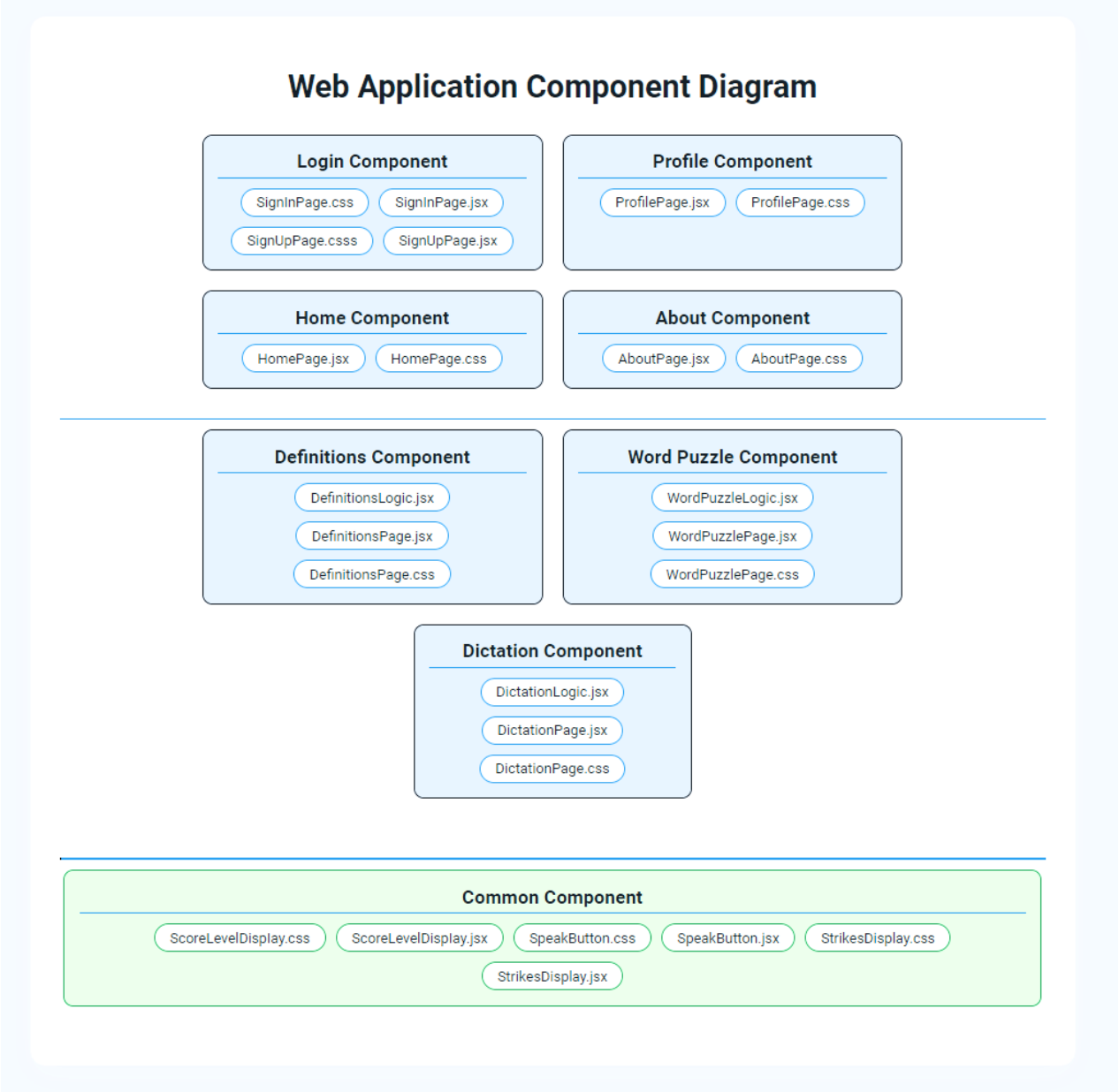


Figure 19. Web Application Component Diagram.

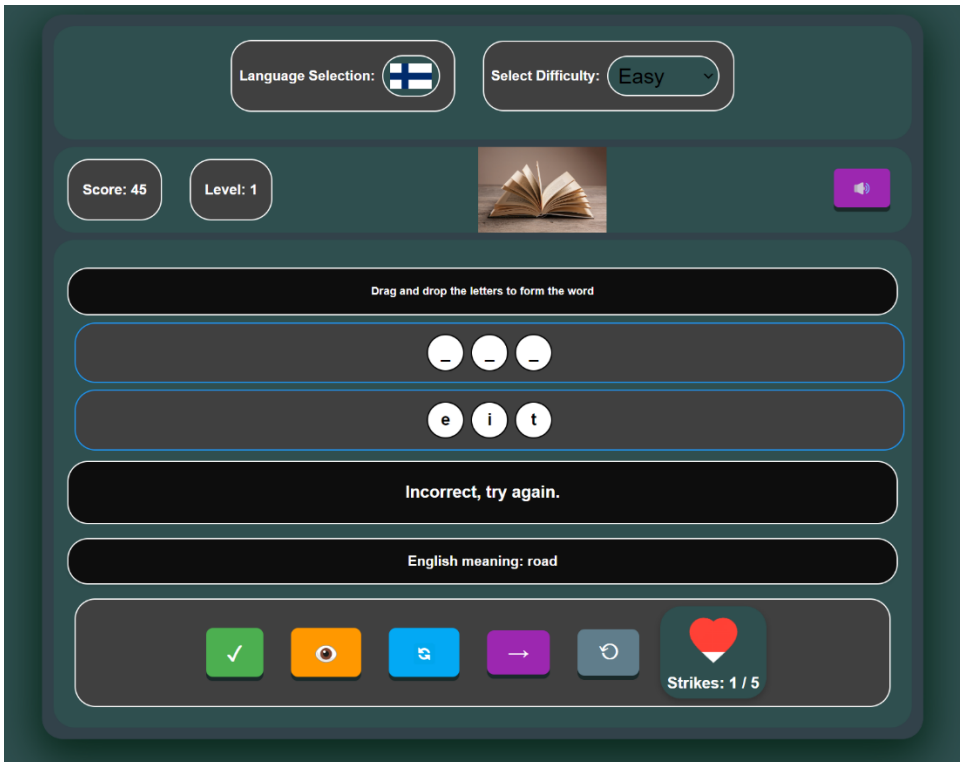


Figure 20. WordPuzzle page.

Figure 21. Definitions page.

6.6.3 Styling and Data

The entire application and navigation bar were styled using the `app.css` file, while individual page components incorporated their own CSS files for better control and display. CSS files employed grid templates and animations to make pages visually appealing. Media queries were included in different CSS files to enhance display on smaller screens. Buttons were defined using CSS, with a hover function to change their color when hovered over. The sidebar menu was designed to disappear when not in use, allowing the app to utilize the full available space. The active navigation page was highlighted with a different color to indicate the currently open page.

The files `EnglishWords.js` and `FinnishWords.js` provided data for the application. In `EnglishWords.js`, words were listed as simple strings within arrays, categorized into 'easy', 'medium', and 'difficult' levels. The `FinnishWords.js` file contained objects with 'word', 'translation', and 'example' properties, such as ` {word: "afta", translation: "a small, painful ulcer in the mouth.”, example: {Finnish: "Minulla on afta suussa.", English: "I have an aphthous ulcer in my mouth."}} `. This structured data was exported from the files and used in the components of the application.

6.7 Deployment

Render offers reliable cloud infrastructure, enabling rapid and scalable product deployment for software teams (Render, n.d.). The platform supports everything from small prototypes to complex applications, with a strong emphasis on reliability and uptime. Render is headquartered in San Francisco, California, and has a globally distributed remote team.

For deployment, React DevTools were disabled, and the API base URL was updated to the production URL. The code was pushed to a new GitHub repository and deployed to Render.com by creating a new static site, connecting the repository, configuring the build command (`npm run build`), and setting the publish directory to `build`. For the backend, allowed origins were set to the frontend URL, sensitive information was added to the`. gitignore` file, and the code was pushed to a new GitHub repository. Deployment involved creating a new Node.js web service, connecting the repository, configuring the build command (`npm install`), and setting the start script to `npm start`. After deployment, the frontend and backend were tested together to ensure functionality and responsiveness on both desktop and mobile devices.

7 Results

The project aimed to create a web-based application for language learning that could be deployed easily, using a solid development stack and providing a good user experience. The results show that these goals were successfully met through careful planning, step-by-step development, and a focus on both functionality and usability.

The project began with the need to move from a mobile app, which had limited features, to a web-based platform that could offer more flexibility and reach. This change was necessary to make the application more accessible and scalable and to provide a better learning experience. The development of the web application followed a clear plan, with each stage contributing to the final product, which not only met the objectives but also improved on the original mobile app.

At the start of the project, a development model, architecture design, and technology stack were defined. These elements were essential to ensure that the application would be reliable and easy to use. A modular design approach was used, breaking the application into smaller parts that could be developed and tested separately before being brought together into the complete system. This approach made the development process smoother and ensured that the application would be easy to maintain and update in the future. For example, components like the `ScoreLevelDisplay` and `SpeakButton` were created as separate modules that could be used in different parts of the application. These modules were tested on their own first to make sure they worked correctly, then integrated into the full application. This modular design was key to making sure the application was ready for deployment without needing further changes.

The architecture design was another important part of the project. A multi-tier client-server architecture was chosen, which provided a strong foundation for handling user data, login, and authentication. This architecture was a good fit for the application’s needs, allowing smooth and secure communication between the client and server. The decision to store English and Finnish word data on the client side was particularly helpful because it allowed quick access to these resources, improving the overall user experience.

The project used the MERN stack, which includes MongoDB for the database, Express.js and Node.js for server-side operations, and React for the user interface. This stack was chosen

because it offered good performance, could grow with the application's needs, and had strong support from the developer community. Each part of the stack played an important role in the project's success. MongoDB provided a flexible and scalable way to manage the application's data, which was important as the number of users grew. Express.js and Node.js handled the server-side tasks, ensuring the backend was strong enough to manage things like data processing and user authentication. React was used to build a user-friendly and responsive interface, making the application easy to use. The fact that all these technologies used JavaScript made the development process smoother because the team could work within a single programming language, reducing the chances of errors and making the development more efficient.

The user interface was designed to be simple and fast, with a focus on providing clear feedback to users. React's component-based structure was very helpful in this regard, allowing the creation of a dynamic and responsive interface. The application was designed to be accessible from any web browser, so users could use it on any device, making it more convenient than the previous mobile app. This broad accessibility was a key factor in improving the user experience, as it meant users could access the application whenever and wherever they needed, without being limited by the mobile app's constraints.

One of the most important improvements over the mobile app was the addition of example sentences. The previous app only showed individual words, which limited the learning experience. The new web application allows users to see example sentences when they check their answers. This feature is important because it helps users remember phrases better and understand how sentences are structured, which is crucial for language learning. To support this functionality, English definitions and examples are retrieved from a free dictionary API, while Finnish meanings and examples are stored internally. This setup ensures quick access to relevant information, enhancing the overall user experience. The feature was designed to meet the needs of users learning both Finnish and English. When a user chooses Finnish as the target language, the application shows a sentence in Finnish with its English translation. When English is selected, the application shows an example sentence in English only. This feature is available on both the word puzzle and dictation pages, greatly enhancing the learning experience by providing context to the words being learned.

Another important feature added to improve the user experience was the `StrikesDisplay` component. This component gives users immediate feedback on their incorrect answers, allowing

them to see their mistakes and learn from them. This feature is simple and easy to use, making the learning process more engaging and helping users improve their language skills. The instant feedback from the `StrikesDisplay` component reinforced learning and motivated users to keep improving.

The web application was enhanced with a precise user ranking system that accurately ranked users based on their scores and levels, even when multiple users shared the same score. This system ensured fairness by assigning identical ranks to users with the same score and provided real-time feedback displaying the user's rank, total users at the same level, score, and current level. These modifications for user ranking and performance analytics were added to the profile page, offering users a clear view of their progress and standings.

Beyond the technical aspects, the project focused heavily on user experience, which was a key factor in the success of the application. The user interface was designed to be intuitive and easy to navigate, with a strong emphasis on providing clear feedback and ensuring users could quickly find the information they needed. The addition of example sentences, the `StrikesDisplay` component, and other user-focused features helped create a more engaging and effective learning environment. The result was a web-based application that was not only functional and reliable but also user-friendly and enjoyable to use.

In the end, the project successfully met its goals by creating a web-based language learning application that is both strong and easy to use. The application was developed with a modular design and built using the MERN stack, which provided scalability and ensured the application was future-ready. The improvements made to the user experience, such as the addition of example sentences and immediate feedback, greatly enhanced the effectiveness of the application. The final product, now hosted on the Render cloud platform, integrates seamlessly with GitHub, allowing for automatic updates and continuous deployment. This complete language learning tool offers users a rich and engaging experience, fully achieving the objectives set at the start of the project.

8 Discussion and conclusion

The primary aim of the thesis was to transform an existing mobile application into a web-based platform and develop it so it’s ready for deployment. This involved several steps: determining the development model, designing the development architecture, selecting the appropriate development stack, and preparing the application for deployment. Ensuring that both the frontend and backend components were robust and reliable was crucial for a smooth deployment. After the preparation, it was important to test the application in real-world scenarios and gather user feedback. As a result, the app was deployed using the Render platform.

This process supported the company's mission to improve the accessibility and user-friendliness of their language learning platform. The development work was carried out systematically, from initial planning to final deployment, with each step carefully executed to meet the thesis objectives. Despite its success, the project faced challenges. Integrating the frontend and backend stacks and ensuring a seamless user experience were difficult tasks. Implementing secure authentication with JWT and the drag-and-drop feature in JavaScript proved particularly demanding. However, these challenges were effectively managed, leading to a robust and reliable application.

The inclusion of example sentences in the web application significantly enhanced its educational value by helping users learn phrases and sentence structures more effectively. This feature addressed a key limitation of the previous mobile app, which lacked this functionality. The addition of the StrikesDisplay module further gamified the learning experience, increasing user engagement. Additionally, user ranking and performance analytics were added to the profile page, ensuring accurate ranking even for users with identical scores and providing a clear view of their progress.

The final product is a web-based language learning application hosted on the Render cloud platform, which supports GitHub integration and automatic updates. This setup offers flexibility and smooth management of ongoing updates. The application fits the company’s mission as a tool for learning Finnish and English. It can be used immediately and for future projects. In the future, the application could be expanded to include more languages and features, enhancing the learning experience. Additionally, incorporating user feedback will help continually improve and adapt the application to meet learners' needs.

BIBLIOGRAPHY

Acharya, V. (2023, July 2). *Mastering asynchronous programming in Node.js. Medium*. https://medium.com/@vishwasacharya/mastering-asynchronous-programming-in-node-js-174cb75272a0

Argüelles, R. D., & Murrugarra, S. E. H. (2018). *Hands-on full stack web development with Aurelia: Develop modern and real-time web applications with Aurelia and Node.js*. Packt.

Boduch, A., & Derks, R. (2020). *React and React Native: A complete hands-on guide to modern web and mobile development with React.js* (Third edition.). Packt.

Bugl, D. (2019). *Learn React Hooks: Build and refactor modern React.js applications using Hooks*. Packt.

Chiarelli, A. (2018). *Beginning React: Simplify your frontend development workflow and enhance the user experience of your applications with React*. Packt.

Express. (n.d.). Express - Fast, unopinionated, minimalist web framework for Node.js. Retrieved April 27, 2024, from https://expressjs.com/

Fedosejev, A. (2015). *React.js essentials: A fast-paced guide to designing and building scalable and maintainable web apps with React.js*. Packt.

Fellows, R., & Liu, A. (2015). *Research methods for construction* (4th ed.). John Wiley & Sons, Ltd.

GeeksforGeeks. (2023). *Unique features of Express JS.* https://www.geeksforgeeks.org/unique-features-of-express-js/

Gorton, I. (2011). *Essential software architecture* (2nd ed. 2011.). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-19176-3

Hallal, J. (2021). *Solution Architecture with .NET: Learn solution architecture principles and design techniques to build modern .NET solutions*. Packt.

Heath, F. (2021). *Professional Scrum Master (PSM I) Guide: Successfully practice Scrum in real-world projects and achieve PSM I certification with confidence*. Packt.

Hoque, S. (2018). *Full-Stack React projects: Modern web development using React 16, Node, Express, and MongoDB*. Packt.

Hoque, S. (2020). *Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js* (2nd edition.). Packt.

Koroliova, E. W. I. (2018). *MERN quick start guide: Build web applications with MongoDB, Express.js, React, and Node*. Packt.

Kothari, C. R. (2004). *Research methodology: Methods & techniques* (2nd rev. ed.). New Age International, Publishers.

Lui, K. M., & Chan, K. C. C. (2008). *Software development rhythms: Harmonizing agile practices for synergy*. Wiley - Interscience.

Martin, R.C. (2018). *Clean architecture: A craftsman's guide to software structure and design*. Pearson Education, Inc.

McClay, N. (2017). *MEAN Cookbook: The meanest set of MEAN stack solutions around*. Packt.

Mehrabani, A. (2014). *MongoDB high availability: Design and implement a highly available server using the latest features of MongoDB*. Packt.

MongoDB. (n.d.). *MERN Stack Explained*. Retrieved May 8, 2024, from https://www.mongodb.com/resources/languages/mern-stack

MongoDB Documentation. (n.d.). *Introduction to MongoDB.* Retrieved May 8, 2024, from https://www.mongodb.com/docs/manual/introduction/

Node.js. (n.d.). *Download Node.js*. https://nodejs.org/en/download/package-manager

OpenAI. (2024). *ChatGPT* (version 2024-05-13, GPT-4o) [Large language model]. https://chat.openai.com/chat

Pasquali, S. (2013). *Mastering Node.js: Expert techniques for building fast servers and scalable, real-time network applications with minimal effort*. Packt.

Phaltankar, A., Ahsan, J., Harrison, M., & Nedov, L. (2020). *MongoDB fundamentals: A hands-on guide to using MongoDB and atlas in the real world*. Packt.

Pillai, A. B. (2017). *Software architecture with Python: Design and architect highly scalable, robust, clean, and high performance applications in Python* (1st edition.). Packt.

Poojary, S., Branas, R., Arora, C., Frisbie, M., & Haviv, A. Q. (2016). *AngularJS: Maintaining web applications: a course in four modules: learn AngularJS and full-stack web development with your course guide Shiny Poojary*. Packt.

Render. (n.d.). *Our Team.* https://render.com/about

Richey, B., Yu, R., Vegh, E., Despoudis, T., Punith, A., & Sloot, F. (2020). *The React workshop: Get started with building web applications using practical tips and examples from react use cases*. Packt.

Stack Overflow. (2023). *2023 Developer Survey: Web frameworks and technologies* https://survey.stackoverflow.co/2023/#most-popular-technologies

Tsonev, K. (2014). *Node.js Blueprints: Develop stunning web and desktop applications with the definitive Node.js*. Packt.

Vite. (n.d.). *Getting Started*. https://vitejs.dev/guide/