UNIVERSITY OF
LEICESTER

School of Computing and Mathematical Sciences

CO3201 Computer Science Project

A Web-Based Task Management System

A dissertation by

Mason C. Harniess

May 2024

Table of Contents

Table of Figures

Declaration

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date, and page(s).

Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date, and page(s).

I understand that failure to do this amount to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Mason Harniess

Signed: 

Date: 02.05.24

Abstract

This dissertation presents the research, development, and evaluation undertaken as part of the CO3201 final year computer science project to create an intuitive task management system designed to enhance user experience in managing tasks beyond that of existing solutions. The project integrates themes and ideas from several research areas, including Cognitive Load Theory, Human-Computer Interaction, Attention Economics, and Interaction Design to address challenges such as information overload and cognitive strain, while assessing existing solutions like Notion and Evernote to identify strengths and weaknesses in design.

The system takes the form of a web application built with ASP.NET Core with Entity Framework Core for the backend web API and database system, and Next.js with React for the frontend logic and user interface, incorporating architecture patterns such as the Model-View-Controller Pattern and Repository-Service Pattern to ensure scalability, maintainability, and robustness.

Key features of the system include intuitive task visualisation, manipulation, and categorization, along with secure user authentication and authorisation. However, due to time constraints and difficulties starting, some features of the application were unable to be implemented, such as calendar functionality, project management, and an alert system. Though lacking in terms of certain features, the application was highly successful in maintaining an intuitive user interface with responsive functionality, and the features that were implemented were done so thoroughly. Future work is suggested to extend functionality and enhance the system's usability further.

1. Introduction

This project builds upon the ideas of *Cognitive Load Theory* (CLT), *Human-Computer Interaction* (HCI), *User Centric Design* (UCD), and *attention economics* to produce an intuitive, accessible, and engaging task management web application. Here, the term *task management* refers to the process of managing, tracking, and executing a task, where a *task* is a piece of work to be done or undertaken. By incorporating the findings from the named interdisciplinary research areas, the application has the potential to alleviate *information overload* and manage *cognitive strain* (discussed in following sections).

The remainder of Section 1 covers the aims and objectives of the project. Section 2 discusses the research context which forms the background of this project. Section 3 briefly examines existing solutions, namely Notion and Evernote. Section 4 discusses the architecture of the system and some key design choices. Section 5 covers the implementation of the system, detailing the frameworks and technology used and focusing on notable code fulfilments. Section 6 provides a description of the testing performed. Section 7 discusses the results with some brief coverage of the results from testing. Section 8 provides a summary and critical analysis of the work completed, including a discussion of project context and a short assessment of personal development through the project. Section 9 concludes the dissertation.

1.1 Aim

The primary aim of this project is to develop an intuitive and modern task management web application, titled *Swift*, that provides an interface for users to manage the lifecycle of their tasks and control their workflows.

In addition to this primary aim, a secondary aim is to incorporate good development practices by adhering to popular software design patterns that facilitate maintainability, scalability, and efficiency. By embracing principles such as the single responsibility principle, open/closed principle, repository design pattern, service layer pattern, and the dependency inversion principle among others, the project seeks to create a robust architecture that can easily adapt to changing requirements and integrate new features without compromising the system's integrity.

1.2 Objectives

This section describes concretely how the aims in Section 2.1 are going to be met, providing a set of quantitative objectives to assess whether the project goals have been met. An objective is a specific, concrete action or outcome that outlines how the broad purposes or aims of the project will be achieved.

To achieve the aims, I am required to complete the following:

1. Produce a frontend *User Interface* (UI) using Next.js web development framework.
2. Produce a backend infrastructure using ASP.NET Core 7 with Entity Framework Core for database management.
3. Implement essential application services:
    a. User Authentication: Users can login, logout, and register.
    b. User Authorisation: Users can be authorised using the *JSON Web Token* (JWT) standard.
    c. Password Security: Passwords are hashed and stored securely.
4. Implement core application functionality:
    a. Task Visualisation: Users can view their tasks in a logical format.
    b. Task Manipulation: Users can create, edit, and delete tasks.
    c. Task Sorting: Users can organise tasks by date, priority, alphabetically, or a custom order.
    d. Task Categorisation: Users can filter their tasks into groups (e.g., school, work, personal).
    e. Task Searching: Users can search through their tasks.
    f. Notifications & Alerts: Users can receive notifications & reminders in relation to points in a task's lifecycle.
5. Implement additional functionality:
    a. Analytics & Productivity Tracking: Users can be provided insights on task completion rates and time management.
    b. Data Exportation: Users can export their data into a standardised format such as JSON.
    c. Feedback & Support System: Users can make inquiries and suggestions through a dedicated channel.

2. Background Research & Context

The research performed as part of this project concerns several key concepts and interdisciplinary fields. This section will briefly discuss the core concepts that comprised this project's background research.

2.1 Information Overload

Information overload refers to the state of having more information available than one can process effectively, leading to decision-making paralysis. Coined by Bertram Gross in 1964 [**1**] information overload has been defined as *'the condition in which the amount of input into a system exceeds the processing capacity of that system'* [**2**]. Research has found that when information overload occurs, decision quality decreases [**3**], [**4**].

The emergence of the internet and a growing dependence on digital platforms have brought forth a new era of information accessibility. However, this has also led to information overload in various areas of life: socialising, news consumption, education, work, advertising exposure — information overload has been identified by several studies to be an issue in these facets of life [**5**]–[**11**]. Soucek & Moser (2010) suggested that self-management strategies like task prioritisation and sequencing can enhance one's own workflow and prevent information overload [**12**].

2.2 Cognitive Load Theory

In the seminal work of George A. Miller, it was proposed that the capacity of immediate memory is bounded, presenting '*severe limitations on the amount of information that we can receive, process, and remember*' [**13**].

Building on Miller's foundational research, John Sweller formulated the *Cognitive Load Theory* (CLT), positing that working memory possesses a finite capacity for the retention and manipulation of new information [**14**]. Sweller's contributions defined two pivotal concepts: *cognitive load*, describing the level of cognitive resources utilised for executing a specific task; and *cognitive strain*, describing the mental fatigue or stress experienced when the demands placed on cognitive resources surpass their processing capabilities.

Expanding upon these concepts, psychologist Daniel Kahneman explored the nuances of cognitive strain, articulating that it is influenced by both the immediate level of exertion and the presence of unresolved demands. Kahneman introduced the concept of *cognitive ease*, characterising it as a mental state wherein information appears familiar, effortless to comprehend, and non-threatening [**15**].

Understanding the principles of CLT, the mechanisms underpinning cognitive strain, and the factors promoting cognitive ease opens avenues to mitigate the risk of inducing strain in users of the task manager.

2.3 Human-Computer Interaction

Human-Computer Interaction (HCI) represents an interdisciplinary field dedicated to exploring the interfaces that connect humans with computers. As detailed by Dix *et al.* [**16**], HCI encompasses all forms of interaction between a *user* — defined as any individual engaged in task completion via technology — and a *computer*, which is broadly characterised to include technologies ranging from conventional desktop computing environments to large-scale computer systems.

A primary objective of HCI is to enhance the usability, accessibility, and overall efficiency of technological systems. Hence, the consideration of HCI principles will prove important for the success of this project.

2.4 Interaction Design

Interaction design is a specialised area within HCI that focuses on designing interactive products, systems, and services. Interaction design places emphasis on shaping the interactive experiences between products and their users, primarily within the digital domain. As described by Winograd (1997), interaction design is about '*designing spaces for human communication and interaction*' [**17**].

In their seminal work [**18**], Preece, Rogers, and Sharp articulate six fundamental usability goals for optimising interaction between humans and computers:

1. Effectiveness: The degree to which the product fulfils its intended purpose and facilitates the completion of user tasks.
2. Efficiency: The degree to which the product supports users to accomplish their tasks in a timely manner with the minimum number of steps.
3. Safety: The degree to which the product prevents the user from entering unintended or dangerous conditions (i.e., mistakenly deleting a file).
4. Utility: The degree to which the product delivers the appropriate features for users to complete their desired tasks.
5. Learnability: The degree to which users can learn to use the functionality of the product.
6. Memorability: The degree to which users can recall how to use the functionality of the product.

These goals are critical in guiding design decisions to enhance the *User Experience* (UX) with the software, ensuring that interactions are intuitive, efficient, and aligned with user needs. They will also provide a meaningful baseline against which to measure success in providing a strong UX.

2.5 Attention Economics

Attention economics is an approach to information management that views human attention in the context of being a finite commodity. As defined by T. Davenport, attention is '*focused mental engagement on a particular item of information*', further explaining the cognitive process as items entering one's perception and being attended to, followed by a decision on whether to act [**19**].

The principles of attention economics are important to consider in designing a strong UX, as the *User Interface* (UI) must facilitate the limits of human attention. If users encounter difficulties in navigating an application or utilising a particular feature, there is a risk that they could switch to alternative software solutions or abandon the effort entirely. UX design is a tool for which the attention economy can be navigated, striving to capture user engagement, foster meaningful computer interactions, and retain user interest amidst a plethora of digital stimuli [**20**].

3. Existing Solutions & Related Works

The area of task management software is one saturated with many existing software solutions. This section will evaluate some of the most popular of these, namely Notion and Evernote, assessing their strengths and weaknesses to gain insight into UX design considerations that must be considered.

3.1 Notion

Notion is a comprehensive organisation and productivity tool designed for notetaking, task management, database management, and various other tasks [**21**]. Its flexibility and wide feature set have made it popular among students, professionals, and teams looking for an all-in-one workspace. Notion is designed to reduce the reliance on separate applications, a goal that, based on user testimonials, has been arguably achieved. For instance, MetaLab, a notable corporate user of Notion, has publicly stated that Notion has allowed them to leave behind nearly a dozen different tools [**22**].

Notion stands out as a versatile and customisable productivity tool that caters to a wide array of user needs. Its all-in-one approach allows users to consolidate their workflows into a single platform, supported by an extensive library of templates and robust collaboration features. This makes it an attractive workflow management tool for individuals and teams seeking an integrated workspace.

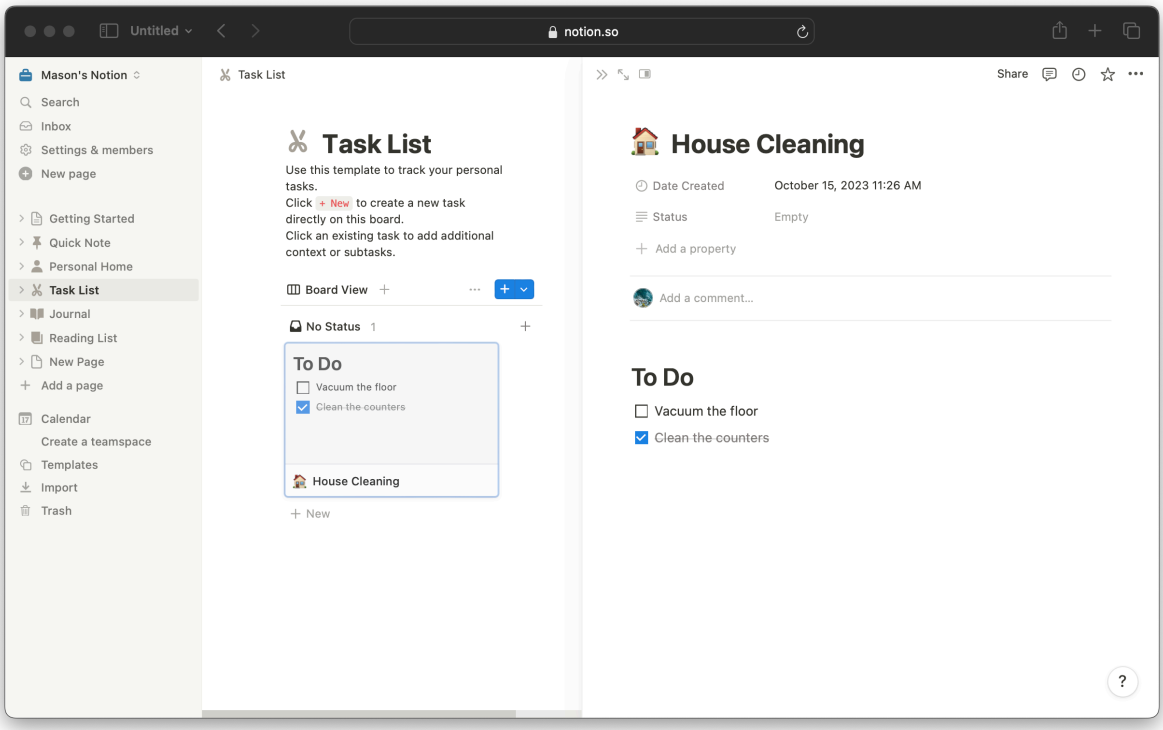

Figure 1 — Notion

However, Notion is not without its flaws. A common complaint is its steep learning curve, with newcomers finding the vast collection of features to be daunting and overwhelming in scope [**23**]. Some users suggest that Notion is mediocre at several things and great at none [**24**], [**25**]. Notion has also been known to have performance issues, especially with larger workloads [**26**].

Swift Task Manager avoids the pitfalls observed in Notion by maintaining a focused and minimalistic approach, often colloquially referred to as the *KISS Principle* ('Keep It Simple, Stupid'). This is a design principle emphasising the importance of simplicity and avoiding unnecessary complexity in design, engineering, and decision-making [**27**].

3.2 Evernote

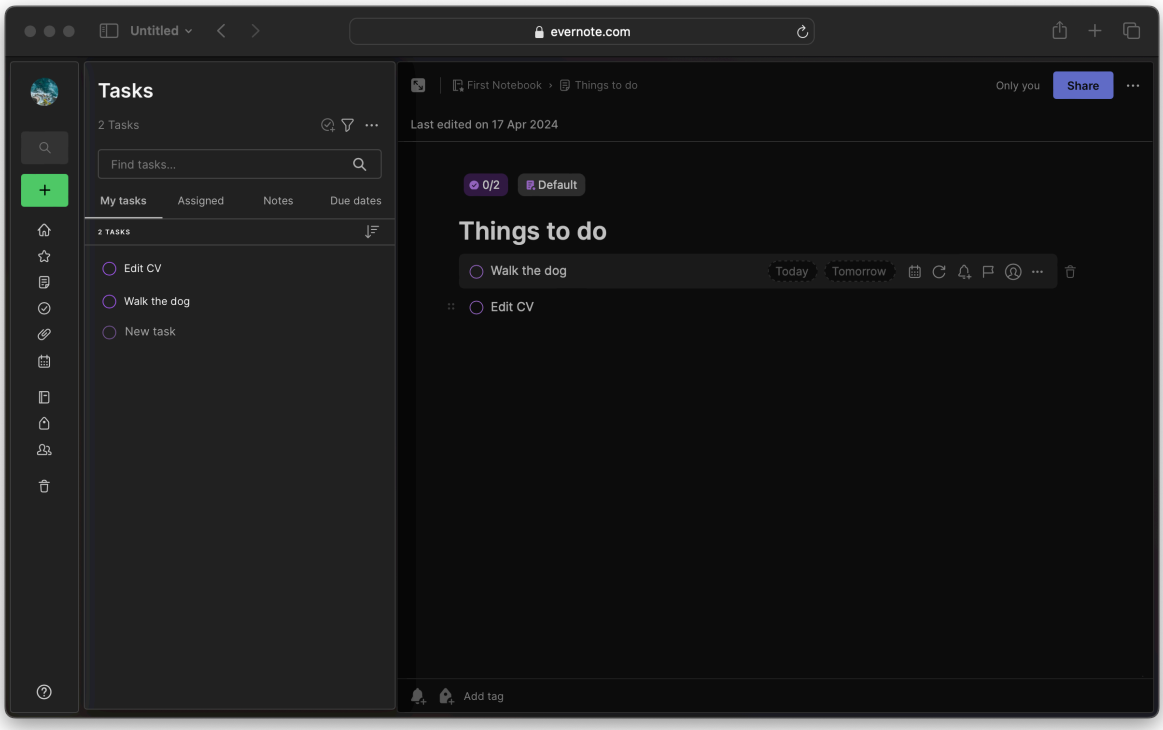

Figure 2 — Evernote

Evernote is a productivity tool designed for notetaking, organisation, task management, and archiving. It allows users to create notes in various formats: drawings, photographs, saved web content, etc. These notes can be organized into notebooks, tagged, edited, given attachments, and exported.

By evaluating Evernote against the fundamental usability goals created by Preece, Rogers, and Sharp, the application's UX design can be assessed. The application has strengths in its effectiveness, utility, learnability, and memorability. Evernote successfully fulfils its intended purpose, delivers the appropriate features for users, is simple enough to learn, and has easily recalled functionality. Evernote's weaknesses lie in its efficiency and safety, having several slow animations in response to user actions,

and lacking recoverability or warnings to prevent the user from entering dangerous conditions, such as accidentally deleting a task.

4. Design

This section details the core design choices and illustrates the system architecture via diagrams.

4.1 User Interaction

A use case diagram illustrates the interactions between users and a system. They can be useful by outlining a systems functionality from a user perspective and are widely used in software and systems engineering to capture the functional requirements of a system.

Figure 3 provides a use case diagram demonstrating the key ways users can interact with the system as of submission of the project.

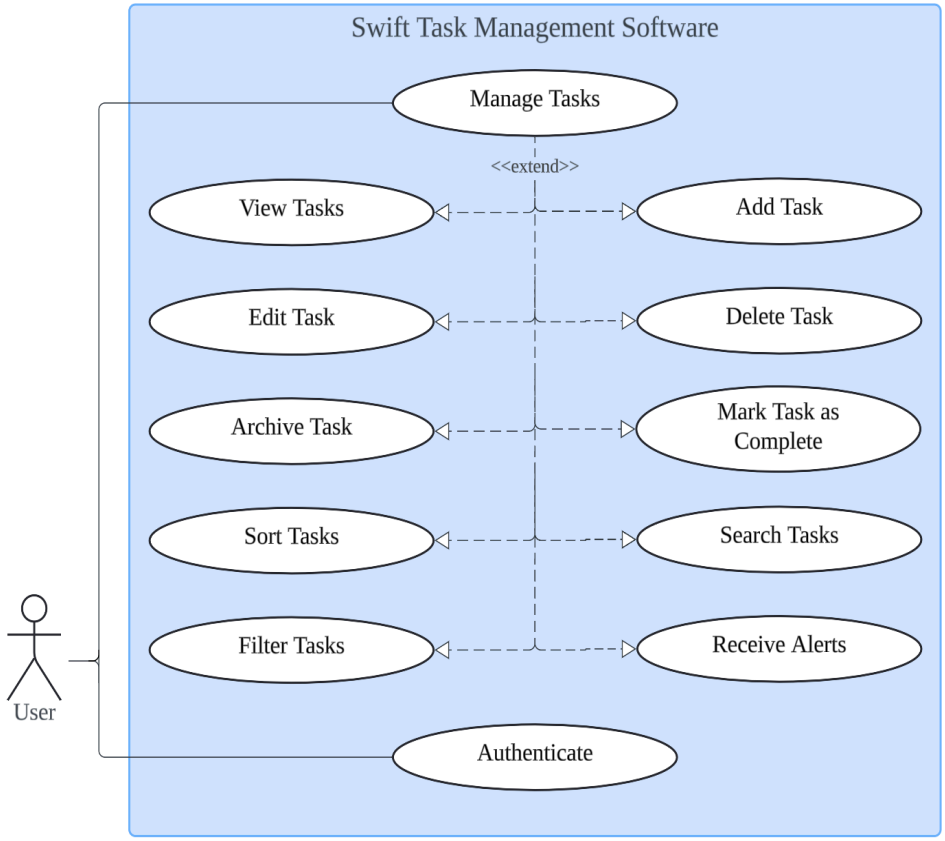

Figure 3 — Use Case Diagram

4.2 System Architecture

This project uses several architectural patterns for the purposes of a clean maintainable project structure. The most notable of these are the *Model-View-Controller* (MVC) pattern, though others are detailed in subsequent subsections.

4.2.1 Model-View-Controller Pattern

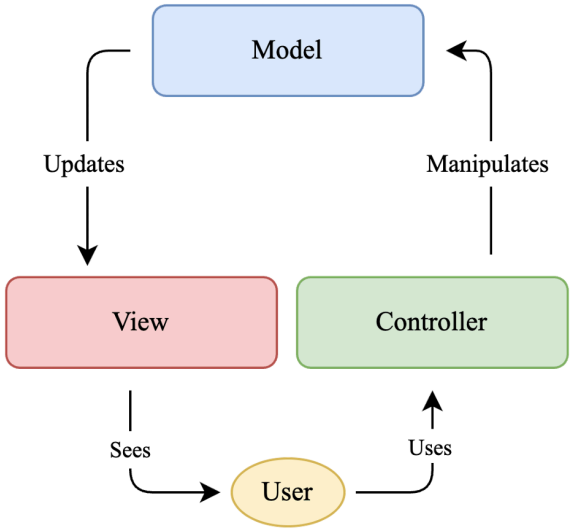

Figure 4 — Model-View-Controller Pattern

Model-View-Controller is a software design pattern, used particularly for designing and organising web applications. MVC separates an application into three distinct components:

1. Model: Represents the application's data structure and handles data, logic, and rules.
2. View: Represents the UI logic of the application, presenting the model's data to the user and sending commands to the controller.
3. Controller: Acts as an intermediary between the model and the view, listening to user input from the view and executing necessary reactions to them.

MVC's primary strength is its adherence to the *Separation of Concerns* Principle (SoC). This is a design principle for separating a computer program into distinct segments, such that each segment addresses an individual *concern*, where a concern is a set of information that affects the code of a program. By adhering to SoC, MVC enables efficient code organisation, allowing simpler maintenance, updates, and scaling.

Swift Task Manager uses this pattern; like any web app, the project comprises a frontend and a backend. The *frontend* (client-side) refers to the part of the application that users interact with directly: the web page design, layout, interactive elements. In terms of MVC, the frontend contains the view. The *backend* (server-side) refers to the *Data-Access Layer* (DAL) and is responsible for storing and organising the application's data. In terms of MVC, the backend comprises the model and the controller.

4.2.2 Repository-Service Pattern

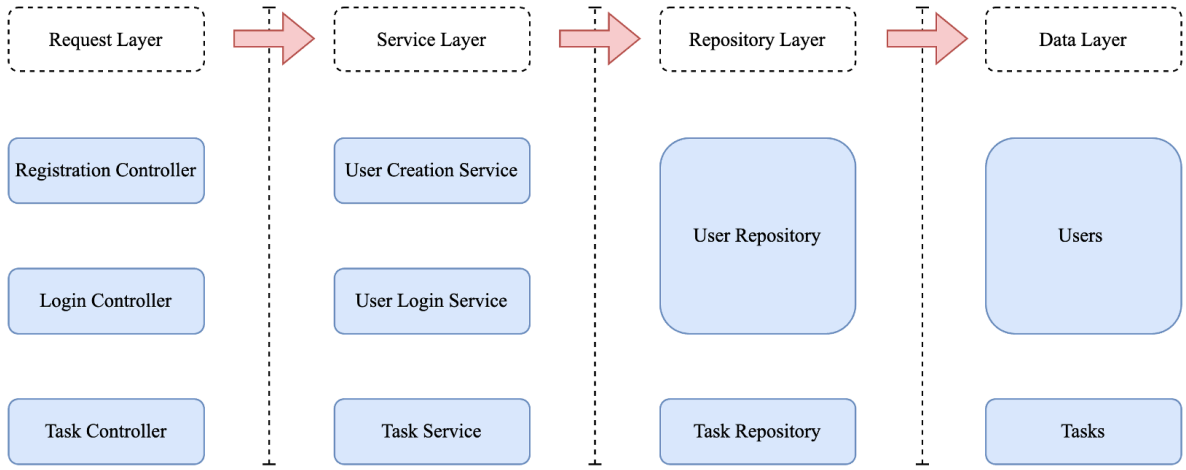

Figure 5 — Repository-Service Pattern

The *Repository Design Pattern* (RDP) provides a means to centralise the way data is stored and received from a data source into one location. It acts as a layer between the business logic and the data access layer, making the application easier to maintain and modify without affecting the business logic or data source. The primary aim of RDP is to provide a standardised way to access, manage, and manipulate data. A key benefit is that it promotes separation of concerns via decoupling. This application uses RPD for all data access logic, namely the logic concerning task and user data retrieval.

Where RPD acts as an abstraction layer between business logic and data access, the *Service Layer Pattern* (SLP) is used to encapsulate business logic and may use the repository layer to implement logic involving the database. Like RPD, the services layer promotes separation of concerns via decoupling, improving maintainability and scalability.

These two patterns can be used in conjunction to form what can be referred to as the *Repository-Service Pattern* (RSP). This pattern has been used in software to maximise the benefits of both. Figure 5 presents a diagram to assist in visualisation of the structure.

4.3 Database Structure

An *Entity-Relationship Diagram* (ERD) illustrates database design through entities and their relationships. ERDs serve as a blueprint for structuring data and defining the relationships between different types of information. Figure 6 displays the working database structure as of submission of the project. The `User` table contains information about users, including the `Email` and `Password` used to authenticate. The `Task` table contains information about user tasks, including `IsCompleted` and `IsArchived`, to mark tasks as completed or archived respectively. A many-to-one relationship exists between `User` and `Task`, as a single user can have many tasks; a

foreign key `UserId` exists in the `Task` table to enforce this relationship.

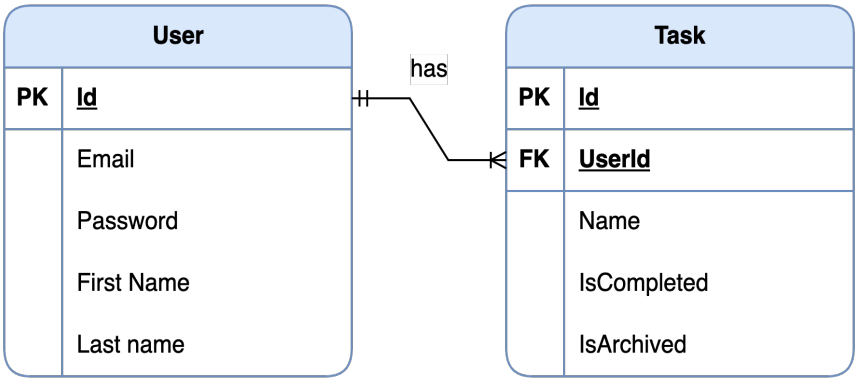

Figure 6 — Entity Relationship Diagram

5. Implementation

This section will discuss some of the core code implementations of the system, detailing the frameworks and technologies used.

5.1 Backend

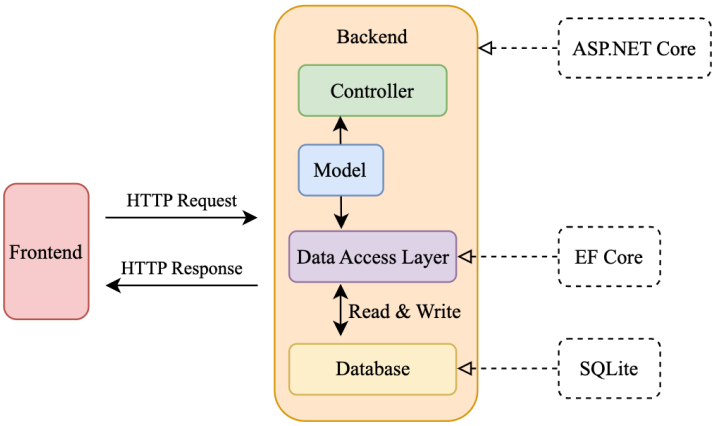

Figure 7 — ASP.NET Core Web API

The backend of this system comprises two key components: the *Application Programming Interface* (API) and the database.

The API of this project uses *ASP.NET Core 7.0* — a free, open-source, high-performance framework for building modern web applications. ASP.NET Core is particularly well-suited for API development, offering a streamlined process that follows the *Representational State Transfer* (REST) architectural style. This style is widely used in the development of web services due to its simplicity, scalability, and statelessness, making it ideal for building APIs that are easily consumed by various clients, including browsers and mobile devices.

The database uses SQLite, a software library that provides a small, fast, and fully featured *Structured Query Language* (SQL) relational database management system. SQLite is designed for single-use machines, making it ideal for this project; however, if this application was to be deployed and was expecting high-traffic usage, switching to a database system like MongoDB may be preferable.

To serve interactions between ASP.NET Core and the database, *Entity Framework Core* (EF Core) is used. This is a lightweight, extensible, open-source version of Entity Framework, Microsoft's *Object-Relational Mapper* (ORM) for .NET, serving as the DAL for this application.

Figure 7 illustrates the basic interaction flow between a client and the backend. The frontend sends a HTTP request to the backend; this could be a request to retrieve, insert, update, or delete data. ASP.NET Core receives the request and uses routing to

determine which controller and action method should handle the request, performed via attribute routing on controllers and actions.

Once the routing logic identifies the appropriate controller and action method, ASP.NET Core binds any request data to the parameters of the action method; this involves parsing any query strings, form data, route data, or JSON payloads, and converting them into .NET objects that the action methods can use.

```
[HttpGet("tasks")]
public async Task<IActionResult> GetTasks()
{
    var userId = GetCurrentUserId();
    var tasks = await
      _taskItemService.GetTaskItemsForUserAsync(userId);
    return Ok(tasks);
}
```

Figure 8 — Get Tasks Action Method Implementation

The action method executes, performing any business/data logic required, and interacts with data through EF Core, which translates *Language Integrated Query* (LINQ) syntax from the action methods into SQL queries that are understandable by the database, while also tracking changes to entity objects and updating the database accordingly. Once EF Core has finished interacting with the database and the action method completes its execution, it typically prepares a result to send back to the client in the form of a JSON object. The client receives and processes the response.

5.2 Frontend

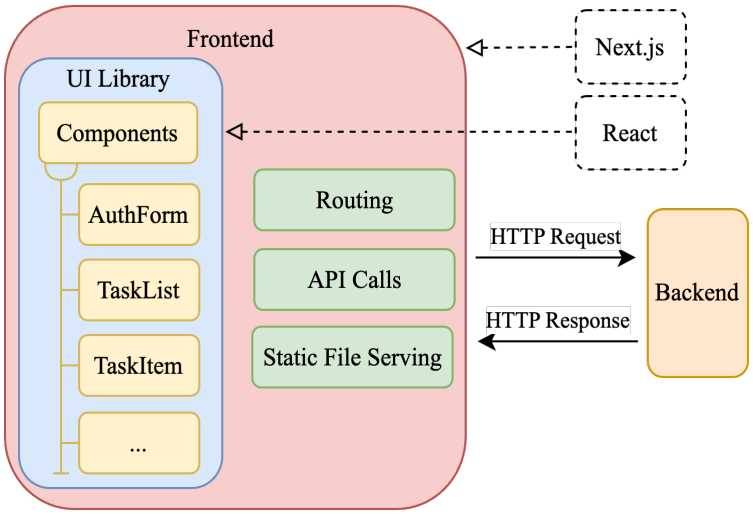

Figure 9 — Next.js Frontend

The client-facing side of the system uses React and Next.js. React is a free, open-source JavaScript library for building user interfaces. It is used to create *Single-Page Applications* (SPAs) where the UI can be updated without reloading the page. React uses a component-based architecture, meaning the UI is broken into small, reusable pieces called components that manage their own state. In the MVC model, React forms the view. Next.js is a framework built on top of React, offering additional features that enable development of complete web applications while still using React for the UI components.

The Next.js frontend sends HTTP requests to the ASP.NET Core backend via the JavaScript Fetch API. To ensure that requests are secure, an asynchronous helper function `secureFetch()` has been created to wrap the native `fetch()` method, adding authorisation and content-type headers to the requests. The function checks if the response was successful by waiting for a 200 OK message; if it was not successful, the function attempts to parse the response body to throw an error with a meaningful message.

```
export const createTask = async (
  name: string) => {
    return secureFetch(`${API_URL}/task`, {
        method: 'POST',
        body: JSON.stringify({ name }),
    });
};
```

Figure 10 — createTask() Request Implementation

Related request methods are stored within their own services; for instance, task-related requests are stored within `taskService.ts`. This service houses several methods for essential *CRUD* (Create, Read, Update, Delete) operations on tasks, enabling functionality to add, view, archive, delete, update, and mark tasks as complete or incomplete.

These methods are called inside of components in response to user actions. For instance, `createTask()` is called within the `TaskList` component; this is a component used to display a list of tasks to the user. `createTask()` is trigged via two other methods within the component: `handleAddTask()`, triggered when the user presses the enter key after typing a task name; and `handleInputBlur()`, triggered when the input field for a new task loses focus and there is a non-empty task name entered. In both scenarios, the `createTask()` method is called with the name that the user has entered for the task. Since `createTask()` is an asynchronous function, it is awaited to ensure that the task creation process is completed before proceeding. Following this, the task list view is refreshed to reflect the updates.

```
const handleAddTask = async (
  e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter' && newTaskName.trim()) {
        await createTask(newTaskName);
        setNewTaskName('');
        await refreshTasks();
    }
};
```

Figure 11 — createTask() Call Implementation

5.3 Security

Swift Task Manager has two key components relevant to security: authentication & authorisation, and password handling. The following subsections will discuss each of these elements in detail.

5.3.1 Authentication & Authorisation

```
export const register = async (
  email: string, password: string, firstName: string, lastName: string) =>
{
    const response = await fetch(`${API_URL}/register`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json', },
        body: JSON.stringify({email, password, firstName, lastName,}),
    });

    if (!response.ok) { throw new Error('Registration failed'); }
    return await response.json()
};
```

Figure 12 — User Registration Request

This project uses an account system to facilitate user-specific task management. Authenticating & authorising ensures that users have their own unique tasks, and do not have access to tasks created by other users. To clarify, *authentication* refers to the process of verifying a user's identity, where *authorisation* refers to determining a user's access rights. To authenticate, a user must first either register or log in, which is done via a form that sends a HTTP Post request containing the user's information. This request is then sent to the relevant controller, which handles either the registration or login attempt. As part of handling the request, the controller calls a method to generate a *JSON Web Token* (JWT). JWTs are compact, URL-safe tokens for securely transmitting information between two parties as a JSON object.

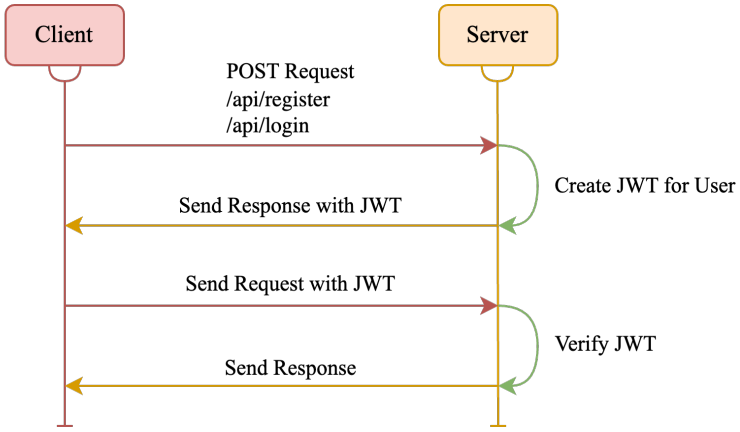

Figure 13 — JWT Process

As shown in figure 13, when a user registers or logs in, the backend generates a JWT that certifies the user's identity. The JWT is sent to the client which then includes the JWT in the header of every request to the server, thus providing a reliable mechanism for the server to verify the client's identify without needing to query the database or maintain session state.

```
public string GenerateJwtToken(UserModel user)
{
    var securityKey = new SymmetricSecurityKey(
      Encoding.UTF8.GetBytes(_configuration["JwtSettings:Key"]));
    var credentials = new SigningCredentials(
      securityKey, SecurityAlgorithms.HmacSha256);

    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, user.Email),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
        new Claim(CustomClaimTypes.UserId, user.Id.ToString()),
    };

    var token = new JwtSecurityToken(
        issuer: _configuration["JwtSettings:Issuer"],
        audience: _configuration["JwtSettings:Audience"],
        claims: claims,
        expires: DateTime.Now.AddMinutes(120),
        signingCredentials: credentials
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
}
```

Figure 14 — JWT Generation

5.3.2 Password Handling

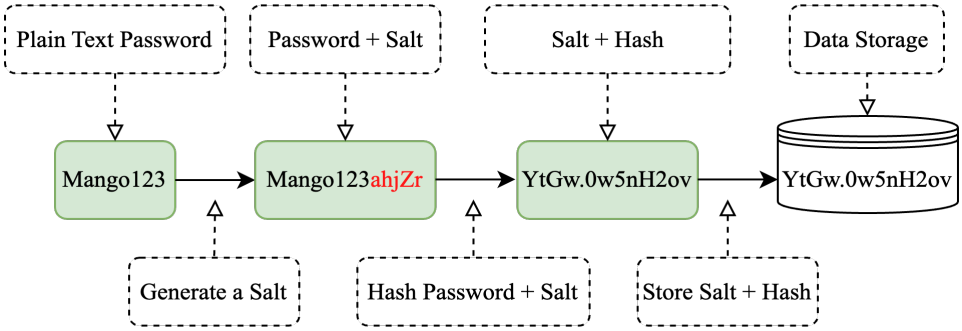

Figure 15 — Password Security Process

User passwords are kept secure via hashing and verifying using modern cryptographic methods. In this application, all password security functionality is found in a password service file, which contains two general functionalities: password hashing and password verification. When a user creates their password, it must be stored securely in the database, as storing passwords in plain text is insecure and leaves them vulnerable to breaches. A hashing technique is used to transform the plain text password into a fixed-size string of characters called a hash.

The first step in the process of storing passwords securely is to generate a random salt — a sequence of random bits which is unique for each password. Its purpose is to ensure that even if two users have the same password, their password hashes will be different.

```
private byte[] GenerateSalt(int size)
{
    byte[] salt = new byte[size];
    using (RandomNumberGenerator rng
           = RandomNumberGenerator.Create())
    {rng.GetBytes(salt);}

    return salt;
}
```

Figure 16 — Salt Generation

The plain text password and salt are then fed into the hashing algorithm. The algorithm used here is the PBKDF2 function with HMAC-SHA256 as its core, iterating the process 100,000 times. This process is called key stretching and significantly increases the time required to calculate each hash, making brute-force attacks impractical.

```
private string Hash(
  string password, byte[] salt, int iterationCount, int numBytesRequested)
{
    return Convert.ToBase64String(KeyDerivation.Pbkdf2
        (
            password: password,
            salt: salt,
            prf: KeyDerivationPrf.HMACSHA256,
            iterationCount: iterationCount,
            numBytesRequested: numBytesRequested
        )
    );
}
```

Figure 17 — Password Hashing

When the user logs in, the password they provide must be verified against the hashed password and salt stored in the database. The first step in verifying a password is to retrieve the stored hash and salt. Then, the password submitted by the user is hashed using the same salt, number of iterations, and hashing algorithm used to create the original hash. The newly generated hash is then compared to the hash retrieved from storage. If they match, the password is confirmed to be correct; otherwise, the password is not correct.

```
public bool VerifyPasswordHash(
  string hashedPasswordWithSalt, string passwordToCheck)
{
    string[] parts = hashedPasswordWithSalt.Split('.', 2);
    if (parts.Length != 2)
    {
        throw new FormatException(
          "The stored password is not in the expected format.");
    }

    byte[] salt = Convert.FromBase64String(parts[0]);
    string storedHash = parts[1];
    string hashed = Hash(passwordToCheck, salt, IterationCount, HashSize);

    return hashed == storedHash;
}
```

Figure 18 — Password Verification

5.4 User Experience

The key element of this project is to create a satisfying user experience, ensuring intuitive design and ease-of-use. UI elements are designed to be responsive and thoughtful in their design, providing the user with solely the information necessary

and nothing more. This section will discuss these elements and the reasons behind design choices.

5.4.1 Registration & Login

The first page that a user is greeted with is the login page. From here, users can navigate to the registration page — which is visually equivalent — or enter their details to log in. Though admittedly simple, a lot of thought was put into this view. Research by Lindgaard [**28**] found that users form an opinion about a web page within the first 500ms and use this conclusion to decide whether they will stay or leave, demonstrating the importance of a strong first impression.

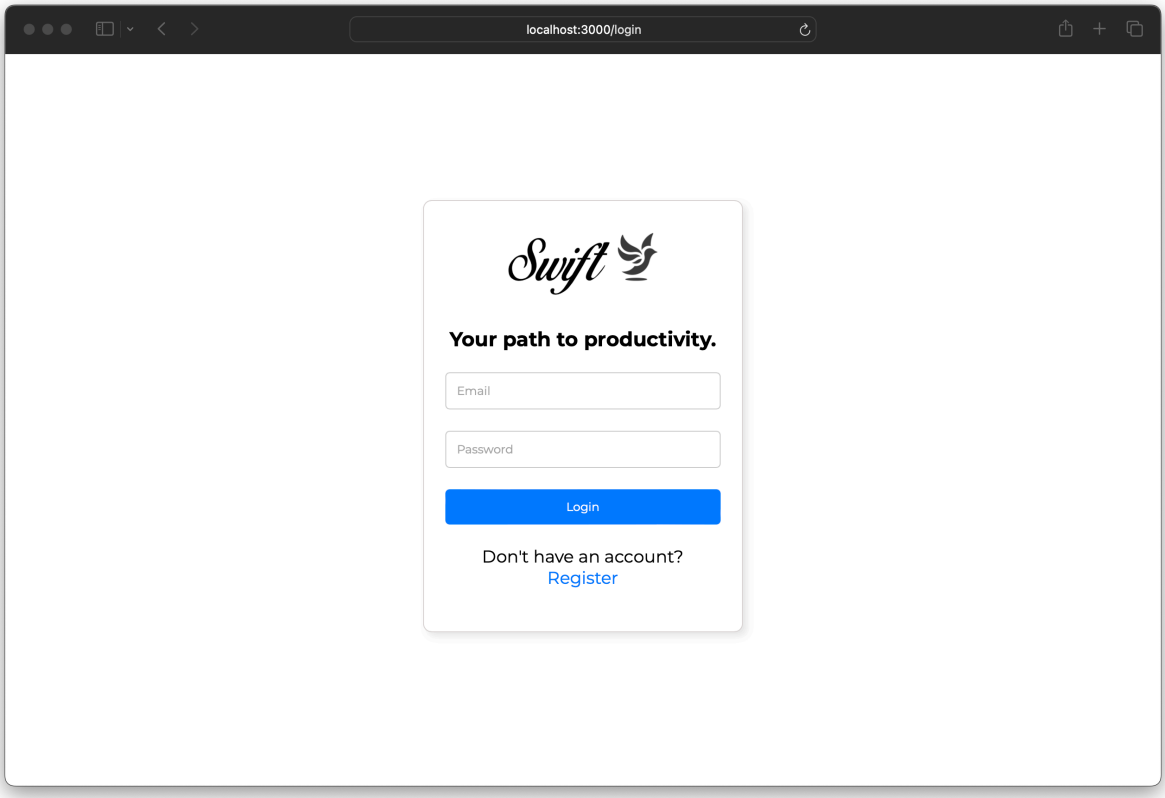

Figure 19 — Login Page

Taking inspiration from Notion's minimalist design aesthetic, the login is visually pleasing and easy to navigate. The layout is logical and uses familiar UI conventions, making the experience easier for users with any experience of logging in to other web applications. The flow from the email input to the password and then to the login button is linear and follows a natural reading pattern. The clear branding establishes confidence in the user that they are in the place they intended to be. There is a strong contrast between background and foreground elements, which ensures compliance with accessibility standards. The use of sans-serif fonts (fonts without decorative strokes, as seen in fonts like Times New Roman) allows for easier reading for persons with certain visual difficulties. The form is centred in the window as to meet most users at eye level. The user is met with feedback when they enter incorrect details, such as an incorrect

password. User feedback is also present when hovering over clickable buttons and links. Switching between the login and registration page demonstrates a consistent design pattern, with both forms being positioned in the same location as to be more intuitive for the user, though being visually distinct enough as to not cause confusion as to where the user is currently at.

5.4.2 Dashboard

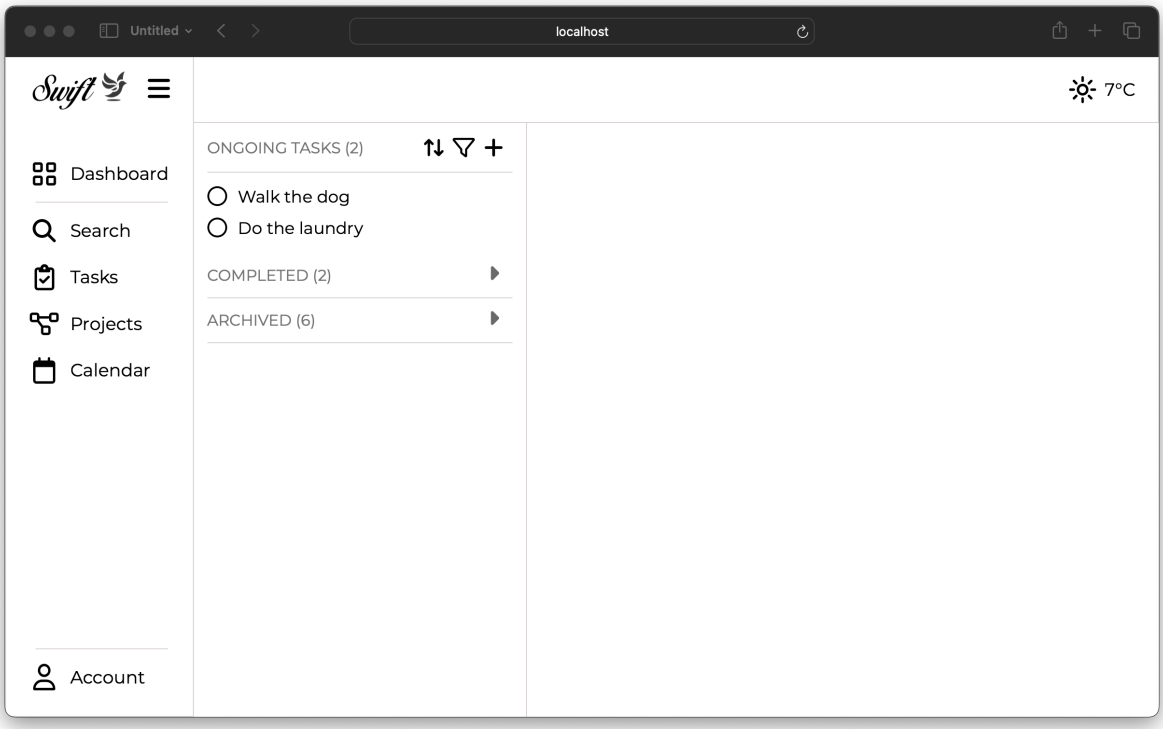

Figure 20 — Dashboard Page

The dashboard page acts as the main workspace for a user of this application, providing all information and navigation options needed for the user to use the application environment. At the top of the page, there is an information bar, acting as a location for weather information. It should be noted that the top bar is unfortunately not finished due to time constraints, though the finished version would have housed other relevant information, including the time, date, and a button to open a notification view. The left of the page houses the sidebar, which contains buttons linking to key areas of the application, including the tasks pane (visible in figure 20), calendar view, project's view, and search functionality. At the bottom of the sidebar, the user can find an Account button which, when clicked, presents the Settings and Logout button. Clicking the Account button again hides these buttons, alleviating the effects of information overload. Like the authentication pages, the sidebar also houses clear branding, further ensuring user confidence that they are in the right place. If the user clicks the menu button located next to the logo, the sidebar will be closed, further eliminating risks of information overload and cognitive strain. The open or closed state of the sidebar is saved in local storage, which guarantees that the sidebar is as the user

left it before they last left or refreshed the site. Other elements on the page (such as the task list) also react to the state of the sidebar, ensuring no content is mistakenly hidden from view while opening or closing it.

```
export const SidebarProvider: React.FC<SidebarProviderProps> =
  ({ children }) => {

    const [isSidebarOpen, setIsSidebarOpen] = useState<boolean>(() => {
        const savedState = localStorage.getItem('sidebarOpen');
        return savedState ? JSON.parse(savedState) : true;
    });

    useEffect(() => {
        localStorage.setItem('sidebarOpen', JSON.stringify(isSidebarOpen));
    }, [isSidebarOpen]);

    const toggleSidebar = () => setIsSidebarOpen(!isSidebarOpen);

    return (
        <SidebarContext.Provider value={{ isSidebarOpen, toggleSidebar }}>
            {children}
        </SidebarContext.Provider>
    );
};
```

Figure 21 — Sidebar State Logic

When the sidebar is closed, tooltips are presented to the user upon hovering over an item, which enables users to get as much information as they need and nothing more. Furthermore, each clickable item responds to hovering by enlarging 10%, which allows the user to be confident they are clicking on the correct element.

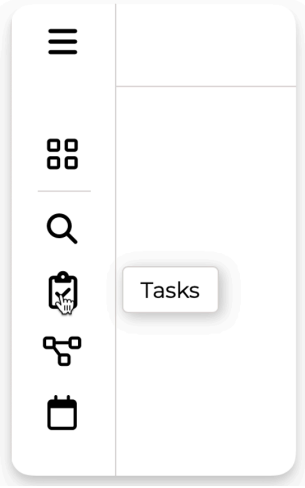

Figure 22 — Tooltips

5.4.3 Task List

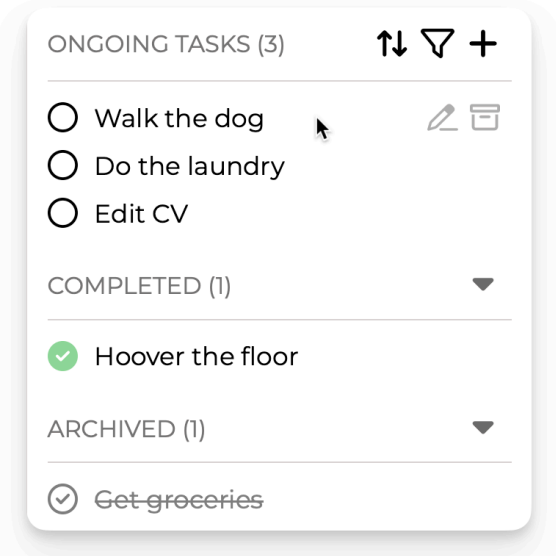

Figure 23 — Task List

The task list comprises the functionality that users will spend most of their time interacting with as part of the application. The task list is ordered hierarchically and splits tasks into sections: ongoing tasks, completed tasks, and archived tasks. The ongoing tasks section is where a user's current work resides; these are the tasks a user is yet to perform. The completed tasks section is for the tasks a user has finished and marked as complete. The archived tasks section is where tasks can be moved should the user decide that they are no longer needed. This structure takes inspiration from Evernote, offering a simple and intuitive interface for task management. Each completed and archived tasks sections can be opened and closed by clicking their respective header. The open or closed state of the completed tasks section remains in local storage similarly to the state of the sidebar.

A task is added by clicking the plus icon in the header of the ongoing tasks section. To the left of the button to add tasks resides buttons for filtering and sorting tasks; however this functionality is not implemented as of submission. A task is marked as complete by clicking the hollow checkbox next to its name, which will then move the task to the completed section; this can be undone by clicking the green check next to a completed task. By hovering over a task, users are presented with buttons for editing and archiving. Tasks can be removed from the archive by clicking the un-archive button on an archived task.

Like the authentication pages, though simple, a lot of thought has been given to UX design in the task list. Splitting tasks into distinguishable sections provides clear segmentation and ease of understanding. A user can quickly look at their task panel and see exactly what needs to be done within seconds. Completed and archived tasks can be hidden from view as to avoid overwhelming the user with unnecessary information. Each button has its own tooltip when the cursor is hovered, assisting the

user in learning what each button does and reminding them if they forget. The buttons to edit and archive tasks only appear when they logically should. Adding a task immediately focuses the user on an input box to name the task. Clicking away from the task after entering a name will automatically save it. Pressing the Enter key after adding a name will save the current new task and open an input box for another, minimising repetitive actions.

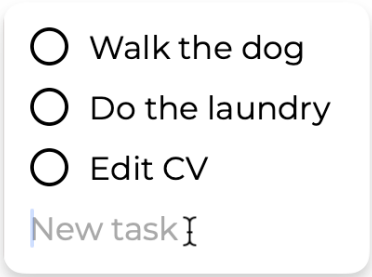

Figure 24 — New Task Input

6. Testing

Software testing is the process of evaluating a software application to ensure it meets the specified requirements and identify any defects in the products. Testing as part of this project involved both automated and manual testing. Automated testing refers to the use of specialised tools to execute preprogramed tests, whereas manual testing refers to performing tests step-by-step without the use of specialised tools or scripts.

6.1 Automated Testing

Automated testing was implemented to ensure reliability and consistency in validating the output of several API endpoints. Postman, an API platform used throughout development — contains built-in automated API testing functionality, automatically running these tests whenever an endpoint is utilised. For instance, figure 25 demonstrates a test written to validate the output of the API endpoint to get a user's tasks by checking if the status code is 200 and the response is an array. A number of tests were written for specific endpoints, ensuring correct outputs when the frontend client makes requests.

```
pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Response is an array", function () {
  let responseData = pm.response.json();
  pm.expect(Array.isArray(responseData)).to.be.true;
});
```

Figure 25 — GetTasks Endpoint Test

6.2 Manual Testing

Manual testing was implemented by interacting with the application step-by-step as a user of the application would do. This was done to ensure all user interactions such as task creation, updating, and archiving behaved as expected. Other manual tests involved verifying that the user interfaces update correctly to reflect changes made to the data and confirming that navigation between different parts of the application is error-free.

7. Results & Discussion

The aim of this project was to develop an intuitive and modern task management web application, titled Swift, that provides an interface for users to manage the lifecycle of their tasks. The design was to take UX research into consideration to facilitate this aim. A secondary aim was to incorporate good development practices by adhering to popular software design patterns that facilitate maintainability, scalability, and efficiency, including the repository design pattern, service layer pattern, and dependency inversion principle. These goals have been achieved, with the caveat that some key features were not able to be implemented due to time constraints. Testing showed that user interactions, UI updates, and API endpoints worked as intended. The project could be improved by implementing the remaining features listed as objectives in Section 1.2.

8. Critical Appraisal

This section will evaluate the work achieved and consider the success of the project, with additional discussions regarding what could have gone better and what would be done differently with hindsight.

8.1 Summary & Analysis of Work Completed

The primary aim of this project was somewhat successful — a web application has been made which facilitates the management of tasks throughout their lifecycle. Users can create an account and manage their own tasks in a dedicated space securely. In this respect, objectives 1–4b have been implemented, leaving 4c–5c incomplete. However, despite it being possible to label this project as 'technically a success', the application does unfortunately lack a lot of features that would make it a complete experience: the dashboard is empty; the weather widget is hard-coded; there is no notification system; tasks cannot be organised into projects, sorted, or filtered; search functionality does not work, and none of the additional features described in Section 1.2 of this report were implemented. This is regrettable and mainly stems from difficulties with getting started and lacking knowledge of where to start (see Section 8.3). Once the project entered full development, features were able to be implemented at rapid pace.

Regarding UX design, this project was highly successful, providing users with a modern and intuitive application. When analysed against Preece, Rogers, and Sharp's size fundamental usability goals, this application achieves all of them in several ways. For instance, the application is effective at facilitating its intended purpose: allowing users to manage tasks. The application is efficient; by using SPA elements (such as the task list, which is simply made visible or invisible when the user clicks the respective button), load-times are reduced and speed is increased. The application is safe, preventing the user from entering unintended or dangerous conditions (i.e., mistakenly deleting a task) by means of task archival. The application has high utility, delivering the sensible features for users to work with tasks and manipulate them. The application is learnable, with its simple design and strong use of tooltips. And finally, the application is memorable due to its strong branding, and uncluttered interface. Ultimately, the learnings from the studied fields — attention economics, human-computer interaction, cognitive load theory, interaction design, and information overload — proved highly valuable in understanding what does and what does not work when designing UIs.

The secondary aim of this project was also highly successful. Several design patterns — the Repository Design Pattern, Service Layer Pattern, Dependency Injection Principle, and Model-View-Controller pattern — were used effectively to create a codebase that is understandable, maintainable, and scalable.

8.2 Discussion of Context

This system is designed for all persons who find themselves unable to manage their workflows, experience feelings of information overload, stress, or cognitive strain, and feel they would benefit from a dedicated tool for organising their work. For this purpose, the application does have the potential to have a positive impact on people and businesses if they find that this kind of tool facilitates a more efficient workflow. The application is relevant in various environments, academic or otherwise.

There are no realistic major risks to society that could stem from this application. A possible concern of users may be privacy, as all user information except passwords are stored in plaintext. It is not likely that the application has impenetrable security if it was to be deployed, meaning user data is at risk should a breach occur.

8.3 Personal Development

This project has provided me with several skills, both technical and otherwise. When starting this project, I had only a vague understanding of how a web application works: I did not know how to authorise or authenticate users; I did not have a meaningful understanding of MVC; I did not know how to securely store and verify passwords; I had some brief experience with ASP.NET Core previously and no experience with React and Next.js. Admittedly, I felt overwhelmed for a long time and lacked confidence in my ability to achieve the work ahead of me. It was for these reasons that I found myself feeling unable to start for a long time, having no idea of where to begin. This assignment has helped me understand that on the surface, a large software project can seem scary and undoable — that is, until I learned to break down the large scary task into its less-scary constituent parts. In this regard, I feel I have grown massively as a programmer, gaining confidence in myself that I can achieve difficult feats when I remember to break them down.

From a technical standpoint, this project taught me how to implement password security, JWT authentication & authorisation, and various other programmatical concepts. My understanding of ASP.NET Core, Next.js, and React (including how they interact) has improved exponentially, providing me with the skills necessary for my desired career in web development.

9. Conclusion

The most crucial aims of the project have been met, though many have not, leaving the project lacking key features. However, the aims that have been met have been done thoroughly and with careful thought and planning to ensure ruggedness in implementation. User experience is one of the strongest elements of the submitted software, which aligns with the overarching goal of usability.

References

[1] B. M. Gross, "The Emergence of Administrative Science" in *The Managing of Organizations: The Administrative Struggle*, vol. 2, New York, United States of America: Free Press of Glencoe, 1964, ch. 31, sec. D, pp. 857.

[2] J. T. Milord, R. P. Perry, "A Methodological Study of Overload," *The Journal of General Psychology*, vol. 97, issue 1, pp. 131–137, 1977.

[3] A. R. Abdel–Khalik, "The Effect of Aggregating Accounting Reports on the Quality of the Lending Decision: An Empirical Investigation," *Journal of Accounting Research,* vol. 11, pp. 104–138, 1973.

[4] M. D. Shields, "Some Effects of Information Load on Search Patterns used to Analyse Performance Reports," *Accounting, Organizations and Society*, vol. 5, issue 4, pp. 429–442, 1980.

[5] P. Hemp, "Death by Information Overload," *Harvard Business Review*, Sep. 2009. [Online]. Available at: https://hbr.org/2009/09/death-by-information-overload (Accessed Feb. 15, 2024)

[6] D. Bennett, "Information Overload, the 24/7 News Cycle and the Turn to Twitter," in *Digital Media and Reporting Conflict*, 1st ed. New York, United States of America: Routledge, 2013, ch. 4, pp. 106–115.

[7] J. Desjardins. "How Much Data is Generated Each Day?" World Economic Forum. April, 2019. [Online]. Available at: https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/ (Accessed Feb. 15, 2024)

[8] G. Gorrell, K. Bontcheva, B. Wessels, "Social Media and Information Overload: Survey Results," 2013. [Online]. Available at: https://www.researchgate.net/publication/237012860_Social_Media_and_Information_Overload_Survey_Results (Accessed Feb. 15, 2024)

[9] G. D. Salyer, "Information Overload: The Effects of Advertising Avoidance on Brand Awareness in an Online Environment," M.S. thesis, School of Professional Studies, Gonzaga University, Spokane, Washington, United States of America. [Online]. Available at: https://www.proquest.com/openview/6624a48d86166431f624fe02915d8bf8 (Accessed Feb. 15, 2024)

[10] M. Hattingh *et al.,* "Factors mediating social media-induced fear of missing out (FoMO) and social media fatigue: A comparative study among Instagram and Snapchat users," *Technological Forecasting & Social Change*, vol. 185, 2022.

[11] B. Mannion. "Information Overload." Risk Management. Available at: https://www.rmmagazine.com/articles/article/2022/06/01/information-overload (Accessed Feb. 15, 2024)

[12] R. Soucek, K. Moser, "Coping with information overload in email communication: evaluation of a training intervention," *Computers in Human Behavior,* vol. 26, issue 6. pp. 1458–1466, 2010. Available at: https://doi.org/10.1016/j.chb.2010.04.024

[13] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information," *Psychological Review,* vol. 63, issue 2, pp. 81–97, 1956. (Accessed Feb. 13, 2024)

[14] J. Sweller, "Cognitive Load During Problem Solving: Effects on Learning," *Cognitive Science,* vol. 12, issue 2, pp. 257–285, April 1988.

[15] D. Kahneman, "Cognitive Ease," in *Thinking Fast and Slow*, New York, United States of America: Farrar, Straus, and Giroux, 2011, ch. 5.

[16] A. Dix *et al.*, "Introduction," in *Human–Computer Interaction*, 3rd ed., Harlow, England: Pearson Prentice Hall, 2003, ch. 1, pp. 3.

[17] T. Winograd, “From Computing Machinery to Interaction Design,” in *Beyond Calculation: The Next Fifty Years of Computing*, Springer-Verlag, 1997, pp. 149–162.

[18] J. Preece, H. Sharp, Y. Rogers, “What is Interaction Design?” in *Interaction Design: Beyond Human-Computer Interaction*, 5th ed., Indianapolis, Indiana, United States of America: John Wiley & Sons, Inc., 2019, ch. 1, pp. 19.

[19] T. H. Davenport, J. C. Beck, “Attention, The Story So Far,” in *The Attention Economy: Understanding the New Currency of Business*, Boston, Massachusetts, United States of America: Harvard Business School Press, 2001, ch. 1, pp. 20.

[20] M. Chainakis. “What is [the] Attention Economy, and Why Should UX Designers Care?” Medium. Available at: https://bootcamp.uxdesign.cc/what-is-the-attention-economy-and-what-is-the-role-of-experience-design-in-it-eca9433cac9 (Accessed Feb. 15, 2024)

[21] Notion Labs Inc. “Your connected workspace for wiki, docs, & projects.” Notion. Available at: https://www.notion.so/ (Accessed Apr. 7, 2024).

[22] Notion Labs Inc. “Design agency MetaLab keeps client work organized & collaborative.” Notion. Available at: https://www.notion.so/customers/metalab (Accessed Apr. 9, 2024).

[23] GetApp User “Notion Reviews.” GetApp. Available at: https://www.getapp.co.uk/reviews/132111/notion (Accessed Apr. 9, 2024).

[24] Reddit User. “One great app for every task or settling on one mediocre app for all our need[s].” Reddit. Available at: https://www.reddit.com/r/Notion/comments/jhqf2d/one_great_app_for_every_task_or_settling_on_one/ (Accessed Apr. 9, 2024).

[25] Hacker News User. “Ask HN: What’s the worst piece of software you use everyday?” Hacker News. Available at: https://news.ycombinator.com/item?id=23807902 (Accessed Apr. 9, 2024).

[26] Nathan Challen. “Why is Notion running so slow?” LinkedIn. Available at: https://www.linkedin.com/pulse/why-notion-running-so-slow-nathan-challen-scmec/ (Accessed Apr. 9, 2024).

[27] Interaction Design Foundation. “Keep It Simple, Stupid (KISS)”. Interaction Design Foundation. Available at: https://www.linkedin.com/pulse/why-notion-running-so-slow-nathan-challen-scmec/ (Accessed Apr. 9, 2024).

[28] G. Lindgaard, G. Fernandes, C. Dudek, “Attention web designers: You have 50 milliseconds to make a good first impression!” *Behaviour and Information Technology*, vol. 25, issue 2, pp. 115–126, 2006. Available at: http://dx.doi.org/10.1080/01449290500330448