**Final Year Project (To-do List Web App)**

**To-do list web app**

**Andrej Gaz P2714384**

**Computer Science**

**Rahim Pasbanigoloojeh**

# 1: Introduction

Task management is an essential component of daily life, both personally and professionally. In a fast-paced digital age where distractions are constant and responsibilities are ever-expanding; individuals increasingly rely on productivity tools to stay organized and focused. Traditional to-do list methods pen and paper, sticky notes, or simple notepad applications have evolved into dynamic digital platforms offering real-time synchronization, personalization, and intelligent task tracking. However, many modern solutions are either overly complex, feature-bloated, or monetized behind subscriptions, limiting accessibility and usability for everyday users. To address these issues, my final year project focuses on designing and developing a full-stack to-do list web application. The objective is to create a modern, intuitive, and responsive productivity tool that empowers users to manage their daily tasks effectively without unnecessary complexity.

The application places usability and performance at its core while also introducing a scalable backend infrastructure and secure data handling practices. It supports all core CRUD (Create, Read, Update, Delete) operations, allowing users to seamlessly create new tasks, view and update existing ones, and remove tasks as needed. Additionally, it introduces subtasks, categorization, and progress tracking featuresenhancing the application’s utility without compromising its simplicity. This project draws inspiration from established task management platforms such as Trello, Notion, and Microsoft To Do. While these platforms offer sophisticated features, the goal here is to balance functionality and minimalism to ensure that the application remains lightweight, accessible, and easy to use for a wide range of users.

## 1.1 Product Overview

## The core objective of this project is to deliver a modern web-based task management application that enables users to efficiently organise, prioritise, and track their daily activities. The design philosophy behind the application is centred on simplicity, clarity, and accessibility. A clean and minimalistic user interface ensures that users are not overwhelmed with unnecessary distractions, allowing them to focus on managing their tasks with ease. This approach supports a seamless and intuitive user experience, which is further enhanced by responsive design principles that ensure compatibility across a wide range of devices, including desktop computers, tablets, and smartphones.

## One of the fundamental features of the application is user authentication. By incorporating a secure authentication mechanism, the system allows individuals to register personal accounts, log in securely, and manage their own data in an isolated environment. This feature not only protects user privacy but also enables personalised functionality, such as task history tracking and progress analytics. Users can review their productivity patterns over time through summaries that reflect completed tasks on a daily, weekly, or monthly basis, helping them gain insight into their behavioural trends and goal completion habits. The technology stack selected for this project supports a full stack JavaScript development workflow, facilitating consistency and improved development efficiency.

## On the frontend, the application leverages React.js, a powerful JavaScript library known for its component-based architecture and efficient rendering via a virtual DOM. React facilitates the development of modular, reusable components that manage their own state and respond dynamically to user interactions. This results in a more responsive and interactive interface. The backend of the application is implemented using Node.js in conjunction with Express.js, forming a lightweight and highly scalable framework for building RESTful APIs.

## This backend infrastructure handles server-side logic, routes client requests, and connects with the database to persist or retrieve user data. For data storage, the application employs MongoDB, a NoSQL document-based database that offers flexibility in handling various data types and structures. MongoDB’s schema-less nature allows the task data to evolve without significant structural rework, making it well suited for this application’s dynamic nature. Altogether, the product is designed to strike a balance between practical usability and technical robustness. It not only functions as a reliable task manager for end users but also serves as a comprehensive learning exercise in full stack development.

## 1.2 Justification for creating the project

## n today’s digitally connected and fast-paced world, productivity tools are no longer considered optional add-ons but rather essential companions for managing daily responsibilities. The constant influx of notifications, messages, and information creates an environment where staying organised can feel overwhelming without the right support systems.

## Task management tools help reduce this chaos by enabling users to offload cognitive burden, clarify priorities, and maintain a focused workflow. However, many existing solutions suffer from feature overload, subscription barriers, or steep learning curves making them less accessible or practical for users seeking a simple, focused experience. This project aims to provide a practical and streamlined alternative in the form of a to-do list web application. The system allows users to register or log in securely, create and manage their tasks, filter and sort them by status or date, add subtasks, and view progress insights all within a clean, minimal interface. These features are implemented with clarity and ease-of-use in mind, ensuring that users can stay productive without being overwhelmed by unnecessary complexity.

## My motivation for undertaking this project stems from a personal and academic desire to deepen my expertise in full stack web development. Over the past few years, I have self-taught foundational web development skills by building websites and completing online coursework. This project provided an opportunity to consolidate that knowledge into a complete and polished system, enabling me to apply theoretical principles in a real-world context. It also challenged me to design a system from the ground up, making architectural decisions, solving technical issues, and implementing both frontend and backend functionality within a single, coherent application. Additionally, this project has allowed me to gain firsthand experience with the full software development lifecycle from planning and system design to development,

## The inclusion of secure authentication, user-specific data isolation, and responsive design has added layers of technical complexity that pushed me to go beyond basic CRUD implementations and engage with more advanced concepts. By developing this application as part of my final year project, I am not only demonstrating my current technical ability but also making a purposeful investment in my long-term growth as a developer. The experience has given me valuable insights into building scalable, maintainable software, and has strengthened my confidence to contribute meaningfully to real development teams. Ultimately, this project reflects both my enthusiasm for software development and my commitment to mastering the tools and techniques required for a successful career in the tech industry.

## 1.3 Objectives and Aims

# The central objective of this project is to design and develop a fully functional to-do list web application that empowers users to efficiently manage their tasks through a streamlined and intuitive interface. At the core of the system lies support for full CRUD operations Create, Read, Update, and Delete enabling users to seamlessly add new tasks, view task details, modify existing entries, and remove tasks when no longer needed. This core functionality serves as the backbone of the application, ensuring users can stay organised and productive with minimal friction. Beyond the basic task operations, the project aims to incorporate a range of user experience and system reliability enhancements.

# A primary goal is to develop a responsive, aesthetically pleasing user interface that adapts fluidly across devices, from large desktop screens to mobile phones. The interface is designed to remain minimalistic and distraction-free while still offering advanced task management features such as subtasks, task filtering, sorting by due dates, and categorisation through labels or tags. Security is also a key objective. The application integrates a secure authentication system to allow users to register accounts, log in, and ensure their data remains private and isolated. User credentials are managed through industry-standard practices, including hashed passwords and protected API endpoints. Robust input validation and error handling are also built into both the frontend and backend systems to reduce vulnerabilities and ensure data integrity.

# The project includes an essential deployment objective as well. Hosting both the frontend and backend on cloud platforms ensures that the application can be accessed publicly, simulating real-world usage conditions. This deployment step enables testing of performance, availability, and user experience outside of the development environment, allowing for iterative improvements based on actual usage feedback. Ultimately, this project also serves as a comprehensive learning exercise. It offers hands-on experience with the full software development lifecycle from requirement gathering and planning to system design, implementation, testing, and deployment. By engaging with modern web technologies including React, Node.js, Express, MongoDB, and cloud deployment platforms, the project will not only deliver a working product but also contribute significantly to my personal and professional growth in the field of software engineering.

# 2: Literature Review (Research)

## 2.1 Methodologies

## The development of this project adopts a modern full stack JavaScript approach, leveraging widely used tools and frameworks that are both industry-standard and academically relevant. This methodology was chosen for its efficiency, cohesiveness, and alignment with best practices in contemporary web development.

## On the frontend, React.js was selected due to its extensive community support, maturity, and effectiveness in building scalable and interactive user interfaces. One of React’s key advantages lies in its component-based architecture, which encourages the decomposition of the user interface into modular, reusable units. This modularity promotes a clear separation of concerns, simplifies debugging, and enables parallel development of components. Additionally, React’s use of a virtual DOM allows for optimized rendering performance by minimizing unnecessary updates to the actual DOM, which significantly enhances responsiveness and provides a smoother user experience. From a design perspective, React enables the development of dynamic single-page applications (SPAs), where page navigation and data updates occur without full-page reloads.

## This is particularly beneficial for a task management system where rapid interaction and real-time feedback are crucial for usability. React also integrates well with state management tools, such as the Context API and third-party libraries, making it well-suited for handling authenticated user sessions and dynamic task data. On the backend, Express.js was adopted within the Node.js runtime environment. Express is renowned for its minimalist and unopinionated structure, which allows developers to build RESTful APIs quickly and with great flexibility.

## The framework’s routing system is intuitive and clean, making it easy to manage the multiple endpoints required for task and user management. Moreover, Express seamlessly supports middleware integration, allowing for the inclusion of key functionalities such as input validation, request logging, error handling, and security enforcement (e.g., token verification for protected routes). Using JavaScript across both the frontend and backend provides a unified development environment, significantly reducing context-switching and enabling the reuse of utility functions and data models where appropriate.

## This not only improves productivity but also leads to a more coherent and maintainable codebase. In this project, shared data formats such as JSON facilitate seamless communication between the React frontend and the Express API. Finally, the chosen stack aligns well with the principles of modern web architecture: it supports modular development, asynchronous data handling, and stateless server interactions. These qualities are essential for building applications that are both scalable and maintainable. By using React and Express in tandem, the application is well-positioned to meet the performance expectations of end-users and the structural demands of long-term growth.

## 2.2 Similar To-do list apps

## The evolution of to-do list applications reflects the broader transformation of productivity tools over the past several decades. Historically, individuals relied on analog methods such as notebooks, planners, and sticky notes to record tasks and organise their time. While effective for basic organisation, these methods lacked scalability, adaptability, and integration with other systems limitations that became increasingly problematic as professional environments grew more complex and collaborative.

## The digitalisation of task management began with early personal productivity software in the 1980s and 1990s. Applications such as Microsoft Outlook introduced users to digital calendars, reminders, and basic task lists, marking a shift toward searchable, prioritised, and more flexible planning tools. Despite their utility, these early systems were primarily designed for individual use and lacked features to support teamwork or dynamic workflows.

## In the contemporary landscape, task management has advanced significantly. Tools like Trello, Notion, and Asana have redefined productivity by combining intuitive interfaces with powerful features for both individuals and teams. These platforms support collaborative workflows by allowing users to share task boards, assign tasks, add comments, attach files, and synchronise with external services such as email, calendars, and cloud storage. Such features foster transparency and accountability in group settings while enhancing individual productivity.

## In designing my own to-do list application, I studied the features and design patterns of these leading platforms. Trello’s card-and-board model, for example, illustrates how task visualisation can aid in project clarity, while Notion’s flexibility in nesting content influenced the decision to support subtasks and hierarchical task breakdowns in my application. Asana demonstrates the value of tagging, deadline reminders, and progress tracking, all of which contribute to user engagement and clarity. While my application does not currently support real-time collaboration or file attachments, the architecture has been designed to support future extensions in those directions.

## Another significant trend in modern productivity applications is the integration of artificial intelligence and behavioural analytics. Some tools provide intelligent suggestions for prioritising tasks or adapting reminders based on user habits. Others incorporate gamification elements to incentivise completion or support wearable integration for mobile productivity. While these features are outside the scope of my current implementation, understanding their value helped shape a roadmap for future development.

## Overall, the shift from static task records to dynamic, integrated productivity ecosystems illustrates how user expectations have changed. Users now demand applications that are responsive, collaborative, and context aware. This research directly informed the design choices for my application. I chose to focus on delivering core functionality CRUD operations, subtasks, tagging, filtering, and a clean interface while ensuring the system is modular and extendable for more advanced features in future iterations.

## By blending simplicity with the most valuable features found in leading task management tools, this project aims to provide a user-focused solution that addresses everyday productivity needs while remaining adaptable to the evolving standards of modern digital workflows.

## 2.3 Task Management Principles

The need for reliable and user-friendly task management tools has grown substantially with the increasing pace and complexity of modern life. Both individuals and organisations require systems that enable them to track responsibilities, prioritise actions, and manage workloads effectively. According to recent surveys and productivity reports, a significant percentage of working professionals use digital task managers on a daily basis, with mobile accessibility and ease of use ranking among the top user demands.

The market for to-do list applications is saturated with a variety of solutions targeting different user bases. Some tools cater to casual personal use with simple interfaces, while others are built for team collaboration and project management. However, user feedback frequently indicates a gap between minimalism and functionality. Many users seek a middle ground an application that is not overly simplistic, but also not burdened with excessive complexity or locked behind paywalls.

This observation directly influenced the direction of my project. The goal was to design a to-do list application that delivers essential task management functionality while remaining accessible and intuitive. Features such as subtasks, tags, due dates, and visual progress indicators meet the core productivity needs without overwhelming the user. Furthermore, offering guest mode access and deploying the application online enhances its accessibility and allows potential users to interact with the system without a barrier to entry.

This approach not only reflects current user expectations but also anticipates the broader need for adaptable, cross-platform tools that can scale with both individual and small team use. As more users shift towards remote and hybrid work settings, the demand for lightweight yet capable web applications continue to rise. By addressing this market niche, my project contributes to the growing ecosystem of digital productivity tools in a way that is both relevant and practically valuable.

# 2.4 Summary of Key Findings

# The literature review highlights three overarching insights that have significantly influenced both the technical design and functional direction of this project.

# 1. A unified JavaScript stack streamlines solo development and future maintainability.

# The use of JavaScript across both the client and server sides allows for a consistent programming model and reduces the cognitive load associated with context switching. React’s component-based architecture encourages the creation of small, reusable UI elements that are easy to test and iterate on. Similarly, Express offers minimalist routing and modular middleware, making it straightforward to structure API logic. This unified stack translates into faster prototyping, easier debugging, and a project architecture that is inherently extendable.

# 2. Modern productivity platforms define user expectations around collaboration, intelligence, and seamless integration.

# Contemporary tools such as Trello, Notion, and Asana have set a standard for real-time interaction, file sharing, and integration with third-party services like email and calendars. While this project initially focuses on core CRUD functionality and user-specific task tracking, the backend is structured using REST principles and middleware layers that will support future features like shared task boards and calendar synchronisation. Likewise, the user interface has been designed with space for reminders, labels, and commentary, allowing smooth evolution without significant architectural changes.

# 3. Behavioural frameworks enhance task prioritisation and user engagement.

# Principles from the Eisenhower Matrix and Getting Things Done (GTD) methodology offer structured approaches to decision-making and task breakdown. The application incorporates these ideas by enabling users to label tasks by priority, nest subtasks beneath broader objectives, and filter tasks based on urgency or importance. These features not only increase user clarity and focus but also promote consistent progress through weekly reviews and visual progress cues. Embedding these behavioural models transforms the system from a basic checklist into a tool that supports thoughtful planning and ongoing personal productivity.

# 3. System Requirements and Project Scope

## 3.1 Functional Requirements

## The To-Do List Web Application is designed to meet a comprehensive set of functional requirements that together provide users with a complete, efficient, and intuitive task management experience. At the core of the system is secure user authentication, which allows individuals to register new accounts and log in using their email address and password. This ensures that all task-related data is isolated on a per-user basis, maintaining privacy and enabling personalised interactions with the system.

## Once authenticated, users gain access to the full range of CRUD operations Create, Read, Update, and Delete across their task list. Tasks can be added, edited, removed, and marked as completed or reverted to pending, with visual indicators clearly distinguishing between completed and incomplete items. This allows users to monitor their task progress and maintain focus on remaining objectives.

## The application also supports the categorisation of tasks through labels such as “Work,” “Personal,” or “Urgent,” which helps users to organise their to-do items in a meaningful and context-aware manner. Each task can include a due date, which ensures that upcoming deadlines are highlighted and easily tracked. These features help users prioritise responsibilities and manage time effectively.

## To enhance usability further, the interface provides filtering and sorting capabilities, allowing users to filter by completion status, label, or due date, and to sort tasks accordingly. This ensures that the task view remains relevant and manageable, even as the number of tasks grows.

## Recognising that many goals consist of multiple actionable steps, the system also supports the creation of nested subtasks, enabling users to decompose larger tasks into smaller, more manageable components. This hierarchy aids clarity and fosters incremental progress toward broader objectives.

## All task data, including user credentials, tasks, subtasks, labels, and associated metadata, is persistently stored in a MongoDB database. This ensures data is not lost between sessions and supports a scalable storage model suited to dynamic user requirements.

## Together, these functional elements form the foundation of a robust task management system tailored to meet the needs of modern users seeking clarity, structure, and flexibility in managing their daily responsibilities.

## 3.2 Non-functional Requirements

## Beyond its core functional capabilities, the To-Do List Web Application must fulfil several non-functional requirements to ensure a high-quality, reliable, and scalable user experience. These requirements address critical aspects such as performance, security, availability, accessibility, maintainability, and deployment flexibility.

## Performance is a key concern for ensuring user satisfaction. Both frontend pages and backend API endpoints are expected to load and respond within two seconds under normal operating conditions. This responsiveness supports smooth task management and prevents user frustration, especially during repeated interactions such as adding or editing tasks.

## In terms of security, the system incorporates best practices for safeguarding user data. Passwords are securely stored using hashing and salting mechanisms, ensuring that raw credentials are never exposed or stored in plaintext. The application also implements comprehensive input validation on both the client and server sides to protect against common threats such as SQL or NoSQL injection, cross-site scripting (XSS), and malformed requests.

## The application is designed for high availability, with a target uptime of 99% over any 30-day period. The underlying architecture supports horizontal scaling, enabling the backend services to handle increasing user loads without requiring fundamental architectural changes. This is particularly important as the application grows beyond its initial user base.

## To ensure inclusivity and usability across a diverse audience, the application interface adheres to the WCAG 2.1 AA accessibility standards. This ensures that users with visual, motor, or cognitive impairments can navigate and interact with the system using assistive technologies, accessible contrast ratios, and keyboard-friendly components.

## Maintainability is achieved through a modular and well-documented codebase. Each component and service is logically separated and follows consistent naming and coding conventions. This structure not only simplifies debugging and testing but also supports the integration of future features and ease of developer onboarding.

## 3.3 Project Scope

Defining the project scope in clear terms helps maintain focus on delivering essential features while laying the groundwork for future growth. In the initial release, development will concentrate on implementing the core task management functions, user authentication, task categorization, due date management, filtering, and the sub‑task hierarchy. The front‑end will be built responsively in React, and the back‑end will consist of RESTful services created in Express and Node.js. Both components will be deployed to cloud hosts for real‑world validation.

# 4 System Design

## 4.1 System Architecture

# Clearly defining the scope of the project is essential to maintain development focus, manage complexity, and ensure successful delivery of a functional and reliable system. The To-Do List Web Application, as implemented in its initial release, is centred on delivering essential features required for effective task management while establishing a flexible and extensible architecture for future enhancements.

# This version of the application focuses on supporting individual users through a complete suite of task management capabilities. It includes secure user authentication that allows individuals to register accounts and log in using their credentials. Additionally, the application supports a guest access mode, enabling users to interact with the core functionality without account creation. This feature is particularly useful for allowing trial access or for scenarios where persistent data storage is not required.

# Users can create, view, update, and delete tasks with real-time interface updates to reflect changes in task status. Each task can be marked as completed or set back to pending, with clear visual cues for tracking progress. The system also allows users to assign categories to tasks and set due dates, which are prominently displayed in the interface to support better prioritisation. Filtering and sorting capabilities enable users to view tasks by category, due date, or completion status, helping them focus on specific subsets of their workload.

# In recognition of the fact that many goals involve multiple steps, the system supports nested subtasks. This hierarchy allows users to break down larger objectives into manageable components and track progress at a more granular level. To enhance user engagement and reflection, the application includes a productivity panel and integrated chart visualisations. These features allow users to review their task completion trends over time and gain insight into their productivity habits.

# The front-end is implemented using React and styled with Tailwind CSS to ensure a responsive and visually consistent experience across devices of various screen sizes. The backend is built with Node.js and Express, and it exposes a set of RESTful API endpoints for managing users, tasks, projects, and subtasks. Middleware is used extensively to handle authentication, validate input, and manage errors, ensuring a stable and secure data flow throughout the system.

# Both the front-end and back-end components are deployed to live cloud environments for the client-side and Render for the server-side making the system accessible for real-world testing and feedback. This deployment enables practical validation of performance, security, and usability in conditions that closely resemble production usage.

# While the current implementation excludes advanced features such as real-time collaboration, calendar integration, and file attachments, the system has been developed with future scalability in mind. Its modular structure and RESTful design allow for straightforward integration of additional functionality in future iterations without the need for major architectural changes.

# 4.2 UML Diagrams

## 4.2.1 Use Case

The Use Case Diagram identifies the primary actors namely, the Registered User and the System Administrator and the key use cases they perform. The Registered User can register, log in, create tasks, edit tasks, delete tasks, mark tasks as completed, add sub‑tasks, and filter or sort tasks. The System Administrator (in a future version) can manage user accounts and monitor system usage. Figure 4.2 provides a visual representation of these actors and their interactions with the system’s core functionality.

## 4.2.2 Class Diagram

## 4.3 Wireframes and UI Design

# 5 Major Components

## 5.1 Frontend

## The frontend of the To-Do List Web Application is developed using React.js, a modern JavaScript library known for its efficient rendering and component-based architecture. This framework was selected for its flexibility, scalability, and ability to deliver dynamic user interfaces that respond efficiently to changes in state. By leveraging React’s modular design, the application interface is constructed from a series of reusable components, each encapsulating a specific area of functionality and interaction.

## Key UI components include TaskList, TaskItem, TaskForm, FilterBar, and LearningSchedule. Each of these serves a dedicated purpose: for example, TaskList dynamically displays the current set of tasks retrieved from the backend or local context; TaskItem handles task-specific display logic and interaction; TaskForm provides the user interface for creating and updating tasks; and FilterBar enables the user to filter tasks by label, status, or date. Additionally, dashboard widgets such as ProductivityChart and ProductivityPanel offer users insight into their task completion history through simple visual analytics.

## State management is handled through a combination of React’s useState and useEffect hooks, which manage component-level state and side effects respectively. For shared or global state such as the authenticated user session or the full list of tasks the application makes use of the React Context API, providing centralised access to state and related actions across nested components. This approach ensures data consistency and supports maintainable, scalable application logic.

## Styling is implemented using Tailwind CSS, a utility-first framework that allows for rapid and consistent UI development through the application of composable classes. Tailwind enables responsive layout handling through its built-in grid and flex utilities, ensuring that the application adapts fluidly across a range of screen sizes, from mobile devices to widescreen monitors. Custom styles are used where needed to support component-specific visual tweaks without introducing inconsistency.

## All interactions with the backend are abstracted through a centralised service layer that uses Axios to handle HTTP requests. This abstraction simplifies the process of making API calls and promotes clean separation between logic and presentation. Asynchronous functions are used to retrieve and manipulate data, and the application gracefully handles success and failure states with loading spinners, toast notifications, and inline error messaging to keep the user informed of system responses.

## Routing is handled by React Router, enabling a seamless single-page application (SPA) experience. It supports navigation between key views such as the dashboard, calendar, project pages, and authentication screens without full page reloads. This routing system contributes to faster page transitions and a more fluid overall user experience.

## Collectively, the frontend architecture prioritises modularity, reusability, responsiveness, and usability, ensuring that the application delivers a professional, intuitive, and efficient user experience. The integration of modern UI frameworks and best practices also ensures that the frontend remains maintainable and extendable for future development.

## 5.3 Database

# The application utilises MongoDB as its primary database system, selected for its schema-less, document-based data model and its ability to scale horizontally with ease. MongoDB is particularly well-suited for applications with dynamic or evolving data structures, such as task management systems, where flexibility in data representation is essential. Its JSON-like storage format aligns naturally with the JavaScript-based technology stack used in this project, allowing for seamless integration between the frontend and backend.

# To interact with MongoDB, the project employs Mongoose, an Object Data Modelling (ODM) library that provides a structured and expressive way to define data schemas and interact with the database. Mongoose enhances developer productivity by offering a layer of abstraction over raw MongoDB operations, allowing for schema definition, input validation, middleware hooks, and built-in query helpers.

# The database schema is organised into four primary collections: Users, Tasks, SubTasks, and Projects. Each collection has been carefully structured with relevant field types and constraints to maintain data integrity and optimise performance. The User collection stores email credentials and securely hashed passwords. The Task collection includes fields such as title, description, status, priority, dueDate, and references to associated subtasks. The SubTask collection models dependent steps linked to parent tasks through taskId references, enabling a nested task structure. Projects can include multiple tasks and provide a container for broader initiatives.

# To support efficient query performance, the database leverages indexes on frequently accessed fields, such as userId, dueDate, and status. These indexes significantly improve the speed of operations like filtering, sorting, and retrieving user-specific data. Mongoose’s pre-save hooks are used to automate common behaviours, such as updating timestamps and enforcing cascading deletions for instance, ensuring that when a task is deleted, all its associated subtasks are also removed.

# The connection to the database is managed using a singleton pattern, ensuring that only one active connection is maintained throughout the lifecycle of the application. This pattern avoids connection bloat and supports scalability, while automatic reconnection logic handles transient connectivity issues gracefully.

# By centralising all database interactions within dedicated Mongoose model services, the backend remains decoupled from direct storage logic. This architectural choice simplifies future upgrades, supports potential multi-database configurations, and enhances maintainability by isolating persistence concerns from business logic.

# The chosen database approach provides a strong balance between structure and flexibility, supporting both current functionality and future extensibility. It ensures data consistency, performance, and scalability three critical attributes for the long-term success of a web-based productivity application.

# 6 Development Lifecycle

## 6.1 Methodology Used

## The development of this project followed an Agile methodology, with specific alignment to the principles of the Scrum framework. This approach was chosen to support an iterative and adaptive workflow, enabling consistent progress while remaining flexible in the face of evolving technical requirements and feedback.

## Work was structured into two-week sprints, each beginning with a sprint planning session where tasks were identified, prioritised, and assigned estimated effort levels. These tasks were derived from a project backlog that captured both functional and non-functional requirements. During planning, high-priority features such as user authentication, task CRUD functionality, subtask integration, and productivity tracking were scheduled for implementation based on available capacity and project milestones.

## Each sprint concluded with a review and retrospective session, in which completed deliverables were evaluated, and lessons learned were documented. These retrospectives provided an opportunity to identify bottlenecks, refine the development process, and adjust future goals. By revisiting both successes and challenges at the end of each sprint, the development process remained focused, efficient, and responsive to feedback.

## In addition to internal reviews, regular meetings with the project supervisor played a critical role in maintaining academic alignment. These check-ins provided a structured opportunity to validate progress, clarify ambiguities, and receive constructive feedback on both technical and design decisions. Supervisor input helped to ensure that the application remained consistent with academic standards and module expectations.

## The Scrum framework’s emphasis on iterative development and continuous delivery proved to be especially effective for managing the full-stack nature of the project. It allowed frontend and backend features to be developed in parallel while still maintaining a cohesive and stable application. This methodology also supported early deployment and testing in a live environment, enabling real-time feedback and refinement throughout the development cycle.

## Overall, the adoption of Agile Scrum principles contributed significantly to the project's success by supporting structured planning, continuous evaluation, and adaptive implementation each of which played a key role in delivering a robust, user-focused application within the given time constraints.

## 6.2 Stages of Development

## The development lifecycle of the To-Do List Web Application followed a structured sequence of stages aligned with the Agile methodology and informed by Scrum practices. Each stage contributed to the systematic progression of the project, from initial requirements gathering to live deployment and post-launch validation.

## The process began with a detailed planning phase, during which high-level project goals were translated into actionable user stories. Core features such as user authentication, task CRUD operations, subtask nesting, filtering mechanisms, and guest access functionality were identified and prioritised based on complexity, dependencies, and overall importance to user experience. Each user story was then decomposed into smaller, manageable development tasks and assigned a relative effort estimate. These tasks formed the basis of the project backlog, which was refined and reassessed at the start of each sprint.

## Following planning, the project moved into the design phase. During this stage, essential system documentation was developed, including UML diagrams (use case, class, and sequence diagrams), wireframes, and high-level architectural plans. The goal of this phase was to visualise how different components would interact, ensure modularity in design, and align with scalability and maintainability goals. Special consideration was given to data relationships, routing flows, and the layout of key user interface components such as the dashboard, task forms, and productivity panels.

## Development occurred iteratively across both the frontend and backend, with the same developer managing both areas to maintain cohesion. The frontend, built in React, focused on building reusable components, state management using hooks and context, and integration with API endpoints. The backend, constructed with Express and Node.js, handled route definitions, controller logic, data validation, and communication with the MongoDB database via Mongoose. GitHub was used for version control, with regular commits marking incremental progress and supporting clear sprint retrospectives. Throughout implementation, features were continuously tested and improved based on feedback and code reviews.

## 6.3 Validation and Verification

Validation and verification were integral to ensuring a robust and reliable product. A test‑driven development (TDD) approach was used for backend services: unit tests were written in Jest to define expected behavior of controllers and service methods before implementation. Frontend components underwent manual smoke testing to confirm visual layouts and user interactions matched design specifications. Input validation routines were verified through both automated tests and exploratory testing scenarios, checking edge cases such as invalid dates or missing required fields. Integration tests simulated end-to-end API calls to ensure that the frontend and backend communicated correctly and that data persisted as expected. These validation activities, coupled with code reviews and continuous integration pipelines, ensured that defects were identified early and resolved promptly, delivering a stable release by project deadlines.

# 7 Implementation

## 7.1 Key Features Implemented

## The application’s functionality was brought to life through the development of a range of carefully integrated features, each designed to support a seamless and productive task management experience. The feature set reflects a balance between technical robustness and user-centric design, ensuring that the application meets both functional requirements and usability standards.

## At the foundation of the system is a secure user authentication and authorisation mechanism. Using JSON Web Tokens (JWT), the application enables users to register accounts and log in securely, ensuring that each session is authenticated and that data is isolated by user identity. This security layer also includes token verification middleware on the backend, which protects access to task data and prevents unauthorised interactions with user-specific resources.

## The core of the application revolves around a complete CRUD (Create, Read, Update, Delete) cycle for tasks. Users can create new tasks by entering relevant details such as title, description, due date, and category. Existing tasks are displayed dynamically in a structured list, with the ability to update or delete any entry as required. This interaction is handled through a RESTful API on the backend, with state changes on the frontend managed via React hooks and context.

## To support more complex task structures, the application introduces nested subtask functionality, allowing users to break larger goals into smaller, manageable components. Subtasks are directly linked to parent tasks and inherit their visual and logical context within the interface. This feature supports granular planning and aligns with modern productivity frameworks such as Getting Things Done (GTD).

## The system also offers filtering and sorting capabilities, enabling users to refine their task view based on specific criteria. Tasks can be filtered by completion status (e.g. pending or completed), due date, or assigned category. This empowers users to stay focused on relevant tasks and enhances the overall clarity of the user interface.

## Changes to task data are reflected in real-time on the frontend through efficient React state management. This eliminates the need for full page reloads and ensures that user interactions result in immediate, responsive feedback. Success and error states are clearly communicated to the user through visual indicators and inline messaging.

## Lastly, the application is designed with responsive layout principles, ensuring accessibility across various devices and screen sizes. The interface is built with a mobile-first mindset, using Tailwind CSS and modern layout techniques such as flexbox and grid systems. As a result, the application delivers a consistent and intuitive experience whether accessed from a desktop, tablet, or smartphone.

## Together, these features form a comprehensive and user-friendly task management solution, fulfilling the goals outlined in the project’s early planning stages while also leaving room for future feature expansion.

## 7.2 Challenges and Solutions

## Throughout the development of the To-Do List Web Application, a number of technical and architectural challenges emerged. These obstacles required careful analysis and the application of creative problem-solving strategies to maintain functionality, reliability, and user experience.

## One significant challenge was state synchronisation between the frontend and backend, particularly when implementing optimistic UI updates. In some scenarios, user actions such as creating or updating a task would visually update the interface before receiving confirmation from the backend. This occasionally resulted in data inconsistencies if an API request failed or was delayed. To address this, a standardised response structure was implemented in the backend API, and a centralised request handler was introduced in the frontend. This handler interpreted response statuses and managed UI state accordingly, rolling back visual changes in the event of errors to maintain consistency.

## Another complex area was the implementation of robust authentication workflows. While JSON Web Tokens (JWTs) were used to secure user sessions, ensuring their secure transmission and storage was non-trivial. Managing session persistence without exposing sensitive tokens in localStorage required careful configuration. Middleware was crafted to handle JWT verification and token refresh, preventing unauthorised access and reducing the risk of token expiry disrupting user experience. Consideration was also given to future support for HTTP-only cookies, although the current implementation uses token storage in a secure, restricted client-side context.

## Designing the subtask data model presented a performance bottleneck during development. Initially, subtasks were embedded directly within task documents, but this limited flexibility and made querying inefficient. The structure was redesigned using a relational approach, linking subtasks to parent tasks via taskId references. To improve query performance, indexes were added to key fields, and batch retrieval of subtasks was optimised using Mongoose’s populate() method. This reduced load times and improved the efficiency of nested data access.

## During local development, Cross-Origin Resource Sharing (CORS) errors became a recurring issue when running the frontend and backend on different ports. These were resolved by explicitly configuring the Express backend to accept requests from whitelisted frontend origins during development, while maintaining security in production.

## The separation of frontend and backend environments also introduced challenges during deployment. Vercel and Render each have their own configuration models, requiring careful alignment of environment variables and routing paths. To simplify this, environment variable management was centralised and mirrored across development and production environments. A Docker Compose template was also introduced during the deployment phase to simulate real-world deployment scenarios locally, making it easier to spot discrepancies between environments before going live.

## A particularly unique challenge was the implementation of guest mode, which allowed unauthenticated users to use the app temporarily without creating an account. Ensuring that guest users interacted exclusively with localStorage, while authenticated users interacted with the backend via API calls, required conditional logic across components and request flows. This included separating task creation and retrieval methods depending on the user state and ensuring that task data persisted only for the duration of the session for guests, while remaining secure and permanent for logged-in users.

## Each of these challenges contributed to a deeper understanding of practical full-stack development, and the solutions implemented ensured the application remained stable, scalable, and user-friendly across a range of usage scenarios.

## 7.3 Code Samples and Descriptions

# 8 Testing and Evaluation

## 8.1 Testing Strategy

The testing process for this project was primarily manual and conducted through real-time interaction with the user interface during development. As the sole developer, I continuously tested features as I implemented them, using the browser’s developer tools and built-in debugging utilities to verify expected behaviour. This included using console.log() statements to track function calls, monitor data flow, and confirm that key actions were being triggered correctly.

For example, when building the guest login functionality, a console.log() message was used to verify that the correct method was being invoked, the authentication token was cleared, and the application correctly entered guest mode. This approach allowed me to isolate and resolve issues quickly during implementation.

Testing also included verifying the correct behaviour of all CRUD operations, user authentication flow, guest session logic, task filtering, and responsive design. These tests were done directly in the browser, simulating user interactions such as logging in, creating tasks, editing them, marking them as complete, and deleting them. State changes and UI updates were closely observed to ensure data integrity and user feedback.

## 8.2 Test Cases

A computer screen shot of text

AI-generated content may be incorrect.

All core features were manually tested through direct interaction with the application. Guest login was confirmed using console logs and localStorage checks. User registration and login worked as expected, including error handling for invalid credentials. Tasks could be created, edited, deleted, and marked as complete or incomplete, with updates reflected immediately in the UI. Filtering and sorting behaved correctly, and the layout remained responsive across different screen sizes. Overall, all key functionalities performed reliably during testing.

## 8.3 Evaluation

The application performed reliably and consistently across all tested workflows during development. Manual interaction testing provided sufficient coverage of the core functionality, including authentication, task management, subtask nesting, and real-time UI updates.

Browser console logs were instrumental in tracking the flow of data and confirming the execution of event-driven functions. For instance, console messages helped verify that the guest mode login function was behaving as intended and that the correct values were being stored or removed from localStorage.

The absence of automated testing tools did not negatively impact the validation process, as each core feature was tested extensively through usage scenarios that mirrored real user behaviour. These included creating tasks as both a guest and a registered user, testing the separation of data between the two modes, and checking for persistence and feedback after updates or deletions.

Overall, the system demonstrated strong reliability during these tests and met the functional and usability requirements defined during planning. While future development could benefit from introducing automated tests to cover edge cases and ensure long-term stability, the current implementation has proven to be stable and fully operational under normal usage conditions.

# 9 Critical Analysis and Reflection

## 9.1 What Worked

Several aspects of the To-Do List Web App development aligned effectively with the project’s initial goals. Choosing a full-stack JavaScript approach using React for the frontend and Express/Node.js for the backend proved highly productive. Using a single language across the stack reduced context switching and allowed for faster prototyping and debugging. React’s component-based architecture enabled me to build and test UI elements in isolation, while Express’s simple routing and middleware support streamlined the development of secure RESTful APIs. Implementing JWT-based authentication added user security with minimal complexity. The responsive design and clear task status indicators (such as completed vs pending) contributed to a user-friendly experience, with informal feedback confirming that the interface was intuitive and visually clear.

## 9.2 What went wrong?

Despite the project’s overall success, several challenges arose that impacted development time and required unexpected adjustments. One major issue was the complexity of handling nested subtasks in MongoDB, which required multiple schema revisions and more advanced query logic than initially anticipated. Additionally, implementing real-time UI updates exposed flaws in my state management, occasionally causing data mismatch between the frontend and backend. Deployment to cloud platforms also introduced difficulties, particularly with CORS configuration and inconsistent handling of environment variables, which differed between local and hosted environments and demanded extra testing and troubleshooting to resolve.

## 9.3 Improvements and Lessons Learned

## This project was a learning experience from start to finish. Since I was building many parts of the system for the first time, I often encountered errors or situations I didn’t fully anticipate. Instead of having everything perfectly planned from the beginning, I figured things out as I progressed. Whether it was fixing broken state logic, adjusting database structures, or solving deployment issues, I kept going by testing, researching, and learning through trial and error. Although this approach led to some delays, it also gave me a much deeper understanding of how full-stack applications work in practice. Looking back, I can see how starting with more structure or documentation might have helped, but pushing through problems as they came up taught me far more than following a rigid plan

## 9.4 Future Work

As I continue to develop my skills, there are several realistic ways this application could be improved in future versions. One area would be adding shared task boards, so that multiple users can see and interact with the same list. Although I haven’t worked with real-time updates yet, this is something I’d be interested in learning more about and could explore later. Another idea would be to add reminders, notifications, or calendar integration to make the app more useful in everyday life. Improving guest mode to allow longer-term session storage or allowing users to easily convert from guest to registered accounts could also enhance usability. On the technical side, I’d like to explore making the backend more scalable for more users and improving the speed and reliability of certain features. All of these ideas build naturally on what I’ve already created and give me a clear direction for future development.

# 10 Conclusion

The development of the To-Do List Web Application has been a valuable and rewarding experience, both technically and personally. This project gave me the opportunity to design and implement a full-stack web application from the ground up something I had never done at this scale before. By combining React on the frontend with Node.js and Express on the backend, and integrating MongoDB for data storage, I was able to create a functional, responsive, and user-friendly productivity tool that meets the core needs of task management.

Throughout the process, I encountered numerous challenges, including handling state management, dealing with authentication logic, and managing deployment configurations across different platforms. While many of these issues required additional time and problem-solving, working through them allowed me to develop stronger coding habits and a more thorough understanding of how web applications operate in real-world environments.

The finished application includes features such as secure user authentication, task and subtask management, guest mode access, filtering and sorting, and a responsive interface suitable for both desktop and mobile use. These features not only fulfil the project's original requirements but also provide a solid foundation for future improvement and scalability.

Beyond the technical deliverables, this project has strengthened my confidence in full-stack development and has reinforced the importance of planning, testing, and iterating during the software lifecycle. It also helped me improve my problem-solving abilities and become more comfortable learning and applying new technologies on the go.

Overall, the project successfully demonstrates my ability to take an idea from concept to deployment and has prepared me for future challenges in software development whether in academic, personal, or professional contexts.