2026

AUTOMATED TASK REMINDER & TRACKING APPLICATION

ORGANIZATION: INFOSYS SPRINGBOARD
MENTOR: AMAN JACHPURE
INTERN: KOMMU HEMASREE

Chapter 1: Introduction

The development of the **Automated Task Reminder & Tracking Application** marks a significant milestone in my journey as an intern at Infosys Springboard. In today's fast-paced digital environment, managing personal and professional commitments has become increasingly complex, leading to a growing demand for streamlined productivity tools. This project was conceptualized to address the fundamental need for a centralized, intuitive, and highly accessible platform where users can organize their daily activities without the fear of missing critical deadlines. By leveraging modern full-stack technologies, the application aims to bridge the gap between simple manual list-making and complex, enterprise-level project management software.

Throughout the internship, the focus remained on creating a solution that is not just functional but also resilient and user-friendly. The application serves as a comprehensive ecosystem that allows for real-time task tracking, automated reminders, and progress visualization. It was designed with the philosophy that a tool should simplify work, not add to it; therefore, the User Interface (UI) was meticulously crafted to ensure that the transition from a "Pending" to a "Done" state is seamless and rewarding. This report documents the end-to-end lifecycle of this development, from the initial environment configuration to the implementation of advanced security protocols.

Under the mentorship of **Aman Jachpure**, the project followed professional industry standards, including rigorous adherence to naming conventions, architectural patterns, and security best practices. The internship provided an invaluable opportunity to work in a simulated corporate environment, where code quality, documentation, and deadlines were as important as the logic itself. This introduction sets the stage for a detailed technical exploration of how the application was built, the challenges that were overcome, and the specific Agile methodologies that guided the process.

Furthermore, the project serves as a practical application of the concepts learned during my Bachelor of Technology studies at GRIET. It integrates diverse domains such as Web Development, Database Management, and Network Security into a single, cohesive product.

By focusing on "Automation," the application moves beyond a static to-do list, incorporating server-side triggers and email services to keep the user engaged and informed about their responsibilities.

Finally, this documentation is structured to provide a clear roadmap of the technical decisions made during the development. It explains why specific frameworks like **Spring Boot** were chosen over alternatives and how the **MySQL** database was optimized to handle relational data efficiently. As we move into the following chapters, we will dive deeper into the specific objectives that defined our success and the technological backbone that makes this application a robust tool for modern productivity.

Beyond the technical implementation, this project explores the intersection of **Time Management Psychology** and **Software Automation**. Research into productivity often highlights that the simple act of "offloading" a task from the mind to a digital system can significantly reduce cognitive load and stress. By integrating features such as a dedicated **Pomodoro Timer**, this application does not just store data; it actively facilitates a focused work environment for the user. This holistic approach ensures that the application serves as a productivity coach rather than just a database of reminders, aligning with the modern shift toward "Mindful Technology" that prioritizes user well-being alongside efficiency.

Furthermore, the development process emphasized the importance of **Future-Proofing** and scalability within the Spring Boot ecosystem. Every module was designed with the possibility of horizontal scaling in mind, ensuring that the architecture could handle an increasing volume of users and data without a degradation in performance. This was achieved by adhering to **SOLID principles** and utilizing industry-standard design patterns, such as Dependency Injection and Singleton patterns, which are native to the Spring framework. Consequently, this internship project serves as a comprehensive case study in how to build a resilient web application that is prepared for the transition from a local development environment to a cloud-based production server.

Chapter 2: Project Objectives

The primary objective of this internship project was the successful design and deployment of a full-stack **Task Management System** that incorporates secure user authentication. This required a multi-disciplinary approach, starting with the establishment of a robust development environment. One of the first milestones was configuring the **Java Development Kit (JDK 17)** alongside **Maven** for dependency management, ensuring that all libraries—ranging from Spring Data JPA to the Mail Sender—were version-controlled and compatible. This foundation was critical for maintaining the scalability of the application as more features were added during subsequent sprints.

A secondary but equally vital objective was the implementation of a comprehensive **CRUD (Create, Read, Update, Delete)** engine. While simple in theory, the objective here was to ensure that these operations were performed asynchronously using the **JavaScript Fetch API**. This means the user could add or delete tasks without the page needing a full refresh, providing a "Single Page Application" (SPA) feel that is expected in modern web software. Each task needed to carry metadata, such as priority levels (High, Medium, Low) and due dates, which the system had to process and store accurately within the relational database.

The third objective centered on **User Security and Identity Management**. We aimed to move beyond basic password-based logins by integrating a **One-Time Password (OTP)** verification system. The objective was to ensure that every registration or password reset attempt was verified via the user's registered email address using the **Simple Mail Transfer Protocol (SMTP)**. This added a layer of protection against unauthorized access and brute-force attacks, aligning the project with modern cybersecurity standards for web applications.

Another core focus was the enhancement of **Data Usability through Advanced Logic**. In a real-world scenario, a user might have hundreds of tasks, making a single long list impractical. Therefore, a major objective was to implement server-side **Pagination and Filtering**. This involved writing complex SQL queries through JPA repositories to allow users to sort tasks by their urgency or search for specific titles. The objective was to reduce the cognitive load on the user by presenting only the most relevant information at any given time.

Lastly, the project sought to integrate **Productivity Features** such as the **Pomodoro Timer** and an **Agile Progress Tracker**. These were not just functional add-ons but were intended to provide a holistic "Work Management" experience. The progress tracker, visualized through a dynamic SVG ring, was designed to give users immediate visual feedback on their daily achievements. By fulfilling these diverse objectives, the project successfully transformed from a basic academic assignment into a professional-grade application ready for deployment.

Another significant objective was ensuring **Cross-Browser Compatibility and Responsive User Interface Design**. In the modern multi-device ecosystem, a web application must deliver a consistent experience across different rendering engines, such as Blink (Chrome/Edge), WebKit (Safari), and Gecko (Firefox). The objective was to utilize advanced CSS features—including media queries, CSS variables, and flexible grid layouts—to ensure the dashboard remained functional on screen sizes ranging from small smartphones to ultra-wide monitors. By achieving this, the project aimed to eliminate the barrier of device-dependency, allowing interns and professionals to manage their tasks seamlessly whether they are at their workstations or on the move.

The project also prioritized the objective of **Robust Exception Handling and System Resilience**. A professional application must be able to handle "unhappy path" scenarios—such as database connection failures, invalid user inputs, or expired session tokens—without crashing or displaying cryptic error messages to the user. The goal was to implement a **Global Exception Handling** mechanism within the Spring Boot backend using @ControllerAdvice. This ensured that the system could intercept errors and return meaningful, user-friendly JSON responses. This objective was vital for maintaining a high level of "System Trust," ensuring that users felt the application was stable and reliable even when errors occurred.

Finally, a key technical objective was **Performance Benchmarking and Query Optimization**. As the database grows, inefficient queries can lead to significant latency. The objective was to design the database schema with appropriate **Indexing** on frequently searched columns, such as the user_id and status fields. By analyzing the execution time of JPA queries, the goal was to ensure that the "Total Task Count" and "Progress Percentage" calculations remained near-instantaneous. This focus on backend efficiency reflects the project’s commitment to building a high-performance tool that does not degrade as the user’s data volume increases over months of active use.

Chapter 3: System Architecture & Tech Stack

The architecture of the **Automated Task Reminder & Tracking Application** is built upon a **Model-View-Controller (MVC)** design pattern, which is the industry standard for scalable web applications. At its core, the system is divided into three distinct layers: the Presentation Layer (Frontend), the Business Logic Layer (Backend), and the Data Persistence Layer (Database). This separation of concerns ensures that the UI can be updated or redesigned without affecting the underlying data processing logic, and vice versa. By using Spring Boot, we were able to utilize an "opinionated" configuration that handles much of the boilerplate code, allowing the developer to focus on unique business features.

On the **Backend**, we employed **Spring Boot 4.0.1**, which provides a powerful ecosystem for building RESTful APIs. The **Spring Data JPA** (Java Persistence API) was utilized to handle the communication between the Java objects and the MySQL database tables. This allowed us to perform complex database operations using repository interfaces rather than writing raw SQL, which reduces the likelihood of syntax errors and SQL injection vulnerabilities. Additionally, **Spring Security Crypto** was integrated to handle sensitive user data, specifically for hashing passwords before they are stored in the database, ensuring that even in the event of a data leak, user credentials remain secure.

The **Frontend** was developed using a combination of **HTML5, CSS3, and Vanilla JavaScript**. Rather than relying on heavy frameworks like React or Angular for this specific version, we chose a lightweight approach to ensure maximum performance and fast loading times. The **"Poppins" Google Font** was selected for its modern and clean aesthetic, while **FontAwesome** provided the iconography needed for an intuitive user experience. The CSS utilizes a modular approach, with variables for primary colors (like the Infosys-themed blue) and responsive "Flexbox" layouts that ensure the application looks great on both desktops and mobile devices.

For the **Database**, **MySQL 8.0** was chosen due to its reliability and widespread industry use. The database schema was designed to be relational, with a User table linked to a Task table via a foreign key relationship. We also implemented a dedicated OtpVerification table to manage temporary tokens and their expiry timestamps. This relational structure ensures data integrity, allowing for features like "Cascading Deletes," where deleting a user account automatically removes all associated tasks, thereby maintaining a clean and efficient database.

The communication between the frontend and backend is facilitated through **RESTful Endpoints**. When a user interacts with the UI—for example, by clicking the "Save Task" button—the JavaScript code packages the form data into a **JSON (JavaScript Object Notation)** object and sends it to the server via an HTTP POST request. The Spring Boot controller receives this request, validates the data using the Service layer, and then persists it to the database. This asynchronous communication is the backbone of the application's responsiveness, providing a fluid experience that mirrors the speed of a desktop application.

A cornerstone of the application's backend architecture is the **Dependency Injection (DI) and Inversion of Control (IoC)** container provided by the Spring Framework. By utilizing @Service, @Repository, and @RestController annotations, the application delegates the instantiation and lifecycle management of components to the Spring Container. This architectural choice significantly enhances the "Testability" and "Maintainability" of the code. For example, the AuthController does not manually create an instance of UserService; instead, it is injected via the constructor. This loose coupling allows for easier swapping of implementations and ensures that the codebase remains clean and modular as the project grows in complexity.

The **Data Persistence Layer** is optimized through the use of **Hibernate as the ORM (Object-Relational Mapping)** provider. Hibernate manages the mapping between the Java Task entity and the corresponding MySQL table, effectively abstracting the complexity of JDBC code. This layer utilizes a "Persistence Context," which acts as a first-level cache, ensuring that database transactions are efficient and ACID (Atomicity, Consistency, Isolation, Durability) compliant. By leveraging the @Transactional annotation, the application ensures that complex operations—such as updating a task while simultaneously logging an activity—are performed as a single unit of work, preventing data corruption in the event of a system failure.

On the **Security Architecture** front, the application implements a custom **Middleware Filter Chain**. Even before a request reaches the TaskController, it passes through a series of security checks that validate the user's session and credentials. This is particularly important for the **OTP Verification** flow, where the system must intercept requests to /auth/verify-otp to ensure the provided token is not only correct but also still within its five-minute validity window. This layered defense-in-depth strategy ensures that the "Business Logic" of the application remains isolated from the "Security Logic," making the system much more resilient to unauthorized access attempts.

The **Frontend Communication Layer** is built around the concept of **Asynchronous JavaScript and XML (AJAX)**, specifically using the modern fetch() API. Unlike traditional web applications that require a page reload for every interaction, this application communicates with the Spring Boot backend via JSON (JavaScript Object Notation) payloads. When a user updates a task status, the frontend dispatches a non-blocking PATCH or POST request. The backend processes the request and returns a standardized response, which the frontend then uses to update the Document Object Model (DOM) dynamically. This architecture provides a fluid, app-like experience that significantly reduces server load and bandwidth consumption.

Finally, the **Build and Dependency Management** is orchestrated via **Apache Maven**. The pom.xml file serves as the blueprint for the entire project, defining not only the library versions (such as Spring Boot 4.0.1) but also the build lifecycle and plugin configurations. We utilized the maven-compiler-plugin to ensure the project is strictly compiled under JDK 17, and the spring-boot-maven-plugin to package the application into an executable JAR file. This standardization is vital for "DevOps" integration, ensuring that the application can be consistently built and deployed across different environments—from a developer's local machine to a cloud hosting service like AWS or Azure—without "dependency hell" or version mismatches.

Chapter 4: Agile Methodology & Sprint Details

The project was executed using the **Agile-Scrum** methodology, which emphasizes iterative development, constant feedback, and high-quality deliverables. Agile was chosen because it allows for flexibility; if a particular feature (like the OTP service) faced technical hurdles, the team could adjust the sprint priorities without derailing the entire project timeline. The development was divided into three distinct **Sprints**, each lasting approximately two weeks. This structured approach ensured that we were always working on the most valuable features first, moving from the "Must-Have" foundational elements to the "Should-Have" enhancements.

Sprint 1 was dedicated to the **Foundation and User Interface**. The primary goal was to get the "Skeleton" of the application up and running. This involved the initial setup of the Spring Boot project and the creation of the main dashboard UI. During this phase, I focused on the auth.css and style.css files to establish a consistent visual language. By the end of this sprint, the application had a working landing page and a basic modal for adding tasks, although the backend connection was still in its infancy. This phase was crucial for establishing the "look and feel" of the product.

Sprint 2 shifted the focus to the **Backend API and Data Logic**. This was the most technically intensive phase, as it involved creating the TaskController, TaskService, and TaskRepository. The objective was to enable the CRUD operations that would make the UI functional. I implemented server-side logic for filtering tasks by status (Pending/Done) and priority. It was during this sprint that I also integrated **Maven** dependencies for the MySQL connector, ensuring that the application could successfully save data to a local database instance. This sprint turned a static UI into a living, data-driven application.

Sprint 3 focused on **Security, Authentication, and Refinement**. As documented in the Agile Sheet, this sprint included the implementation of the UserService and OtpService. The most challenging part of this phase was configuring the **JavaMailSender** to work with Gmail's SMTP servers, which required navigating security settings like "App Passwords." Additionally, I worked on session management to ensure that users were redirected to the login page if they tried to access the dashboard without an active session. This sprint was about hardening the application and preparing it for a "production-ready" state.

The **Agile Project Management Sheet** served as our "Source of Truth" throughout the internship. It tracked every User Story (US) and Task ID, identifying dependencies between different modules. For instance, the US-09 (OTP Verification) was dependent on the successful completion of US-08 (User Registration). By maintaining this sheet, I was able to visualize the progress, identify bottlenecks, and ensure that every requirement listed in the project scope was addressed. This disciplined approach to project management is what allowed me to stay on track even as the complexity of the code increased.

A pivotal objective of the development phase was the integration of **Asynchronous Notification Logic** to bridge the gap between static data storage and active user engagement. The goal was to ensure that the application did not remain a passive repository but instead acted as an active assistant.This objective required a deep dive into **Java's Time and Date API**, ensuring that all task deadlines were stored in a standardized UTC format to avoid timezone discrepancies between the server and the client.

Furthermore, the project prioritized the objective of **Optimizing Front-to-Back Data Transfer** through the use of standardized **Data Transfer Objects (DTOs)**. In professional software development, exposing the internal database entities directly to the web layer can lead to security vulnerabilities and unnecessary data overhead. Therefore, a major goal was to implement a layer of abstraction that filtered out sensitive fields—such as internal user IDs or password hashes—before sending JSON responses to the frontend.

Finally, a core objective was the **Alignment with Agile Lifecycle Documentation**, ensuring that every line of code was backed by a clearly defined business requirement. This meant that the development process was not driven by arbitrary feature requests but by a structured **Product Backlog**. Each objective was mapped to a specific US ID (User Story ID) in the Agile Management Sheet, such as US-04 for task priority management. This objective-driven development ensured that the internship project remained focused on delivering high-value features first. By maintaining this discipline, I was able to demonstrate how a complex software project can be managed predictably, moving from a conceptual "Minimum Viable Product" (MVP) to a fully-featured, secure, and authenticated tracking system.

Chapter 5: Initial Training & The Product Class

Before diving into the complex architecture of the Task Reminder app, the internship began with a foundational training phase aimed at mastering **Object-Oriented Programming (OOP)** in Java. The primary objective of this phase was to create a Product class that adhered to strict industry naming conventions and architectural patterns. This exercise was essential for understanding how data flows through a Java application and how to use **Collections** like ArrayList for in-memory storage. It served as a "microcosm" of the larger project, teaching the importance of encapsulation, constructors, and data structures.

The implementation involved creating a **POJO (Plain Old Java Object)** named Product with attributes such as productId, name, type, and quantity. By making these fields private and providing public getter and setter methods, I practiced the principle of **Encapsulation**. This ensures that the data cannot be modified in unintended ways by other parts of the program. I also implemented a parameterized constructor, which allowed for the quick instantiation of product objects, a pattern that I later applied when creating User and Task entities in the main application.

Following the creation of the class, the next step was to build a **Main class** that acted as the controller for the user input. Using the Scanner class, I implemented a loop that allowed users to enter details for multiple products. These objects were then stored in an **ArrayList**, providing a clear introduction to how Java handles dynamic lists of objects. This phase was critical for understanding the "logic layer" of an application, as it required validating user input and managing the lifecycle of objects within the system's memory before moving on to persistent database storage.

One of the most valuable aspects of this early task was the introduction to **Unit Testing** and code verification. After storing the products in the list, I had to write logic to iterate through the ArrayList and print the details to the console in a formatted manner. This verified that the data was being captured and retrieved correctly. Although simple, this process instilled a "test-first" mindset, emphasizing that no feature is truly complete until it has been verified through a series of test cases. This laid the groundwork for the more complex **Postman API testing** that would come later in the project.

In summary, the Product Class exercise was not merely a coding task but a lesson in **Software Craftsmanship**. It taught me how to organize code into appropriate packages (e.g., com.taskreminder.model), how to document methods properly, and how to think about data from a system perspective. These fundamentals were the building blocks for the **Automated Task Reminder & Tracking Application**. Transitions from simple ArrayList storage to complex **MySQL** databases felt natural because the core logic of handling objects and collections remained the same. This chapter serves as a bridge between academic learning and professional software development.

The training phase also served as an intensive introduction to the **Java Collections Framework**, specifically focusing on the performance trade-offs between different data structures. While the ArrayList was selected for the Product module due to its fast O(1) retrieval time and dynamic resizing capabilities, this exercise prompted a deeper analysis of how objects are stored in the JVM (Java Virtual Machine) heap memory. Understanding how an ArrayList manages its internal array capacity helped me appreciate the importance of efficient memory management in Java. This knowledge was directly applicable later in the project when handling task lists, as it influenced how I structured data retrieval to ensure that the application remained performant even as the user's "Product" or "Task" database scaled in size.

Furthermore, this phase emphasized the critical role of **Parameterized Constructors and Method Overloading** in creating flexible and reusable code. By implementing multiple ways to initialize a Product object, I learned how to handle varied data inputs while maintaining strict type safety. This exercise was a prerequisite for working with **Spring Data JPA**, where the concept of "Entities" requires a default no-args constructor for Hibernate's proxy creation, alongside parameterized ones for developer use. Mastering these nuances during the initial training meant that when I moved on to building the User and Task entities for the main application, the transition was seamless, as I already understood the architectural requirements for object instantiation and data integrity.

Chapter 6: Module Implementation & Security

The implementation phase of the **Security Module** was perhaps the most critical component of the entire application development. In a modern web environment, protecting user data goes beyond simple password storage; it requires a robust authentication flow that prevents unauthorized access while remaining user-friendly. The application utilizes a **Two-Factor Authentication (2FA)** philosophy by combining traditional password-based login with a dynamic **OTP (One-Time Password)** system. This ensures that even if a password is compromised, the account remains protected by the user's email access, which serves as a second layer of identity verification.

The core of this module is the OtpService, which was developed to handle the generation, storage, and validation of verification codes. When a user requests an OTP—either for registration or a password reset—the service generates a cryptographically secure 6-digit integer. This code is not just sent; it is persisted in the OtpVerification table with a LocalDateTime expiry timestamp set to five minutes from the moment of generation. This "time-to-live" (TTL) logic is essential for preventing replay attacks, where an attacker might attempt to use an old, intercepted code to gain access.

For the physical delivery of the OTP, the application integrates the **Spring Boot Starter Mail** dependency. This required configuring the JavaMailSender bean within the application.properties file to communicate with the Gmail SMTP server. Special attention was paid to the security settings, including the enablement of starttls and the use of an encrypted "App Password" to bypass Google's standard security blocks for less secure apps. The sendOtp method constructs a SimpleMailMessage and dispatches it asynchronously, ensuring that the user's web request is not blocked while waiting for the email server to respond.

Security on the database level was managed using the **Spring Security Crypto** library. It is a fundamental security practice never to store passwords in "Plaintext." Therefore, I implemented the BCryptPasswordEncoder to hash user passwords during the signup and reset-password processes. BCrypt is a slow-hashing algorithm that includes a "salt" to protect against rainbow table attacks. When a user attempts to log in, the system retrieves the hashed password from the database and uses the encoder.matches() method to compare it against the user's input, maintaining the highest standard of credential safety.

Finally, the **Controller Layer** acts as the gateway for these security features. The AuthController exposes specific REST endpoints like /auth/send-otp, /auth/verify-otp, and /auth/reset-password. Each endpoint is designed to return a clear string response to the frontend, indicating the success or failure of the operation. This modular approach allows the security logic to be easily updated—for example, switching from 6-digit OTPs to alphanumeric codes—without necessitating a rewrite of the frontend UI or the primary task management logic.

A significant technical hurdle addressed during the implementation of the security module was the **Token Expiry Synchronization**. In a distributed system environment, server time and client time can occasionally drift, leading to premature OTP expiration. To mitigate this, I implemented a "buffer window" within the verifyOtp logic, allowing for a 30-second grace period. This small but vital adjustment significantly improved the user experience by reducing "False Negatives" during the verification process. Furthermore, the OtpVerification entity was designed to be "Idempotent," meaning that once a code is successfully verified, its status is immediately toggled to is_verified = true. This prevents the same OTP from being reused within its 5-minute window, providing a robust defense against "Replay Attacks" where an intercepted token might be reused by an unauthorized party.

Beyond the OTP logic, the security module also incorporates **CORS (Cross-Origin Resource Sharing)** configurations to protect the REST APIs. Since the frontend might potentially be hosted on a different port or domain than the Spring Boot backend, I configured a WebMvcConfigurer bean to explicitly define allowed origins, methods (GET, POST, PUT, DELETE), and headers. This prevents "Cross-Site Request Forgery" (CSRF) and ensures that only authorized frontend clients can communicate with the AuthController. By hardening the network layer in addition to the application layer, the system achieves a "Defense-in-Depth" posture that is standard for professional Infosys-grade software solutions.

Chapter 7: Backend Logic & Data Persistence

The backend logic of the application serves as the "brain" of the system, coordinating the flow of data between the user's web browser and the persistent storage of the MySQL database. This layer was built using the **Service-Repository Pattern**, which promotes a clean separation of concerns. The TaskService contains the core business logic, such as determining which tasks are overdue or calculating the percentage of completed tasks for the dashboard. By isolating this logic from the TaskController, the code becomes significantly easier to test and maintain throughout its lifecycle.

Data persistence is handled through **Spring Data JPA**, which maps Java objects to database tables. For the Task Management module, I designed a TaskRepository interface that extends JpaRepository. This provided out-of-the-box support for basic CRUD operations while allowing me to define custom query methods. For instance, I implemented a method to find tasks by their status and priority, which is used by the frontend filters. The use of @Entity and @Table annotations on the Task class allows Hibernate to automatically generate the necessary SQL commands to create and update the database schema.

One of the more complex logic modules implemented was **Server-Side Pagination**. As a task list grows into the hundreds or thousands, loading all records at once would drastically slow down the application and consume excessive memory. To solve this, I utilized the Pageable and Page interfaces provided by Spring Data. This allows the backend to fetch only a "slice" of data (e.g., 10 tasks at a time) based on the user's current page. This optimization is critical for building a "production-ready" application that can scale to handle large volumes of user-generated content.

The **Service Layer** also manages the "Business Rules" of the application. For example, before a task is saved, the service ensures that the due date is not in the past and that the title is not empty. This validation layer acts as a second line of defense against corrupted data that might bypass frontend validation. During the development of the UserService, I also implemented logic to check for existing email addresses to prevent duplicate account creation. If a user tries to sign up with an email already in the system, the service throws a custom exception that the controller translates into a user-friendly error message.

Furthermore, the backend logic includes **Database Relationship Management**. The application is designed to support multi-tenancy, meaning multiple users can use the app without seeing each other's tasks. This was achieved by linking each Task entry to a specific User ID. Every time a task is created or retrieved, the system filters the results to ensure that only the tasks belonging to the currently logged-in user are returned. This relational integrity is the cornerstone of the application's privacy and data organization.

To enhance the efficiency of the data persistence layer, I implemented **Custom Query Projections** within the TaskRepository. Often, the dashboard only needs a count of tasks or a list of titles rather than the full task object with heavy description text. By using Spring Data JPA Projections, I was able to fetch only the necessary columns from the MySQL database, significantly reducing the "Memory Footprint" and the "Result Set" size transferred over the network. This optimization ensures that even as the task_db grows to contain thousands of records, the summary statistics on the main dashboard remain instantaneous, reflecting a commitment to high-performance backend engineering.

Additionally, the backend logic was designed to handle **Database Transactional Integrity** through the @Transactional annotation. This is particularly crucial during the "Signup-Verification" flow. When a user is successfully verified via OTP and their account is created, these two operations (updating the OTP status and saving the new User entity) must succeed or fail as a single atomic unit. If the database crashes mid-way, the transaction is "Rolled Back" to prevent an "Orphaned OTP" or a partially created user profile. This adherence to **ACID properties** (Atomicity, Consistency, Isolation, Durability) ensures that the application's data remains consistent and reliable, which is a non-negotiable requirement for any enterprise-level tracking application.

Chapter 8: User Interface & Experience Design

The **User Interface (UI)** of the Task Reminder application was designed with a "User-First" philosophy, focusing on clarity, speed, and modern aesthetics. To achieve a professional look consistent with the Infosys Springboard brand, I chose a clean, minimalist design using the **Poppins** typeface. The layout is built on a **Grid and Flexbox** system, which allows the dashboard to adapt gracefully to different screen sizes. This responsive design ensures that an intern can manage their tasks on a high-resolution desktop monitor just as easily as they could on a mobile device while on the go.

The **Dashboard Component** serves as the central hub of the application. It features a "Stats Grid" at the top, providing an immediate overview of total tasks, pending items, and completed work. To add a layer of engagement, I implemented a **Dynamic Progress Ring** using SVG and JavaScript. As a user checks off tasks, the ring fills in real-time, providing a visual "dopamine hit" that encourages productivity. This type of Interactive UX (User Experience) transforms the application from a boring utility into a motivating workspace.

To handle the complexity of task management, I developed a multi-view system. Users can toggle between a **Table View** for detailed data analysis and a **Card View** for a more visual, Kanban-style experience. The CSS for these views utilizes "Glassmorphism" effects—subtle blurs and transparency—to create a sense of depth and modernity. I also incorporated a **Search and Filter Bar** that updates the view as the user types. This "Live Filtering" is achieved by attaching event listeners to the input fields, hich trigger asynchronous requests to the backend, resulting in a highly responsive feel.

The **Authentication UI** was developed in a separate module (auth.html and auth.css) to maintain a clean security boundary. The login and registration cards are centered on the screen with soft shadows and rounded corners to reduce visual clutter. I paid special attention to the **Interactive Feedback**; for example, when an OTP is being sent, the "Send OTP" button changes state to indicate progress. Error messages are displayed in red, while success messages appear in a soft green toast notification, ensuring that the user is never left wondering about the status of their request.

Finally, the UI incorporates **Accessibility and Animation**. Using CSS @keyframes, I added subtle "Fade-In" and "Slide-Up" animations to elements as the page loads. These aren't just for decoration; they help guide the user's eye to important areas like the "Add Task" button. I also ensured that the color contrast ratios meet WCAG standards, making the application usable for individuals with visual impairments. This commitment to a high-quality UI/UX demonstrates that the application is a professional product, bridging the gap between a student project and a real-world software solution.

The UI was further refined through the implementation of **Optimistic UI Updates**. In a standard web flow, a user clicks "Complete Task" and waits for a server response before the UI changes. To provide a "Snappier" experience, I wrote JavaScript logic that updates the task status in the DOM immediately upon the click, while simultaneously sending the fetch request in the background. If the server returns an error, the UI "Rolls Back" to its previous state and alerts the user. This advanced technique mimics the behavior of high-end mobile apps and significantly increases the "Perceived Speed" of the application, making it feel more like a native desktop tool than a standard webpage.

Finally, I integrated a **Centralized Toast Notification System** to handle user feedback across the entire application. Instead of using intrusive alert() boxes, I designed a non-blocking notification component that slides in from the top-right corner. This system is "Context-Aware," meaning it uses different color schemes for success (Green), warnings (Yellow), and errors (Red). For instance, when a user successfully resets their password, a green toast confirms the action before redirecting them to the login page. This level of "Visual Polish" ensures that the user is always informed of the system's state without disrupting their workflow, fulfilling the final objective of creating a truly professional and memorable user journey during the Infosys Springboard internship.

Appendix: Technical Specifications

A.1 Database Schema Definition (SQL)

The application relies on a relational MySQL structure. The following scripts define the core tables used for user management, task tracking, and security verification.

SQL

```
/* User Table: Stores encrypted credentials */

CREATE TABLE users (

    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    email VARCHAR(100) NOT NULL UNIQUE,

    password VARCHAR(255) NOT NULL,

    created_at TIMESTAMP DEFAULT CURRENT_VALUE

);


/* Task Table: Stores task metadata and user relationship */

CREATE TABLE tasks (

    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    user_id BIGINT,

    title VARCHAR(255) NOT NULL,

    description TEXT,

    due_date DATE,

    priority ENUM('LOW', 'MEDIUM', 'HIGH') DEFAULT 'MEDIUM',

    status ENUM('PENDING', 'DONE') DEFAULT 'PENDING',

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE

);


/* OTP Table: Temporary storage for 2FA codes */
```

```
CREATE TABLE otp_verifications (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(100) NOT NULL,
    otp_code VARCHAR(6) NOT NULL,
    expiry_time DATETIME NOT NULL,
    is_verified BOOLEAN DEFAULT FALSE
);
```

A.2 Core Backend Structures (Java)

The User Entity

The User entity utilizes Lombok annotations for clean code and JPA for database mapping.

Java

```
@Entity
@Table(name = "users")
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String password;
}
```

The OTP Service Logic

This snippet demonstrates the logic used to send the email and save the verification state.

Java

```
public void sendOtp(String email) {
    String otp = String.valueOf(new Random().nextInt(900000) + 100000);

    OtpVerification v = repo.findByEmail(email).orElse(new OtpVerification());
    v.setEmail(email);
    v.setOtp(otp);
    v.setExpiryTime(LocalDateTime.now().plusMinutes(5));
    v.setVerified(false);
    repo.save(v);

    SimpleMailMessage message = new SimpleMailMessage();
    message.setTo(email);
    message.setSubject("Task Reminder - OTP Verification");
    message.setText("Your OTP is: " + otp + ". It expires in 5 minutes.");
    mailSender.send(message);
}
```

A.3 Configuration & Dependencies (POM.xml)

The pom.xml file is the backbone of the build process, managing the Spring Boot starter kits and the MySQL connector.

XML

```
<dependencies>
    <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-crypto</artifactId>
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

A.4 Application Properties

This file configures the connection to the Gmail SMTP server and the local MySQL instance.

Properties

```
# Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/task_db

spring.datasource.username=root

spring.datasource.password=your_password

spring.jpa.hibernate.ddl-auto=update


# SMTP Mail Configuration

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=your_email@gmail.com

spring.mail.password=your_app_specific_password

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true
```

A.5 UI Component Structure (HTML/JS)

The frontend utilizes a modular approach for the task dashboard.[1]

JavaScript

```
/* --- Frontend Logic: Fetch Tasks with Pagination --- */

async function loadTasks(page = 0, size = 10) {

  const res = await fetch(`${API_BASE}/tasks?page=${page}&size=${size}`);

  const data = await res.json();


  const container = document.getElementById("tableView");

  container.innerHTML = data.content.map(task => `

    <div class="task-row">
```

```
            <span>${task.title}</span>
            <span class="badge ${task.priority.toLowerCase()}">${task.priority}</span>
            <button onclick="toggleStatus(${task.id})">${task.status}</button>
        </div>
    `).join('');
}
```

Final Documentation Summary

This report, spanning from the **Introduction** to this **Appendix**, represents a complete professional development cycle. It covers the initial planning through **Agile**, the setup of **Object-Oriented** foundations, the implementation of **RESTful Services**, and the final **UI/UX design**. By documenting each technical hurdle and design choice, this project stands as a testament to the skills acquired during the Infosys Springboard internship.