CS/SE/CE 3354.004 Software Engineering
Final Project Deliverable 2

Aurora - Focus App

Group Members:
Oghenetejiri Etaghene, Adrian Hautea, Mohammad Khan, Sravya Kotamraju, Vrishti Misra, Nihal Paul, Ece Yobas

Table of Contents

I. About

Aurora: Focus App is a productivity tool designed to help students and professionals manage their time effectively and maintain consistent focus. The app combines the proven Pomodoro technique with features that encourage motivation and accountability, such as streak tracking, background customization, and leaderboards. Users can personalize their study sessions by setting focus and break intervals, tracking their progress, and competing with peers to stay engaged.

Assumptions: Aurora assumes standard global app store publishing rules and general data privacy requirements (e.g., GDPR-like encryption and parental consent for users under 13).

The implementation includes several core modules:

- **Focus Timer Module**
  Handles session timing, notifications, and transitions between focus and break periods. This module forms the foundation of Aurora, allowing users to start, pause, and resume focus sessions with customizable durations. It ensures accurate timing, automatic transitions, and reliable performance even if the app is paused or closed.
- **Task Management and Data Handling Module**
  Enables users to create, edit, categorize, and prioritize tasks that align with focus sessions. Completed tasks are automatically logged into session history and stored securely for progress tracking. The Data Handling component manages the storage and retrieval of user data, including session logs and task lists, through local persistence and optional cloud synchronization for multi-device access.
- **Streak Tracking and Leaderboard/Progress Tracking Module**
  Tracks consecutive focus sessions, computes productivity streaks, and displays motivational progress statistics. This module also powers the leaderboard feature, allowing users to compare productivity scores with peers. Data is securely synchronized with the cloud to maintain accurate, up-to-date rankings across all devices.

Aurora is being developed by a team of students who understand the challenges of balancing coursework and productivity. The goal is to create a user-friendly, visually appealing platform that promotes deep work and time management habits. With intuitive design and practical features, Aurora aims to make studying and working more structured, rewarding, and sustainable.

Github Link

II. Delegation of Tasks

- ➔ Oghenetejiri Etaghene
  - ◆ Architecture Design
- ➔ Adrian Hautea
  - ◆ Project Scope, Cost & Pricing Estimation
- ➔ Mohammad Khan
  - ◆ Sequence Diagram (Focus Session), Sequence Diagram (Manage Tasks), Conclusion
- ➔ Sravya Kotamraju
  - ◆ README, Class Diagram, Project Scheduling
- ➔ Vrishti Misra
  - ◆ GitHub Creation, About, Feedback Response, Use-Case Diagrams, Software Process Model, Final Implementation of Project
- ➔ Nihal Paul
  - ◆ Functional and Non-Functional Requirements, Test Plan
- ➔ Ece Yobas
  - ◆ Sequence Diagram (Tracking Progress), Objective (Slide), Comparison

III. Feedback Response

The feedback suggested elaborating on the proposed implementation, specifically clarifying what the “task management” and “data handling” modules represent in Aurora. In response, we expanded our **About** section to include a concise explanation of each core module. The **Task Management Module** now describes how users can create, organize, and track tasks within their focus sessions, while the **Data Handling Module** explains how user progress, session logs, and leaderboard data are stored and managed. This addition provides a clearer understanding of how these components function within the app and how they contribute to the overall user experience.

IV. Software Process Model

Aurora will follow the **Incremental Process Model**, implemented through **Agile development practices** such as weekly sprints, GitHub commits, and iterative feedback cycles. This approach will divide the project into functional increments that will be developed, tested, and integrated one at a time. Each increment of Aurora will deliver a usable component that can be evaluated and refined before proceeding to the next stage. This method will ensure that the team always maintains a working version of the application and can incorporate continuous feedback to improve performance, usability, and design consistency.

Structure of Development

Increment 1: Focus Timer Module

The first increment will establish Aurora's foundation. It will include the Pomodoro-style timer with customizable focus and break intervals, automatic transitions between sessions, notifications for session start and end, and optional long breaks after multiple cycles. This increment will also define the core user interface and ensure the timer logic remains stable across sessions. Testing will verify that the timer operates accurately even when the application is paused or closed, setting the groundwork for later modules.

Increment 2: Task Management and Data Handling Module

The second increment will introduce productivity features. Users will be able to create, edit, delete, categorize, and prioritize tasks and link them to specific focus sessions. A local storage system will be implemented to retain task lists and session data even without internet access, while optional cloud synchronization will support multi-device use. The module will be tested to confirm data persistence, synchronization accuracy, and task-to-session linkage. It will also refine the user interface to display tasks dynamically with real-time updates as sessions are completed.

Increment 3: Streak Tracking and Leaderboard Module

The final increment will focus on motivation and engagement. It will implement streak tracking to record consecutive focus sessions, display progress statistics, and generate motivational messages when streak milestones are reached. The leaderboard feature will allow users to compare productivity scores with peers, supported by secure synchronization with the cloud-based leaderboard server. Testing will validate the accuracy of streak computation and verify that leaderboard data updates correctly across users.

Suitability of the Incremental Model for Aurora

- Continuous Delivery: Each module will produce a usable version of the application that can be demonstrated and reviewed by the team before integration.
- User Feedback: Testing after each increment will provide insight into timing accuracy, data persistence, and interface clarity, guiding improvements in subsequent stages.
- Parallel Teamwork: The structure will support collaboration since different team members can work simultaneously on UI design, logic development, and data management.
- Risk Reduction: Problems identified in one increment will be isolated and resolved before they affect later modules.
- Quality Assurance: Validation and verification will occur after each release, improving system reliability and reducing the likelihood of major integration issues.

Using the Incremental Process Model will allow Aurora to **evolve from a simple timer into a fully featured productivity application.** Each increment will add measurable functionality, maintain consistent design, and align with user needs for focus, organization, and progress tracking. The model's flexibility and structured milestones will make it ideal for the team's development timeline, available resources, and academic project environment.

V. Software Requirements

A. Functional

1. **Session Control**
   The system **shall** allow the user to start, pause, resume, and stop focus and break sessions, automatically transitioning between them using user-defined durations.
2. **Session Persistence**
   The system **shall** save the current timer state when the app is closed and restore that state when the app is reopened.
3. **Task Management**
   The system **shall** allow users to create, edit, categorize, prioritize, delete tasks, and link tasks to focus sessions.
4. **Progress Tracking**
   The system **shall** log each completed session, compute streak counts, display total focus minutes, and show the number of completed tasks.
5. **Notifications**
   The system **shall** send notifications at the start and end of every focus and break session.
6. **Leaderboard**
   The system **shall** maintain a leaderboard ranking users by total focus time and streak count, and allow users to opt out of leaderboard participation.

B. Non-Functional

1. **Product Requirements** – define how *Aurora* should perform and behave to ensure reliability, security, and efficiency during use.
   a) **Dependability Requirements** – the app **must** save timer data locally, so no session progress is lost if the app is closed.
   The app **must** save task data locally, so productivity data is retained even without network access.

   b) **Security Requirements** – user data **must** be stored securely on the device and **must** be transmitted only through encrypted HTTPS connections.
   c) **Efficiency Requirements** – the app **shall** provide quick response times

and minimal lag while using limited memory and battery resources on supported devices.

(1) **Performance Requirements** – the app **must** load the main dashboard within 3 seconds under normal network conditions.

(2) **Space Requirements** – the app **must** occupy no more than 200 MB of device storage, and cached data **must** remain below 350 MB.

d) **Usability Requirements** – the interface **shall** be simple and easy to understand for first-time users without requiring instructions.

2. **Organizational Requirements** - outline the internal standards and processes the development team will follow while building and maintaining *Aurora*.

   a) **Environmental Requirements** – the app **shall** operate correctly on all supported devices without requiring special hardware or customized settings.

   b) **Operational Requirements** – the team **shall** follow an Agile workflow with weekly progress updates and GitHub commits to document development.

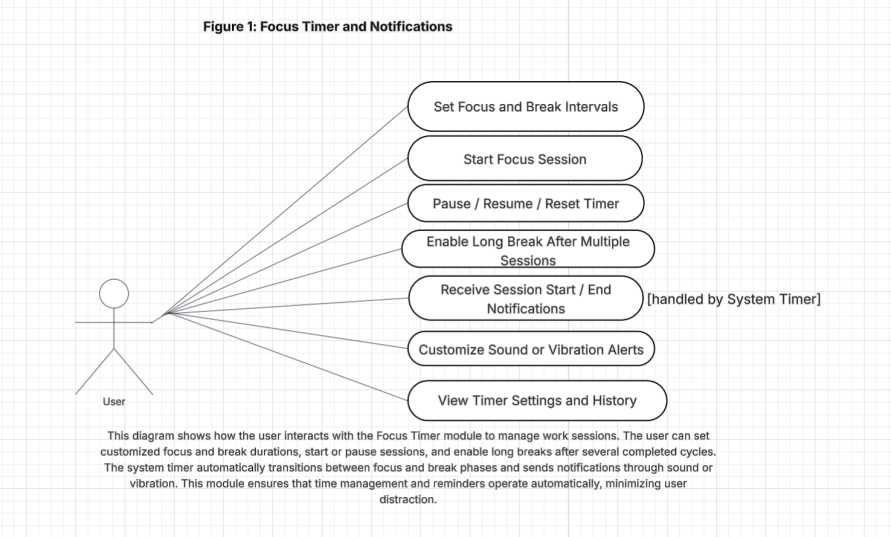
   c) **Development Requirements** – all UML diagrams **shall** be created using an approved tool as required by the professor.

3. **External Requirements** - specify how *Aurora* will comply with privacy standards, ethical design principles, and platform publishing policies to protect user data, promote healthy study habits, and maintain eligibility for app store distribution.
   a) a) **Legislative Requirements** – the app **must** comply with all applicable app-store privacy rules and all required data-handling regulations for distribution.
   b) (1) **Accounting Requirements** – if in-app purchases are added, all transactions **must** use store-approved payment systems and **must** be logged for at least one year.

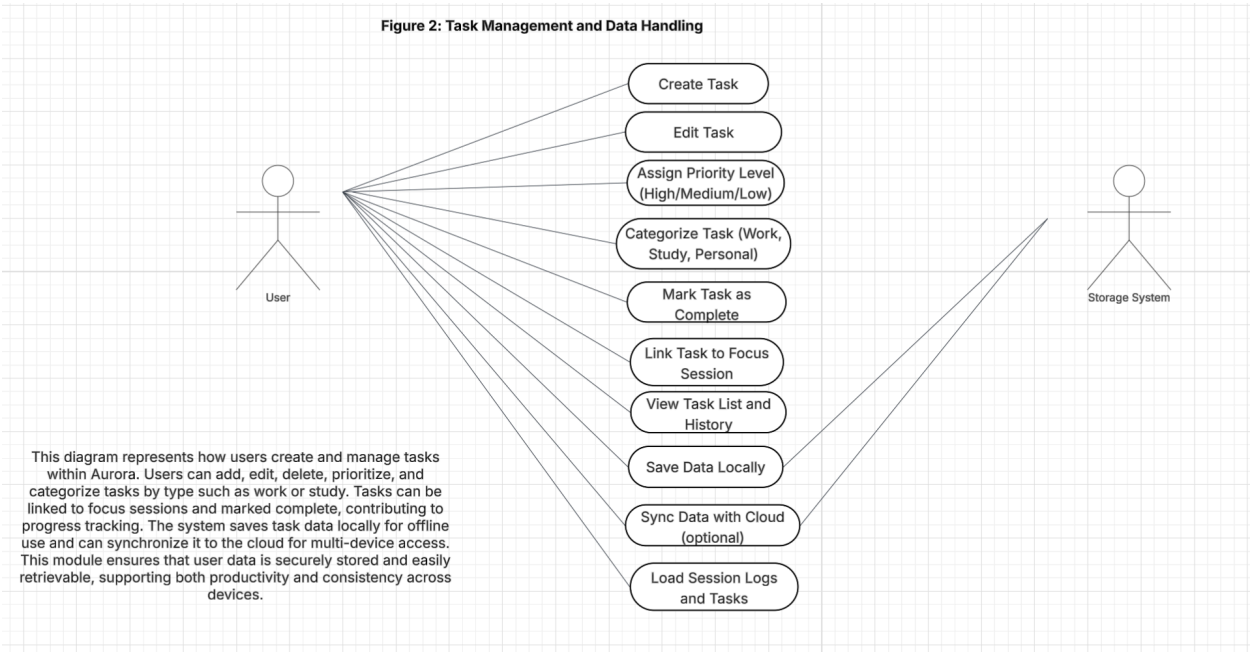
      (2) **Safety / Security Requirements** – the app **must** avoid collecting unnecessary personal data and **must** restrict accounts for users under age 13 until parental consent is verified.
   c) b) **Ethical Requirements** – the app **shall** promote healthy study habits by preventing excessively long continuous focus sessions and **shall** remind users to take breaks.

d) c) **Regulatory Requirements** – the app **must** comply with all platform publishing policies and privacy best practices required for approval on supported app stores

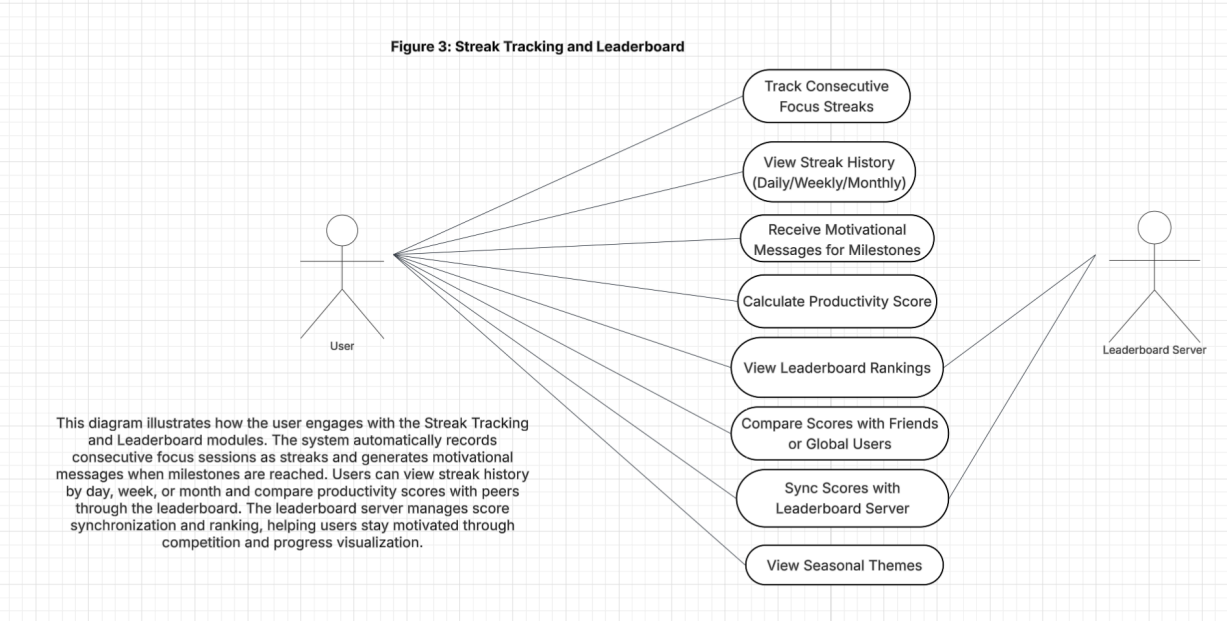
VI. Use Case Diagram



Use Case	**Aurora – Focus Timer and Notifications**
Actors	User (primary actor), System Timer (internal)
Description	The user sets focus and break intervals, starts or pauses sessions, and enables long breaks after several cycles. The System Timer automatically transitions between focus and break phases and sends notifications through sound or vibration.
Data	Focus and break durations, timer state, notification preferences
Stimulus	User starts, pauses, resets, or modifies a focus session
Response	System begins countdown, switches between phases, and issues alerts to the user
Comments	Timer operates autonomously and preserves session state if the app is closed.



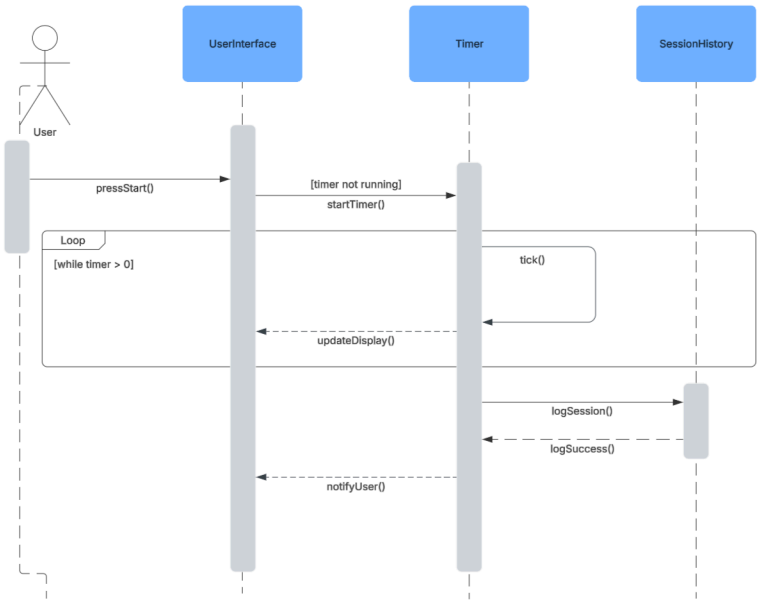
Use Case	**<u>Aurora – Manage Tasks and Store User Data</u>**
Actors	User (primary actor), Storage System (external actor)
Description	The user creates, edits, prioritizes, and categorizes tasks, marking them complete and linking them to focus sessions. Task data are saved locally for offline access and can be synchronized with the cloud for multi-device use.
Data	Task lists, task categories, completion status, local/cloud storage files
Stimulus	User adds or updates tasks or requests data synchronization
Response	System stores the changes locally and, when requested, syncs data to the cloud via the Storage System
Comments	Ensures secure data persistence and retrieval to support consistent task tracking across devices.



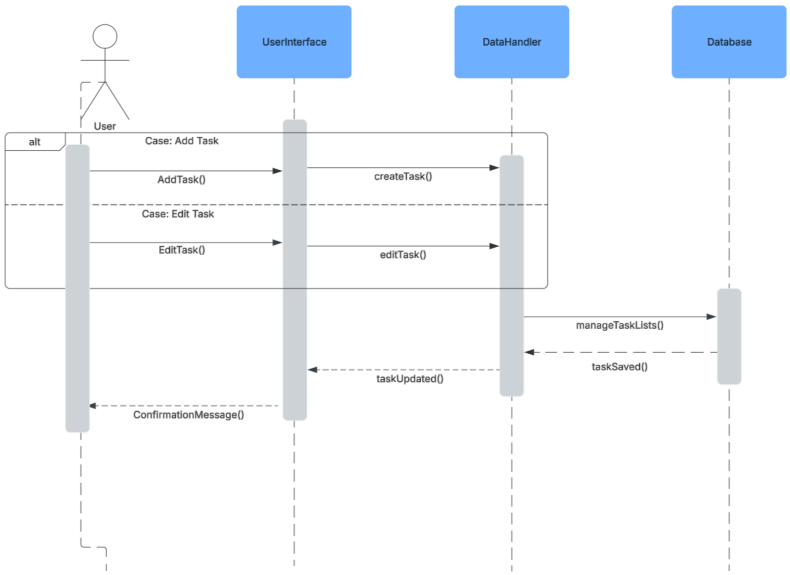
Use Case	**<u>Aurora – Track Progress and Leaderboard Ranking</u>**
Actors	User (primary actor), Leaderboard Server (external actor)
Description	The system logs consecutive focus sessions as streaks and generates motivational messages for milestones. Users can view streak history, calculate productivity scores, and compare rankings with peers through the leaderboard. The Leaderboard Server handles synchronization and ranking updates.
Data	Streak counts, focus session logs, leaderboard scores, user rankings
Stimulus	User completes a session or requests leaderboard update
Response	System updates streak records and productivity stats, then sends/receives ranking data from the Leaderboard Server
Comments	Module motivates users through progress tracking and peer comparison; data sync occurs periodically or on demand.

VII. Sequence Diagrams

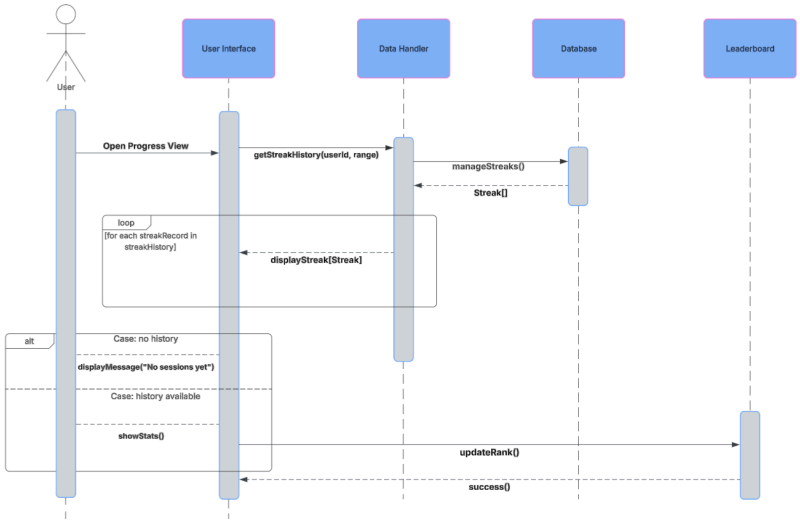
Focus Timer



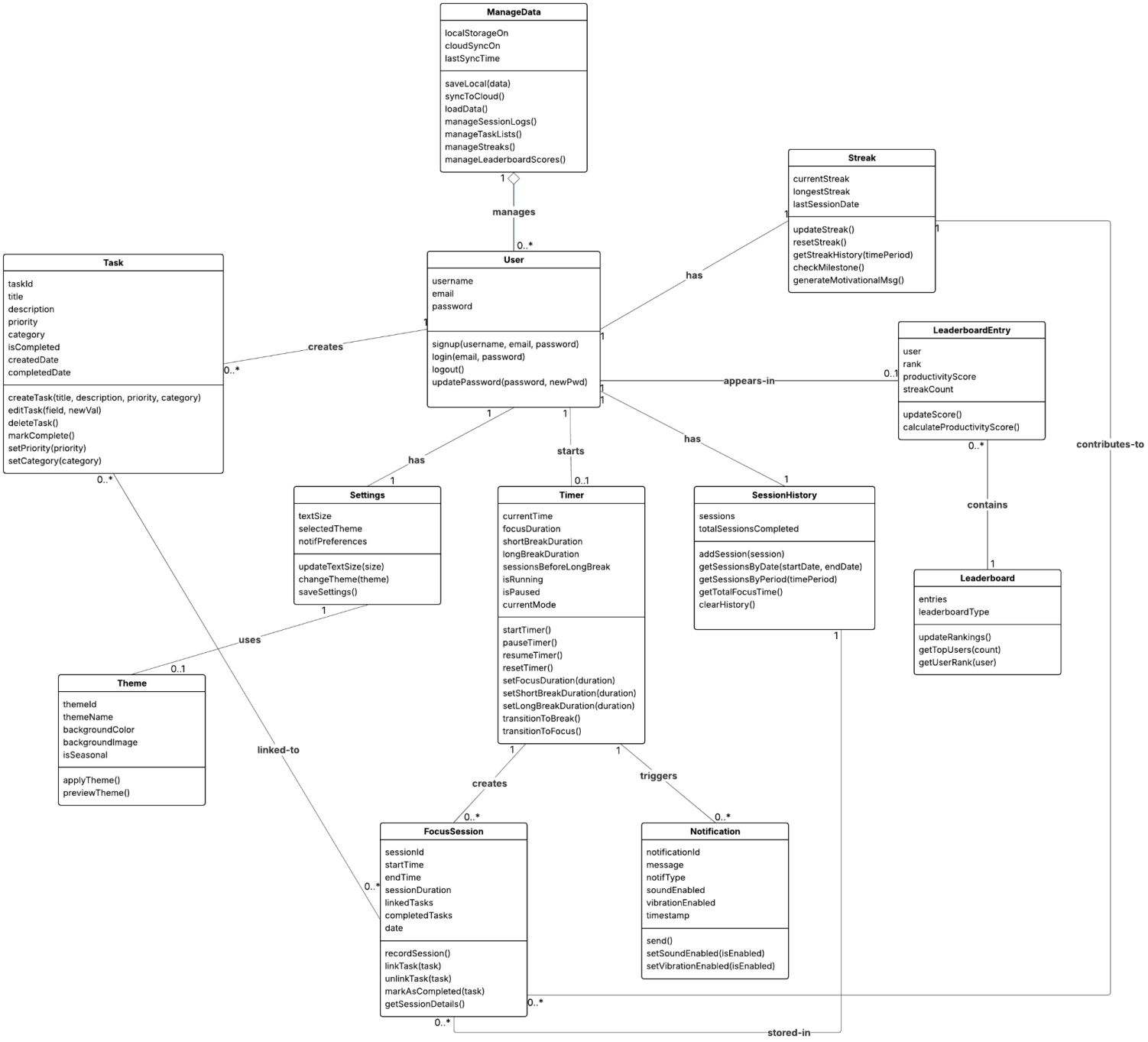
Task Management and Data Handling



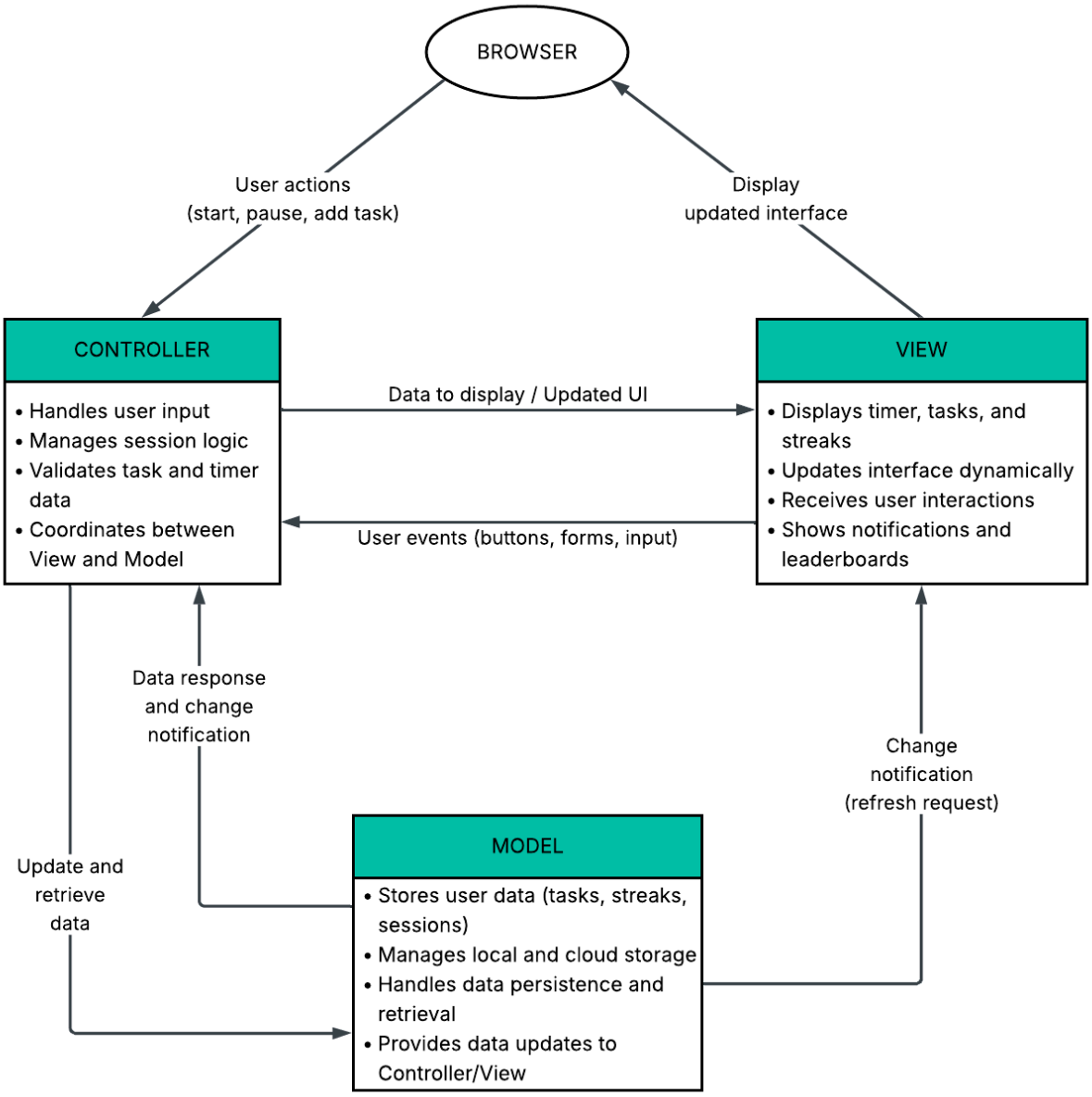
Streak Tracking and Leaderboard/Progress Tracking Module



VIII. Class Diagram

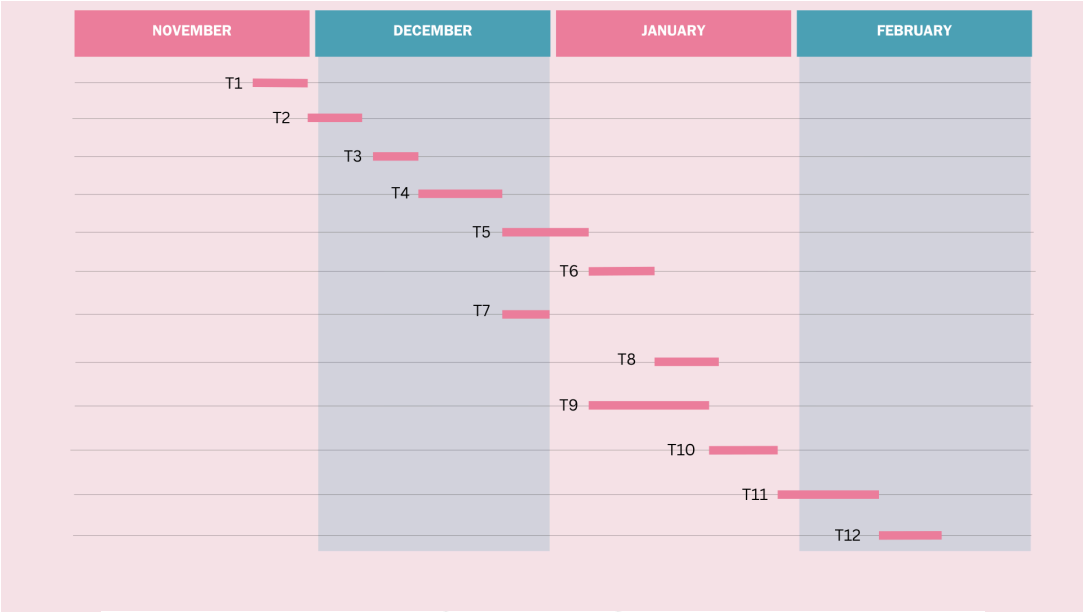


IX. Architectural Design



X. Project Scheduling, Cost, Effort, and Pricing Estimation

A. Project Scheduling



Task	Effort (person-days)	Duration (days)	Dependencies	Start date
T1: define scope and requirements	12	4		11/24/25
T2: create system architecture and UML diagrams	15	5	T1: define project scope	11/28/25
T3: set up development environment	6	3	T2: system architecture	12/5/25
T4: implement timer functionality	32	8	T3: development environment	12/10/25
T5: develop task management system	32	8	T4: timer	12/22/25
T6: build session history tracking	20	5	T5: task management	1/5/26
T7: create streak tracker	20	5	T4: timer	12/22/25
T8: implement leaderboard	20	5	T6: session history	1/12/26
T9: design UI	36	9	T5: task management	1/5/26
T10: integrate frontend and backend	42	7	T4: timer, T9: UI	1/16/26
T11: testing and bug fixes	48	12	T10: integration	1/27/26
T12: documentation	36	6	T11: testing	2/12/26

The project schedule begins on November 24, 2025, which is the point at which our team transitioned from planning to full implementation after completing the first deliverable. The planned end date is February 12, 2026, giving the team enough time to finish all three increments, complete integration, perform testing, and finalize documentation before the academic deadline. Weekends are not counted in the schedule because the team only works on weekdays due to class and personal commitments. The project assumes 5 working hours per week per team member, and this assumption is the basis for all task durations shown in the schedule. These hours match the effort estimation that was calculated using the function point method, and every task in the timeline is sized according to this weekly workload.

B. Cost, Effort, and Pricing Estimation

1. For the cost, effort, and pricing estimation, we used the function point method.

a)

Function Category	Description	Count	Complexity	Weight	Calculation	Total
User Inputs (EI)	Updates Internal Data	16	Simple	3	16x3	48
User Outputs (EO)	System Outputs	12	Simple	4	12x4	48
User Queries (EQ)	Read-Only Requests	12	Complex	6	12x6	72
Data Files (ILF)	Internal Logical Files	5	Moderate	10	5x10	50
External Interfaces (EIF)	External Data Sources	5	Complex	10	5x10	50
GFP	-	-	-	-	Sum	268

2. For processing complexity and adjustment, we first must determine which factors we want to place higher priority on. We decided that reliable backup and user-friendliness needed the highest priority due to the nature of our app, which led for our total processing complexity to be 46. Using standard values for calculating PCA, the PCA came out to be 1.11. Using the PCA and GFP we've attained, we concluded that our final function points would come out to be 298 FP.
    a) PC = (12 x 3) + (2 x 5) = 46
    b) PCA = 0.65 + 0.01(46) = 1.11
    c) FP = 268 x 1.11 = 298 FP
3. Assuming a productivity rate of 30FP per person-week, we've determined that effort comes out to 9.93 person-weeks (or 397 hours)
    a) Effort(person-weeks) = 298/30 = 9.93 person weeks
    b) Effort(hours) = 9.93 pw x 40 hrs/pw = 397

4. Factoring in personnel and labor, hardware, and software, our final project cost came out to be $6674 with a recommended project cost of $10,011.

C. Hardware Cost

1. For the hardware cost, we chose to use Amazon S3 for data storage. The Amazon S3 standard plan charges $0.023 per GB for the first 50TB utilized in the month, which is what we assume would be a good estimate of how much storage we'd need to host our users [1].

D. Software Cost

1. For the software cost, we would need rights to publish on major app stores like Google Play and the App Store. For Google Play, they charge a flat $25 fee to publish an app, while the App Store charges a $99 dollar annual fee for publishing [2]. There isn't a set time we want to be hosting this app for so we are treating the $99 annual fee as a flat rate due to the time constraint.

E. Personnel Cost

1. For the personnel cost, we are assuming a team of 6 students working around 5 hours a week for 4 months. We are also assuming a rate of $15/hr for development, which comes out to a total of $7200 for the development.

XI. Test Plan

A. The unit selected for testing is the Task Management unit, which is responsible for creating, deleting, and updating tasks in the application. In our implementation, the logic is handled by the TaskList React component, which receives a list of tasks and a callback functions for deleting tasks, toggling completion, and changing the priority of tasks.

B. Testing Approach

We used a black box unit testing approach. Instead of testing the internal implementation of the component, we tested its observable behavior, such as what renders and which callbacks are triggered when the user interacts with the interface. The tests were written in TypeScript, utilizing Jest as the test runner and the React Testing Library to render the component and simulate user interactions. The goal of the tests was to verify that the task management unit can correctly display the tasks it receives as input, calls the delete handler when the user deletes a task, and calls the priority change handler when the user updates a task's priority.

C. Test Cases

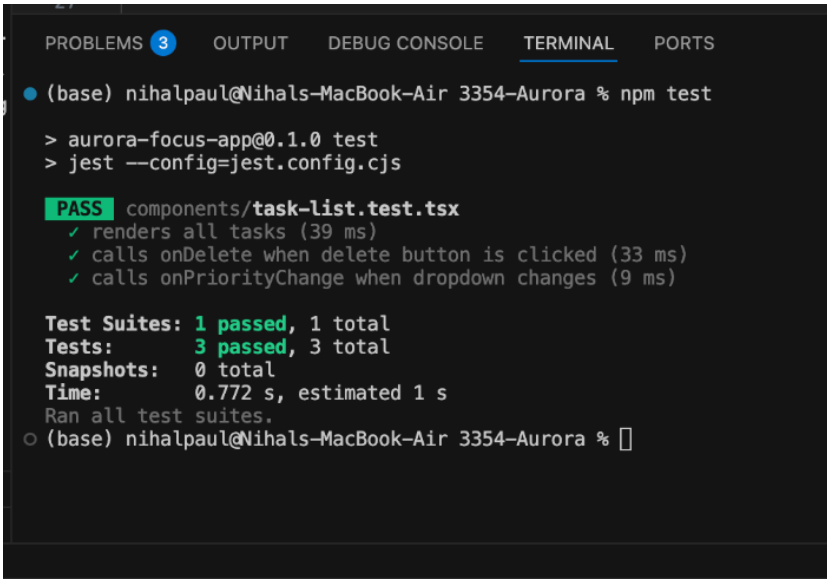
1. Render Tasks Test

a. Purpose: Confirm that all the tasks passed to the component are displayed.
b. Input: Two sample tasks with the titles of Write Report and Study.
c. Expected Result: Both task titles appear in the rendered output.

2. Delete Task Test
   a. Purpose: Verify that deleting a task triggers the callback for deleting.
   b. Input: Two sample tasks with the titles of Write Report and Study, with a mocked onDelete function. The test simulated a click on the delete button for the second task.
   c. Expected Result: onDelete is called exactly once with the ID 2.
3. Change Priority Test
   a. Purpose: Verify that changing the priority of a task triggers the priority change callback.
   b. Input: Two sample tasks with the titles of Write Report and Study, with a mocked onPriorityChange function. The test simulates changing the final task's priority from high to low using the dropdown.
   c. Expected Result: onPriorityChange is called once with the arguments 1 and low.

D. Automated Testing Results

The tests were executed using the npm test command, which runs Jest with the configurations of the project. All three unit tests for the Task Management unit passed successfully.



XII. Comparison with Similar Designs

Aurora is a productivity application that helps users maintain focus, manage tasks efficiently, and track progress through a Pomodoro-based system with streak tracking, statistics, and leaderboard features. The app's versatility allows for comparison with Pomodoro Apps, Task Management Apps, and Hybrid Focus Apps, just like Aurora. The comparisons here will only include free features since Aurora is a free app.

Traditional Pomodoro apps like Focus Timer and Flow prioritize simplicity, offering basic start/stop timers, predefined intervals, and minimal tracking [4], [5]. Although Focus Timer provides ambient sounds, none of these apps have free Spotify integration for music like Aurora. While Focus Timer and Flow add up the minutes focused and show the total statistics, Aurora records consecutive focus sessions as streaks and includes statistics like current streak and best streak. Similar to Flow's daily motivational messages, Aurora can send motivational messages when streak milestones are reached. Aurora also implements an optional setting where users can add their friends and see them on the leaderboard.

Task management focused apps like Todoist and Structured offer strong organizational tools, with task creation, editing, and priority tracking just as Aurora does [5], [6]. They also all provide ways to categorize or group tasks, giving users a clear structure for managing their responsibilities. The differences between Todoist and Structured and Aurora come from the fact that in Aurora, tasks can also be directly tied to focus sessions, contributing to streaks, productivity statistics, and milestone achievements. While Todoist also utilizes streaks, it tracks task completions, while Aurora tracks productivity sessions, sends motivational milestone messages, and shows streaks on a leaderboard if the user decides to do so.

Hybrid focus apps such as Focus To-Do combine task management with focus oriented features in a single system, similar to Aurora's overall design [7]. Like Aurora, Focus To-Do provides session notifications and keeps track of completed work. Focus To-Do offers white noise, whereas Aurora has Spotify integration, and the user can categorize tasks into groups in both apps. On top of these, Aurora also offers small accessibility options like adjustable text size and high-contrast themes, giving users more choices over how they want to customize their app.

XIII. Conclusion

Ultimately, the planning and implementation of our project proceeded smoothly and without significant obstacles. We did not have to make any changes to our proposed application structure and we were able to deliver a product that met all of our requirements. Since our implementation was only a prototype, we did not include every feature we originally envisioned, but we still incorporated several enhancements that we felt would elevate the user experience, including embedded Spotify study playlists, a leaderboard, a streak counter, night mode, and a task list with priority marking. These additions made the prototype feel more complete and gave us a clearer idea of how the full version of the application could function.

As a team, we are satisfied with the project that we were able to plan and implement. It is hard to make an evaluation without actual delivery of the product and user feedback, but we are proud of meeting the requirements we set out and the user experience we created.

XIV. Final Implementation

https://aurora-focus-app.vercel.app/

Code & instructions to run on local terminal found on GitHub.

Framework: Next.js 16
Language: TypeScript
Styling: Tailwind CSS
UI Components: Radix UI + shadcn/ui
Icons: Lucide React
Charts: Recharts
State Management: React Hooks

XV. References

[1] Amazon Web Services, Inc., *"S3 Pricing."* Accessed: Nov. 22, 2025. [Online]. Available: https://aws.amazon.com/s3/pricing/?nc=sn&loc=4&refid=8fb2f435-ab10-499e-a971-e5ab7d41a7a0

[2] B. Patel, *"Cost to Put an App on App Store: 2025 Guide & Steps,"* SpaceO Technologies, Nov. 15, 2025. Accessed: Nov. 22, 2025. [Online]. Available: https://www.spaceotechnologies.com/blog/cost-to-put-app-on-app-store/

[3] Y. K. Lee, *Be Focused – Focus Timer*, iOS App. Accessed: Nov. 22, 2025. [Online]. Available: https://apps.apple.com/us/app/be-focused-focus-timer/id973130201

[4] Flow App Ltd., *Flow: Focus & Pomodoro Timer*, iOS App. Accessed: Nov. 22, 2025. [Online]. Available: https://apps.apple.com/us/app/flow-focus-pomodoro-timer/id1423210932

[5] Doist Inc., *Todoist: To-Do List & Planner*, iOS App. Accessed: Nov. 22, 2025. [Online]. Available: https://apps.apple.com/us/app/todoist-to-do-list-tasks/id572688855

[6] L. Mehlig, *Structured – Daily Planner*, iOS App. Accessed: Nov. 22, 2025. [Online]. Available: https://apps.apple.com/us/app/structured-daily-planner/id1499198946

[7] Pomodoro & Task Team, *Focus To-Do: Pomodoro Timer*, iOS App. Accessed: Nov. 22, 2025. [Online]. Available: https://apps.apple.com/us/app/focus-to-do-pomodoro-timer/id966057213