**CS673 Software Engineering** 

**Team 3 - Study Buddy**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Melissa Cron | Project Leader / Requirements Leader | *Melissa Cron* | 10/16/2025 |
| Ittoop Shinu Shibu | Configurations Management Leader | *Ittoop Shinu Shibu* | 10/16/2025 |
| Bohan Lin | Design & Implementation Leader / QA Leader | *Bohan Lin* | 10/16/2025 |
| Junzhe Chen | Security Leader | *Junzhe Chen* | 10/16/2025 |
| Qiuting Zhao | Design & Implementation Leader / QA Leader | *Qiuting Zhao* | 10/16/2025 |
| Dexiao Zhang | Design & Implementation Leader/QA Leader | *Dexiao Zhang* | 10/16/2025 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1.1 | Bohan Lin | 10/1/2025 | Database design |
| 1.2 | Junzhe Chen | 10/1/2025 | HTML design |
| 1.3 | Ittoop Shinu Shibu | 13/10/2025 | Class Diagram |
| 1.4 | Melissa Cron | 10/14/2025 | Introduction, REST APIs, and AI log |
| 1.5 | Bohan Lin | 10/14/2025 | Software architecture |
| 1.6 | Bohan Lin | 10/15/2025 | Design patterns |
| 1.7 | Junzhe Chen | 10/16/2025 | UI design, security design |
| 1.8 | Qiuting Zhao | 10/16/2025 | [Business Logic and Key Algorithms](#_mtfbusfb0eq3) |
| 1.9 | Dexiao Zhang | 10/16/2025 | Design patterns |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_3ipvmjgn6clp)

[Class Diagram](#_ky60nv8suxxm)

[UI Design (if applicable)](#_7ucksmkf6rzx)

[Database Design (if applicable)](#_tcmuor4nl1kz)

[Security Design](#_x18fj36s1121)

[Business Logic and/or Key Algorithms](#_mtfbusfb0eq3)

[Design Patterns](#_9zvwkmc4luo5)

[Any Additional Topics you would like to include.](#_15tmymhipvdv)

[References](#_50ojo9i46ytq)

[Glossary](#_8n34lvocupub)

# Introduction - Melissa

Study Buddy is an app to encourage studying and reduce distractions while studying. As such, our goal is to make a sleek approachable design that has students ready to study in the least amount of clicks possible, while still being engaging enough to encourage them to come back for more.

# Software Architecture - Bohan/Dexiao

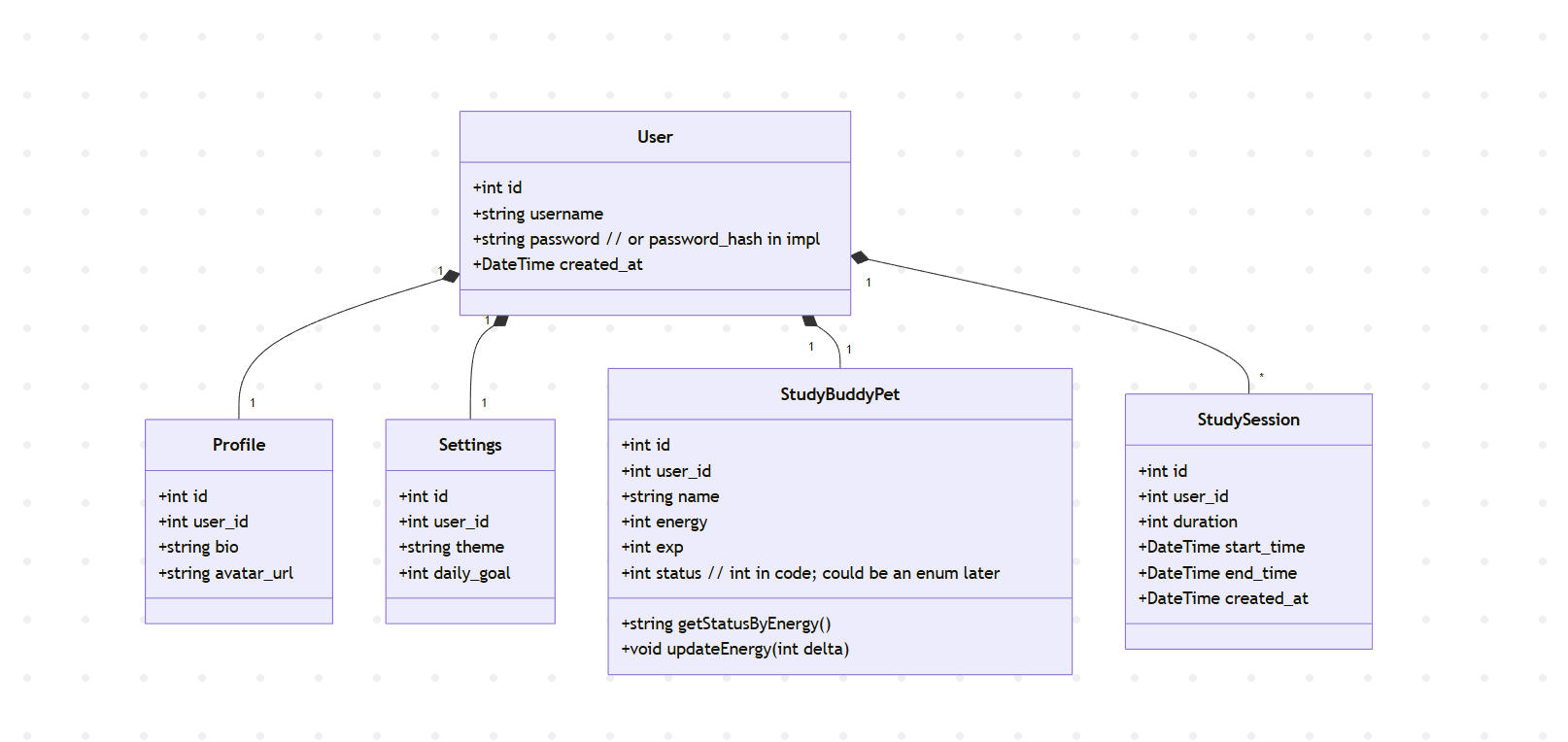
The Study Buddy follows a client-server architecture with a three-tier structure, consisting of the frontend, backend, and database layers. The frontend is implemented as a component-based system, organized into two main folders: screens and components. The screens folder defines the main user interfaces for different pages, such as login, study timer, settings, and so on, while the components folder contains reusable UI components such as buttons, checkboxes and background. The backend is organized into routes, models, and db folders. The routes folder defines API endpoints that serve as the interface between the frontend and backend, handling HTTP requests and responses. The models folder defines the data structures and schemas used to interact with the database, while the db folder manages the actual database connection and queries. The frontend communicates with the backend via RESTful API calls, typically using HTTP GET, POST, PUT, and DELETE methods. The backend depends on the database for persistent data storage and retrieval. The project uses the React Native framework for the frontend and [Express.js](http://express.js) with [Node.js](http://node.js) for the backend.

# Class Diagram - Shinu

In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.

This section has been divided into a general class diagram of our application as well as a component diagram of the frontend.

**Class diagram:**



**User**

Role: Account holder; root owner of related data.

Relationships: Owns exactly one of each: Profile, Settings, StudyBuddyPet; owns many StudySession.

**Profile**

Role: Public/visual user info.

Relationship: Belongs to User.

**Settings**

Role: Personal app preferences.

Relationship: Belongs to User.

**StudySession**

Role: A single timed study interval.

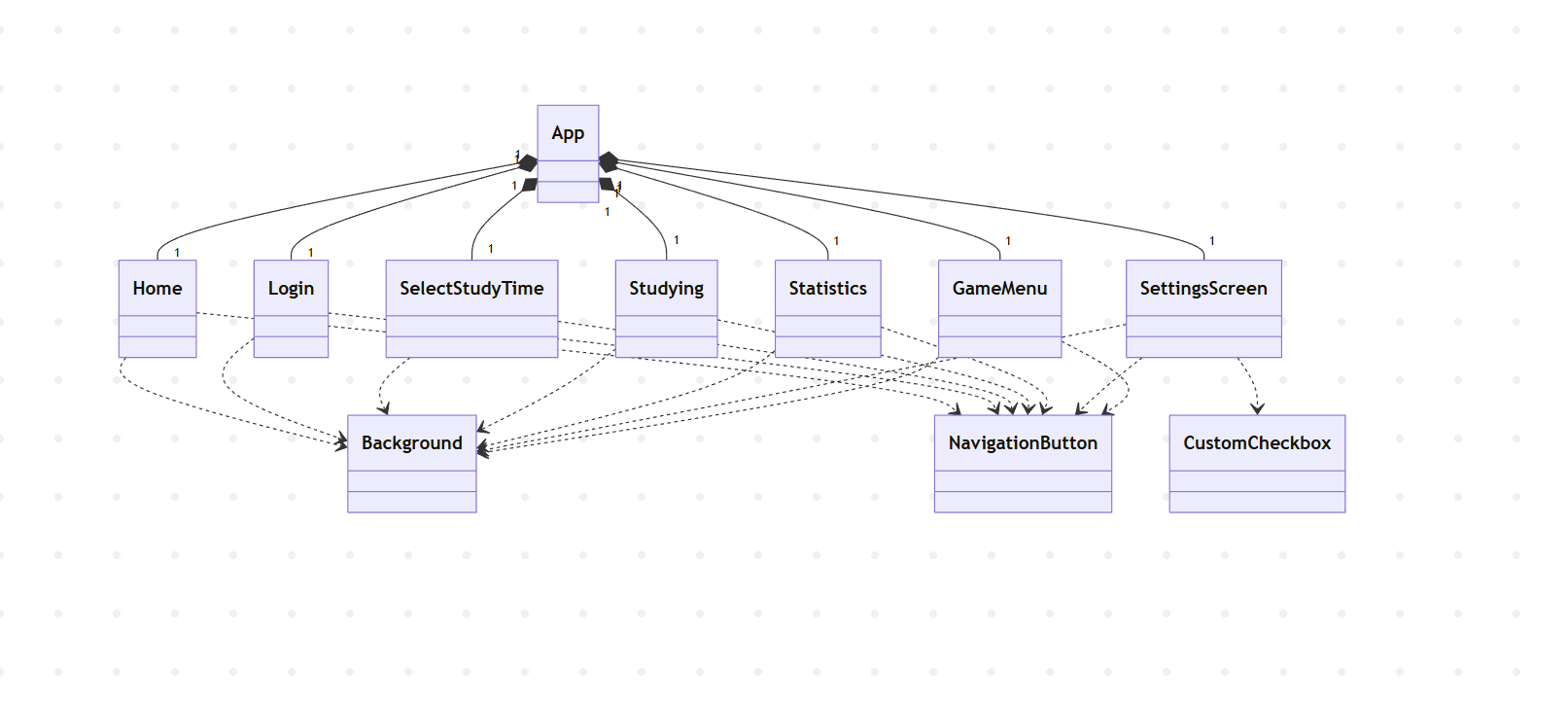
Relationship: Many per User.

**StudyBuddyPet**

Role: Pet reflecting study progress.

Relationship: Belongs to User.

**Component diagram:**



**App shell & navigation**

Component: App

Purpose: Bootstraps the React Native app and declares the navigation graph.

Responsibilities:

• Initialize the navigation container and stack. Owns (creates/destroys) screens as users navigate.

• Register screens (Home, Login, SelectStudyTime, Studying, Statistics, Settings, GameMenu).

Collaborators: All screen components.

**Screens package (screens/)**

Home

Purpose: Landing hub after login.

Responsibilities:

• Presents primary actions (start studying, view stats, open settings, pet/game menu).

• Simple routing to other screens.

Collaborators: Background, NavigationButton.

State/events: Stateless besides local UI; navigates on button press.

Login

Purpose: Collects credentials and kicks off auth flow (stubbed/local for now).

Responsibilities:

• Render inputs and submit action.

• Basic validation feedback (empty fields, etc.).

Collaborators: Background, NavigationButton.

State/events: Local form state; on submit, route to Home (or show error).

SelectStudyTime

Purpose: Configure a session length before starting.

Responsibilities:

• Let users pick a duration (preset options or input).

• Start the session by routing to Studying with params.

Collaborators: Background, NavigationButton.

State/events: Holds the selected duration; validates non-zero positive time.

Studying

Purpose: Focus/timer view during an active session.

Responsibilities:

• Countdown timer, pause/stop controls.

• (Optionally) update pet’s energy/exp at milestones or on completion.

Collaborators: Background, NavigationButton.

State/events: Timer state, lifecycle hooks on mount/unmount; emits “session complete.”

Statistics

Purpose: Show historical metrics and summaries.

Responsibilities:

• Render totals (e.g., total minutes), streaks, most recent sessions.

Collaborators: Background, NavigationButton.

State/events: Reads from stored sessions; handles empty state gracefully.

Settings

Purpose: Personal preferences (theme, daily goal, toggles).

Responsibilities:

• Render controls for preferences and persist changes locally.

Collaborators: Background, NavigationButton, CustomCheckbox.

State/events: Local settings state bound to UI; save on change/navigation.

GameMenu

Purpose: Entry point to pet/game-related actions.

Responsibilities:

• Navigate to pet status, actions, or cosmetics (as implemented).

Collaborators: Background, NavigationButton.

State/events: Stateless hub; routes into pet features.

**Reusable UI components (components/)**

Background

Role: Shared layout/visual wrapper that applies consistent theming/background.

NavigationButton

Role: Primary CTA button with consistent styling and spacing.

CustomCheckbox

Role: Styled checkbox for settings toggles.

# UI Design (if applicable) - Kimi

The login page adopts a clean and modern design style, developed using HTML, CSS, React, and Babel, with Visual Studio Code as the main development and debugging environment.

The styling combines TailwindCSS utility-based design principles with custom CSS (styles.css), using Flexbox layout for centered alignment and responsive adaptation. Ant Design Icons are applied to enhance visual consistency and interactive feedback.

The UI prototype was first created using the Pixso framework and refined in Figma for color palette selection, typography, and component arrangement.

The theme background applies a linear gradient (linear-gradient(135deg, #d041a0, #d3b3d1)), while some page backgrounds experiment with Nano Banana to generate more visually appealing icons, improving the overall interface aesthetics.

# Database Design - Bohan

The Study Buddy app uses SQLite as its database to store user information, study records, and personalization settings. The main design goal is to keep the structure simple, scalable, and ensure that each user’s data is isolated from others.

The **Users** table stores basic identity information. Each user has a unique *id* and *username*, along with their registration time. In the future, if Google login is integrated this table can store the Google-provided user ID and email instead of managing passwords directly.

Each user also has a **Profile**, which contains personal details such as full name, avatar, and their study buddy. This has a one-to-one relationship with the Users table, separating authentication data from user-facing profile information.

In addition, users have their own **Settings** that define app preferences, such as theme and notification preferences. This is another one-to-one relationship, ensuring that each user has their own independent configuration.

The **Study Sessions** table is central to the application’s purpose. Each record corresponds to a study session and stores the start time, end time, duration, and optional notes. Since users can log multiple sessions, this forms a one-to-many relationship with the Users table.

In summary, the database consists of four main tables: Users, Profiles, Settings, and Study Sessions. One user has one profile and one setting but has many study sessions. This design ensures clean data organization while leaving room for future features such as achievements and reminders.

# Security Design - Kimi

The current system adopts a **local login and data storage approach**, which helps minimize the risk of data leakage and unauthorized access. Since all user interactions and validations occur locally, no sensitive information is transmitted to external servers. This design ensures a safer environment during the prototype and development phases.

# Business Logic and/or Key Algorithms - Mia

1. Frontend Logic (React Native + Expo)
   1. Study Timer Control Logic

To manage starting, stopping, and recording a study session. When the user taps the study button, the app either starts or stops a session, depending on the current state.

When the study session starts, *isRunning* triggers a one-second interval that decrements the counter. When the time hits zero or the user stops, the timer resets or records the session.

if start pressed then:

isRunning = true

while isRunning:

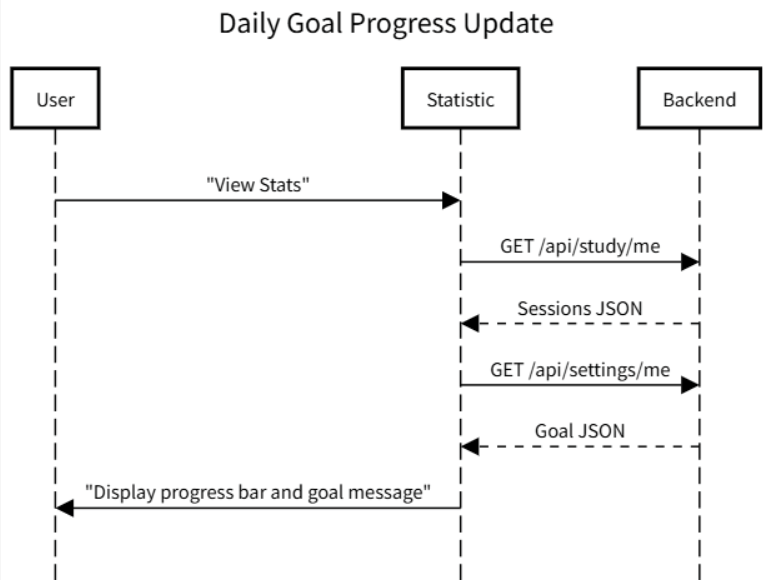
timeLeft = timeLeft - 1 each second

if timeLeft = 0: stop

* 1. Daily Goal Progress Update

To retrieve user session data, calculate today’s total study duration, and compare it against the daily goal.

Each time the user opens the statistics screen, two backend requests are made to retrieve study sessions and user settings. The system calculates the percentage of the daily goal completed and updates the progress bar accordingly.



* 1. Study Time Selection

To allow the user to pick study durations (15min, 30min, 60min, or counting-up) and sends the value to Studying screen.

User-selected time is passed as navigation parameter, time = 0 activates count-up mode.

Display buttons [15,30,60,CountUp]

when pressed -> navigate(“Studying”, time)

* 1. Settings and Checkbox Control

To change user preferences (sound, vibration, etc.) with independent checkbox states.

if checkbox toggled: update corresponding preference state

* 1. Navigation Structure

To manage transitions between app screens (Login, Study, Settings, GameMenu) while preserving state.

Using Expo Router’s <Stack /> component, this algorithm ensures navigation without losing in-memory state. It allows users to switch between modules seamlessly. The state persistence is critical for preserving study timers and user session tokens.

if isAuthenticated():

navigate(“Studying”)

else:

navigate(“Login”)

onNavigation(target):

Stack.navigate(target)

1. Backend Logic (SQLite)
   1. User Registration and Hashing

To safely store user credentials by hashing passwords before database insertion.

The system uses bcrypt to hash the password. This prevents exposure of plaintext passwords in the event of data breaches.

receive username, password

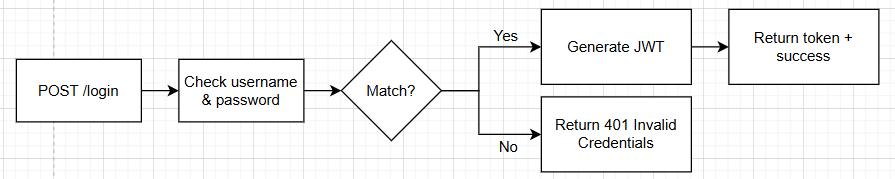
passwordHash <- bcrypt(password)

INSERT user(username, passwordHash)

* 1. JWT Token Login

To authenticate users and issue a JSON Web Token for secure access to protected routes.

After verifying the entered credentials, the system signs a JWT containing the use’s ID. This token is returned to the frontend, where it is stored for authorization in future requests.



* 1. Study Session Creation and Calculation

To record a study session and automatically calculate its duration.

If the frontend sends only start and end times, the backend computes the total duration in minutes and stores the completed record in the *StudySessions* table.

extract user\_id from token

if duration is missing:

duration = (end\_time - start\_time)/60000

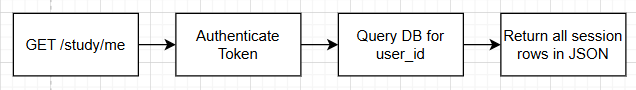
INSERT study session (user\_id, start\_time, end\_time, duration)

return confirmation

* 1. Retrieve Study Sessions

To fetch all study session records belonging to the authenticated user.

This algorithm queries the *StudySessions* table using the current user’s ID and returns all results in descending chronological order.



* 1. Profile Creation and Update

To create or update the user’s profile data efficiently.

Use SQLits’s INSERT OR REPLACE syntax to perform an “upsert” —

if a record exists for the user, it’s replaced; otherwise, a new record is created.

receive user\_id, bio, avatar\_url

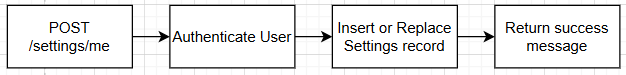
INSERT OR REPLACE INTO Profiles(user\_id, bio, avatar\_url)

return updated profile

* 1. Settings Upsert

To save or update a user’s theme preferences and daily goal in the Settings table.

The algorithm ensures idempotency by replacing previous entries with the same user\_id, allowing smooth updates without duplicate data.

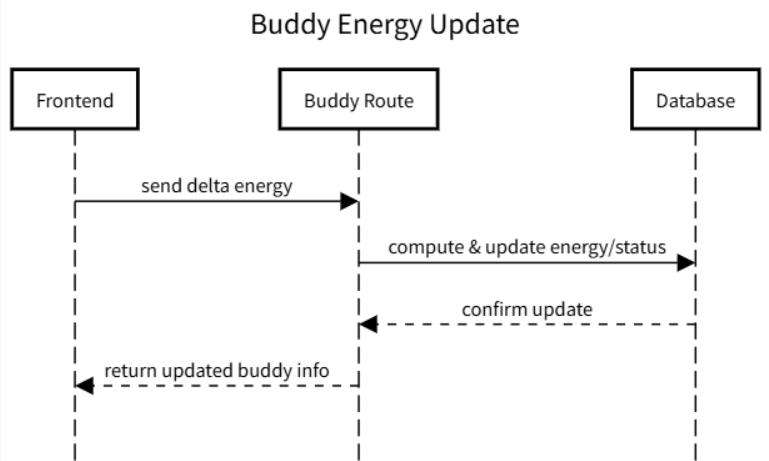


* 1. Buddy Energy and Status Update

To dynamically update the virtual buddy’s energy level and emotional status based on study performance.

Energy is capped at 100. The system categorizes the buddy’s mood:

* Energy > 80 → “Happy”
* 40 < Energy ≤ 80 → “Okay”
* ≤ 40 → “Tired”



* 1. Goal Evaluation (Progress Tracking)

To calculate total study time for the current day and compare it with the user’s goal.

By summing today’s session durations and retrieving the daily goal,

the system can report completion percentage or notify when the goal is achieved.

goal = SELECT daily\_goal FROM Settings WHERE user\_id=id

total = SUM(duration) WHERE date(start\_time)=today AND user\_id=id

progress = total / goal \* 100

return progress

1. Game Logic (Reac Native)
   1. Game1 Physics Simulation Loop

To simulate continuous gravity and motion in the player character across frames.

Each frame increases the player’s vertical velocity by the gravity constant and updates the Y-position accordingly. If the player collides with the ground, velocity resets to 0, stopping downward motion. This loop produces realistic jump/fall motion similar to side-scrolling physics games.

for each frame:

velocity.y = velocity.y + GRAVITY

player.y = player.y + velocity.y

if player.y + PLAYER\_SIZE ≥ groundY:

velocity.y = 0

isJumping = false

if player.y > groundY:

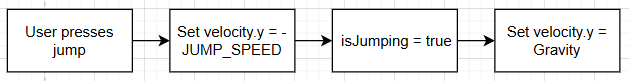
gameover

* 1. Game1 Jump Mechanism

To apply jump velocity when the player releases a press, creating upward motion.

This algorithm triggers when the player releases the screen(*onPressUp*).

It sets the Y-velocity to a negative jump speed, simulating upward acceleration. Gravity then gradually pulls the player down again.



* 1. Game Menu Navigation

To provide an interface for choosing between available games within the app.

This algorithm defines a simple menu with navigation buttons. Each button triggers navigation to either Game1 or Game2 components.

function GameMenu():

display "Game Menu" screen

display button "Game 1"

if user clicks Game 1": navigate("Game1")

display button "Game 2"

if user clicks "Game 2": navigate("Game2")

display button "Return Home"

if user clicks "Return Home": navigate("Home")

* 1. Game2 Core Logic

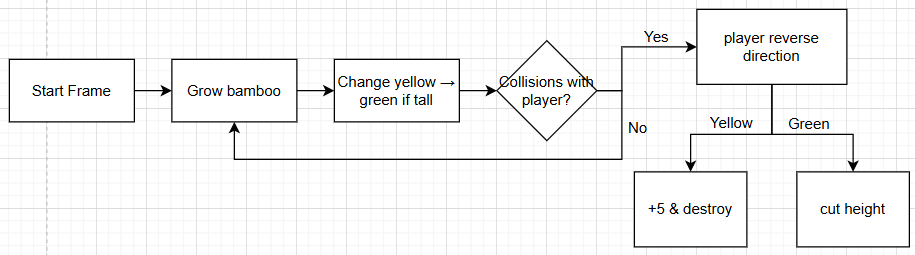
To implement Bamboo’s complete gameplay loop: player movement, bamboo growth, color evolution, collision reaction, and scoring within a continuous frame update.

Each frame, all bamboo objects grow upward; yellow bamboo turns green after reaching a certain height.

The player moves horizontally and reverses direction on wall contact or when colliding with bamboo.

Colliding with yellow bamboo destroys it and increases score; colliding with green bamboo cuts its height instead.

If any bamboo grows too tall, the game ends.



# Design Patterns - Dexiao

Our software incorporates several design patterns to ensure modularity, maintainability, and scalability.

**1. MVC Pattern** The backend is built following the Model–View–Controller (MVC) architecture:

**Models** define the database structure and encapsulate data access logic (e.g., userModel.js, eventModel.js, buddyModel.js).

**Controllers / Routes** handle incoming HTTP requests, perform validations, invoke model functions, and send JSON responses (e.g., /api/users, /api/events, /api/buddy).

**Views** are implemented in the frontend (Next.js / React), consuming RESTful APIs exposed by the backend.  
 This structure improves code readability, supports parallel frontend–backend development, and simplifies debugging and maintenance.

**2. Middleware Pattern** Express middlewares such as auth.js, logger.js, and errorHandler.js are used to handle cross-cutting concerns:

auth.js validates JWT tokens and ensures secure access control.

logger.js records request information for debugging and monitoring.

errorHandler.js unifies error responses for better consistency.  
 This pattern promotes reusability and clean separation of concerns.

**3. Singleton & Modular Configuration Pattern** Database connections (SQLite / PostgreSQL) and configuration files use the singleton pattern to ensure only one active connection throughout the app.  
 Modules like db.js and config.js are imported wherever needed, reducing redundancy and preventing connection conflicts.

**4. RESTful API Design Pattern** The backend follows RESTful principles — each resource (users, events, buddies, attendees) is accessed via standardized endpoints (GET, POST, PUT, DELETE).  
 This pattern ensures scalability and allows future extension (e.g., analytics, notifications) without major architectural changes.

# Rest APIs - Melissa

GET /api/profiles/me

Get the profile of the logged-in user

Parameters:

None

Responses:

200 Profile Data Retrieved Successfully

Returns:

{

id: integer

user\_id: integer

bio: string

avatar\_url: string

}

401 Unauthorized

500 Internal server error

POST /api/profiles/me

Update the profile of the logged-in user

Notes:

If a profile does not exist, it creates a new profile

Parameters:

None

Request Body:

{

bio: string (required)

avatar\_url: string (required)

}

Responses:

200 Profile Updated Successfully

Returns:

{

user\_id: integer

bio: string

avatar\_url: string

}

401 Unauthorized

500 Internal server error

GET /api/settings/me

Get the settings of the logged-in user

Parameters:

None

Responses:

200 Settings Data Retrieved Successfully

Returns:

{

id: integer

user\_id: integer

theme: string

daily\_goal: integer

}

401 Unauthorized

500 Internal server error

POST /api/settings/me

Update the settings of the logged-in user

Notes:

If a settings does not exist for this user\_id, it creates a new settings

Parameters:

None

Request Body:

{

theme: string (required)

daily\_goal: integer (required)

}

Responses:

200 Settings Updated Successfully

Returns:

{

user\_id: integer

bio: string

avatar\_url: string

}

401 Unauthorized

500 Internal server error

GET /api/study/me

Get all the study sessions of the logged-in user

Parameters:

None

Responses:

200 List of Study Sessions

Returns array of jsons formatted in the following way:

{

id: integer

user\_id: integer

duration: integer

start\_time: timestamp

end\_time: timestamp

created\_at: timestamp

}

401 Unauthorized

500 Internal server error

POST /api/study/me

Add a new study session for the logged-in user

Parameters:

None

Request Body:

{

duration: integer (required)

start\_time: timestamp

end\_time: timestamp

}

Responses:

201 Study Session Added Successfully

Returns:

{

id: integer

This is the id of the study session

user\_id: integer

duration: integer

start\_time: timestamp

end\_time: timestamp

}

401 Unauthorized

500 Internal server error

POST /api/users/register

Register a new user

Parameters:

None

Request Body:

username: string (required)

password: string (required)

Responses:

201 User Registered Successfully

Returns:

{

message: string

user: {

id: integer

username: string

}

}

400 Bad request

E.g. missing username/password

POST /api/users/login

Login with username and password

Parameters:

None

Request Body:

username: string (required)

password: string (required)

Responses:

200 Login Successful

Returns:

{

message: string

user: {

id: integer

username: string

}

token: string

}

400 Bad request

E.g. user not found

401 Invalid Credentials

User found but password does not match

GET /api/users/me

Get Current Logged-in User Info

Parameters:

None

Responses:

200 Current User Info

Returns:

{

id: integer

username: string

}

401 Unauthorized

Invalid or missing token

# Any Additional Topics you would like to include.

# AI usage Log

You are allowed and even encouraged to use AI tools to help you generate the project idea, plan it and build it, but you need to clearly describe 1) What tools were used? 2) for what specific tasks and 3) Is it helpful? 4) how did you evaluate or modify AI-generated content? Additionally, you should submit the exported AI chat history as an appendix or share that with the instructor and facilitators.

| Tools | Who | Tasks | helpful | Evaluation/modification | links |
| --- | --- | --- | --- | --- | --- |
| chatgpt | Bohan | Produce swagger code | yes | The AI tool does help to put in swagger comments to the backend code file. However, it does not give the precise and correct comments. It does a great job in providing templates. However, human works are needed to examine and make corrections. |  |
| Nano banana | Junzhe | Find the picture of the buddy | no | The AI tool can provide the thing that we have. We don’t have the basic idea about buddy. |  |
| ChatGPT | Melissa | Consulted about how to use React Native functions | yes | I read the code generated until I understood the function, and then rewrote it to my specifications | <https://chatgpt.com/share/68dde7ce-8d7c-8011-a5df-c7505429da47> |
| ChatGPT | Melissa | Fix a broken test | yes | I copy and pasted error codes into it. ChatGPT was able to describe possible causes and give possible fixes. One of the fixes worked | <https://chatgpt.com/share/68e821e5-8b98-8011-aaae-f4cd791a3945> |
| ChatGPT | Melissa | Consulted about expo command that stopped working |  | I copy and pasted error codes into it. ChatGPT was able to describe possible causes and give possible fixes. One of the fixes worked (yes, this is the same text as the above, but I used it the same way and it was just as effective) | <https://chatgpt.com/share/68ed15a7-2b80-8011-b9aa-2f6e5fc80df1> |
| Github Copilot | Shinu | Consulted Copilot about building a class diagram based on the code |  | Copilot was able to iterate through my code and build a sufficiently good class diagram. I was then able to manually edit it using my own understanding of the code. I manually evaluated if it was accurate. |  |
| ChatGPT | Shinu | Consulted ChatGPT to put together some of the explanations for the class diagram |  | I used chatgpt to put together the explanations on the class diagram. Manually looked through the explanations and edited it as necessary to ensure it was accurate. |  |

# 

# References

# Glossary