Software Engineering Project

KidQuest

May 2025 - Team 25

Milestone - 6

Siddhardh Devulapalli	21f2000579
Jeevan Bidgar	22f2000813
Sreekanth A	23f2003894
Om Aryan	21f3002286
Sinu Maria Jeeson	21f1001608
Pal Amitkumar Shish	21f1006870
Pankaj Mohan Sahu	21f2001203

Table of Contents

Milestone - 1 & 2

Problem | User Requirements | User Stories

1. Problem Statement

Design and implement a web or mobile application for school-aged children (8–14 years) to help them develop essential life skills such as:

- Time Management
- Emotional Intelligence
- Financial Literacy
- Communication
- Healthy Habits

The solution will be informed by real user feedback (children, parents, and educators) and will be designed to actively engage children in practicing these skills through interactive, gamified, and child-safe features.

1.1 User Research & Needs Identification

- Conducted interviews/surveys with target user groups (children, parents, educators).
- Identified gaps in life skills learning and daily routine challenges.
- Selected priority skill areas based on impact, feasibility, and interest.

1.2 Skill Areas & Learning Goals

- Age-appropriate & impactful learning topics.
- Clear, measurable outcomes (e.g., habit streaks, budgeting milestones).
- Engaging, relatable, and safe content for children.

2. Various Users

2.1 User Types

Primary Users:

- Children (8–14 years): Use apps daily for tracking tasks, healthy habits, savings, and completing skill modules. Interact with chatbot, psychometric tests, and earn rewards.

Secondary Users:

- Parents/Guardians: Account setup, monitor progress.
- Educators/Teachers: Assign homeworks, track student progress, and identify students needing support.

Tertiary Users:

- App Administrators/Developers: Maintain, enhance features, add content, analyze data for improvements.

2.2 User Stories

As a Child:

1. I want to track homework and manage my time, so I can complete assignments without last-minute stress.
2. I want reminders to take breaks, to maintain healthy study habits.
3. I want to chat with a friend when I feel sad/anxious, so I can express my feelings safely.
4. I want to record pocket money and savings, so I can learn budgeting skills.

As a Parent:

5. I want regular updates on my child's progress, to support their learning.
6. I want to set privacy controls and monitor content for safety.

As a Teacher:

8. I want to assign homework to my students.
9. I want to track student progress and to provide targeted help.

As an Administrator/Developer:

10. I want to analyze data, to continuously improve app features and engagement.

2.3 Core Features

1. AI Chatbot – Virtual Mentor

- Simple, age-friendly answers to questions.
- Motivational support, tips, and guidance tailored to user activity.
- Resource suggestions for continuous learning.

2. Finance Tracker

- Track savings & expenses, set goals.
- Interactive budgeting challenges.

3. Health Tracker

- Daily wellness checklist with streak counts.
- Water intake counter + weekly trends.
- Charts & badges to reward healthy habits.

4. Psychometric Test

- Assess personality traits, emotional intelligence, and strengths.
- Provide personalized growth suggestions.
- Store and compare past results.

5. Task Tracker

- Manage homework, chores, and personal tasks.
- Pomodoro timers to improve focus.
- Productivity tracking and historical trends.

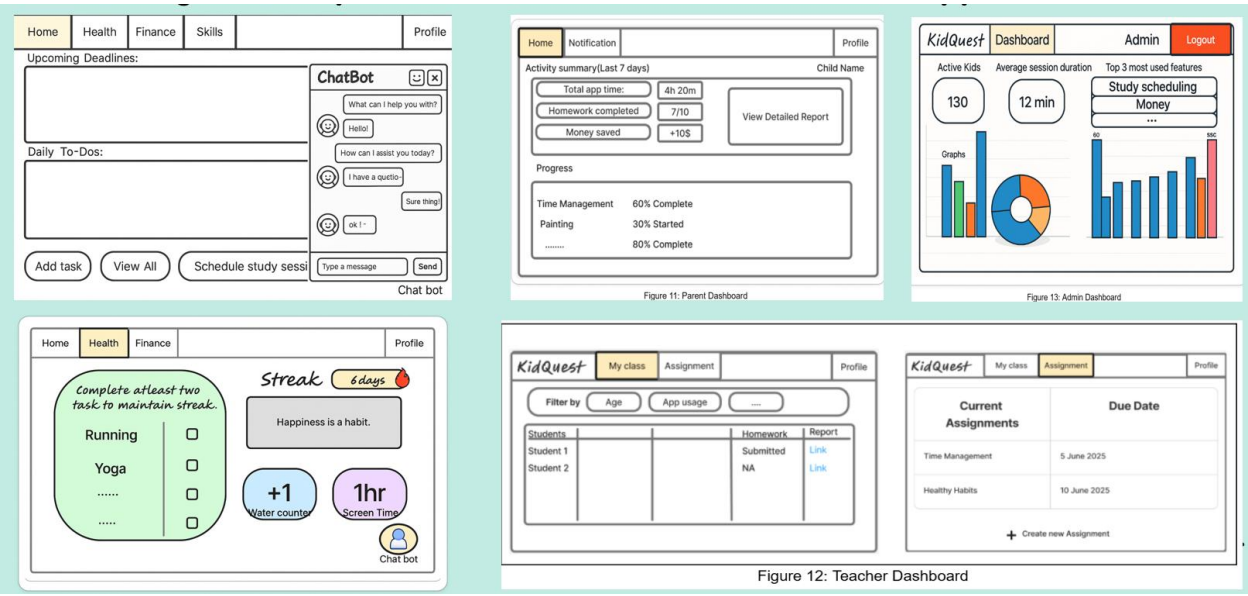
6. Engagement & Motivation

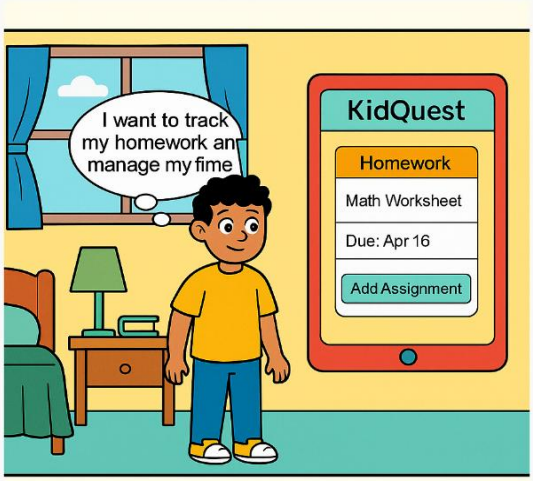
- Gamification: Points, badges, leaderboards, unlockable rewards.
- Story-driven, interactive content.
- Visual animations and engaging UI/UX.

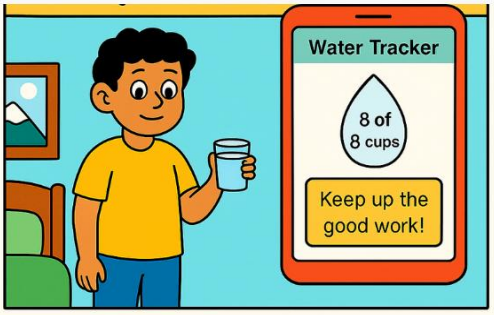
7. Safety & Accessibility

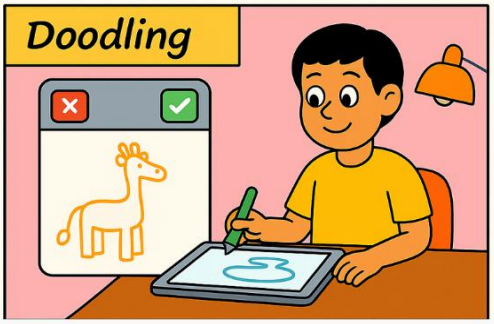
- Parental controls: Monitor usage
- Simple and intuitive interface for young learners.

2.3 Sample Wireframes and Storyboards

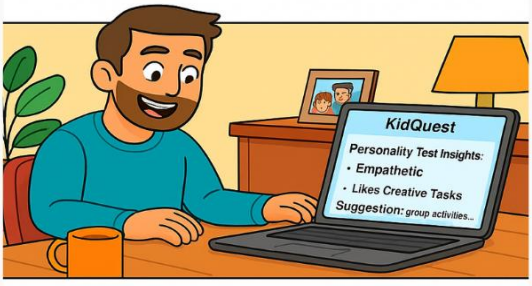








From the perspective of a child learning to manage money



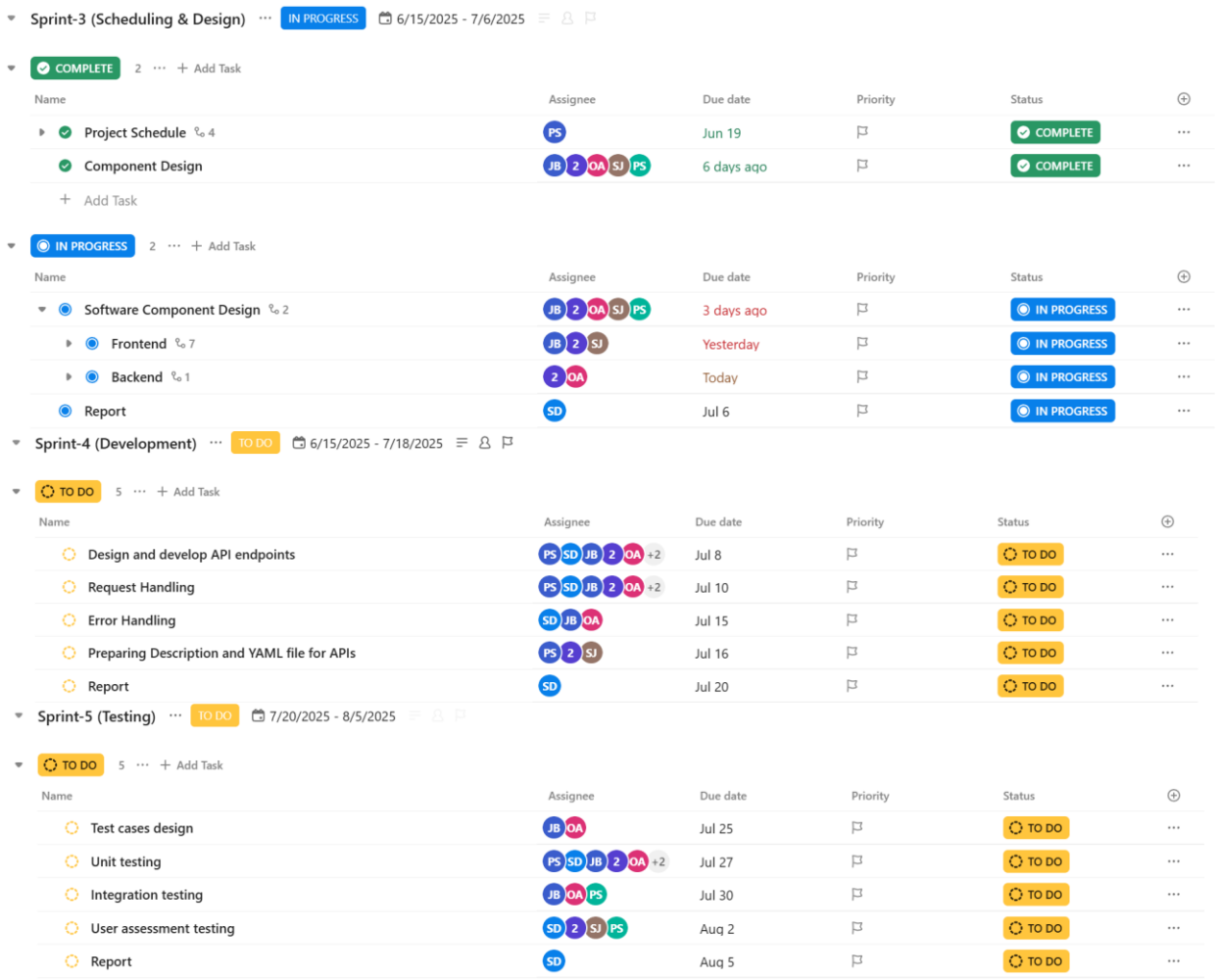
Milestone - 3

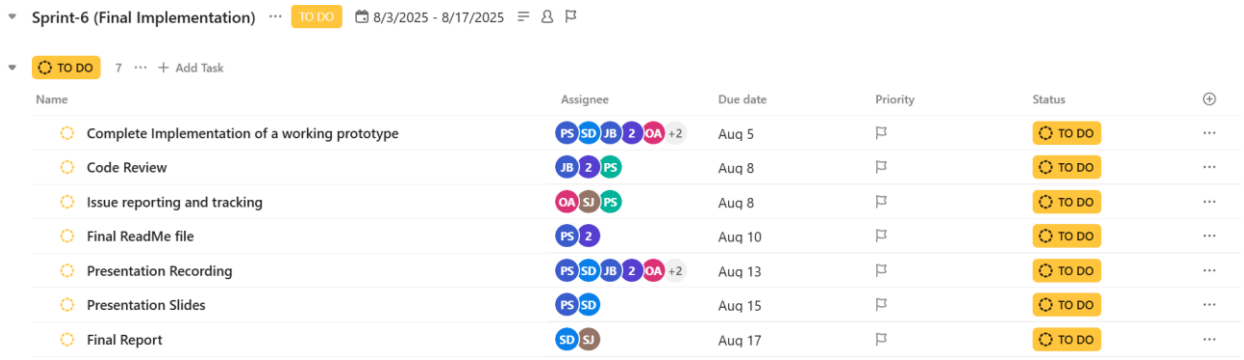
Schedule | Design

3. Project Schedule

3.1 Task Distribution

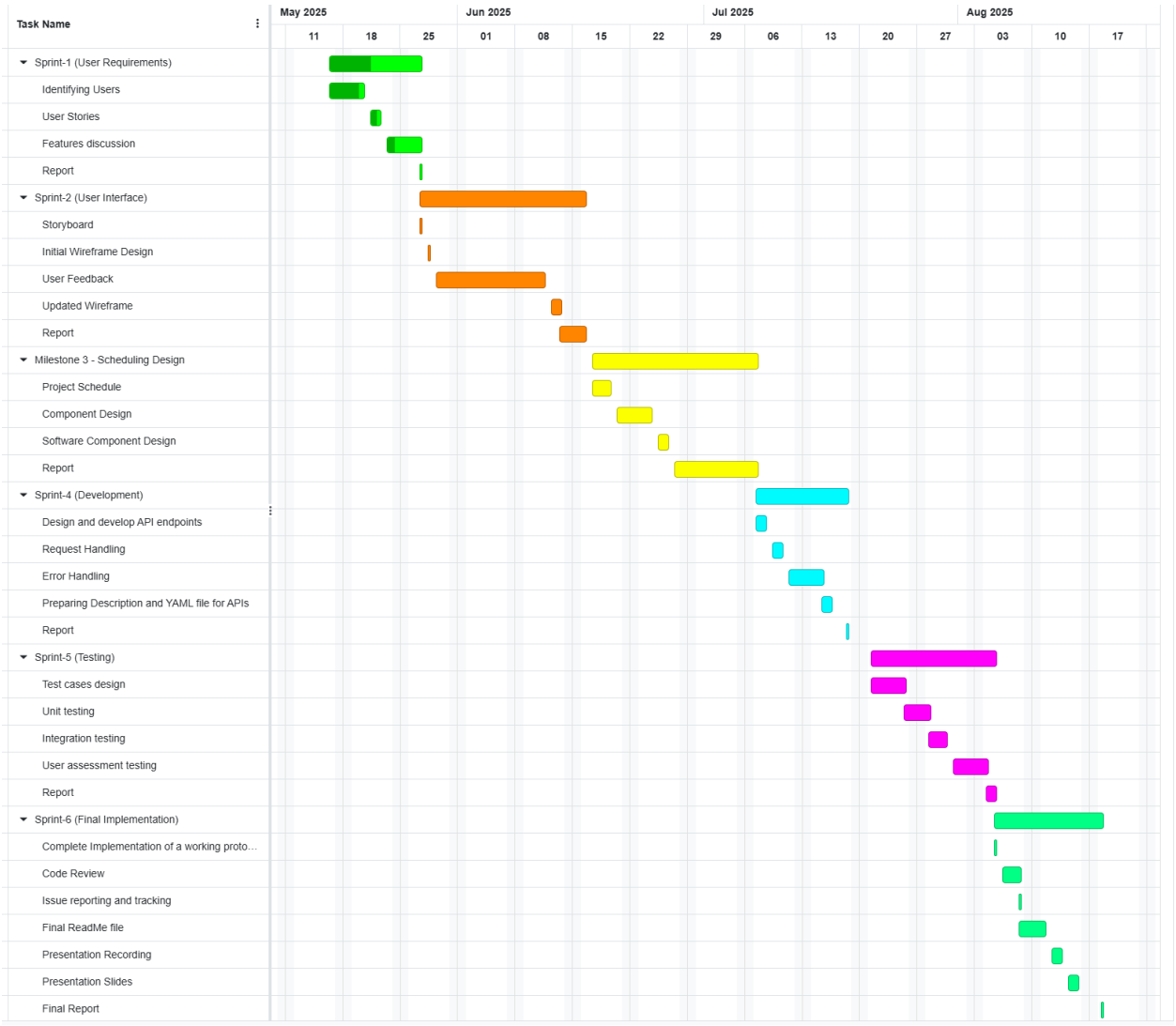
We've strategically mapped out each project milestone, crafting sub-tasks in line with the SMART guidelines— ensuring clarity and achievability. To elevate our approach, we've evenly assigned these tasks to team members, capitalizing on their unique strengths.





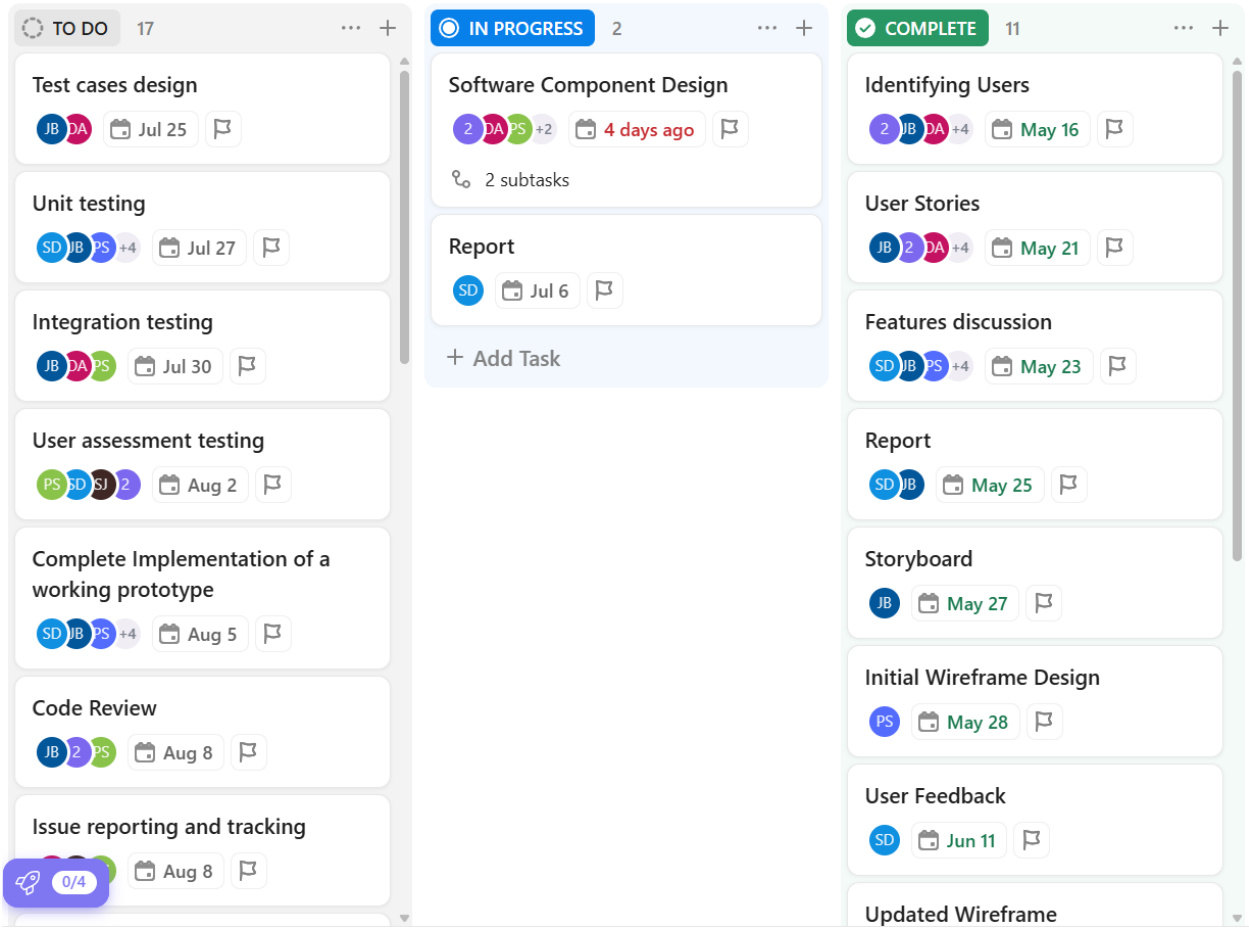
3.2 Gantt Chart

Below is the visual representation of our project schedule through a Gantt Chart. The sprints have been designed in such a way that the dependency of each component is satisfied. The scheduling and management aspects of the project were coordinated using the tool ClickUp.



3.3 Scrum Board

The Scrum Board is divided into three parts - tasks to do, tasks in progress and completed tasks. The Scrum Board for Milestone 3 is shown below.



3.3 Scrum Meetings

The scrum meetings were scheduled biweekly and at critical progress checkpoints. These meetings ensured that all team members were aligned on objectives, progress was tracked, and blockers were addressed in a timely manner. The details of discussions from **Milestone 1 to Milestone 5** are given below.

Milestone 1 – User Identification and Early Feature Ideation

- **Project Overview:** The meeting commenced with a comprehensive analysis of the project's scope. We articulated the problem statement and thoroughly reviewed the associated requirements.
- **Understanding the Problem Statement:** We engaged in an in-depth conversation about the potential users, focusing on strategies to accurately categorize them, thereby ensuring that our solutions align with user needs and expectations.
- **Identification and Categorization of Users:** Following our discussion, we systematically identified and categorized users, enabling us to formulate a

structured approach to address the needs of various user segments.

- **User Stories and Feature Brainstorming:** Based on user stories, the team proposed features including educational games, emotional support chatbot, time management tools, interactive doodling, financial literacy, safety education, daily updates, skill development, and health guidance.

Milestone 2 – Design, Task Allocation

- **Discussion of Wireframes:** The meeting commenced with an evaluation of wireframe structures and designs, ensuring a shared understanding of their pivotal role and anticipated deliverables.
- **Discussion of Functionalities:** We conducted a thorough examination of the functionalities to be integrated, fostering a collaborative atmosphere to refine these components for improved user interaction.
- **Task Distribution:** Roles assigned for milestone deliverables, including Gantt charts, Scrum boards, ER diagrams, UML diagrams, and frontend/backend explanations..
- **UI/UX Coordination:** Designers and backend developers aligned on wireframes and functional workflows.

Milestone 3 – Frontend Integration and Feature Completion

Progress Updates:

1. Development of Admin, Parent, and Child dashboards.
2. Implementation of Health tracker, Pomodoro timer, Financial tracker, Emotion chatbot, and Doodling modules.

Frontend Compilation: Assigned tasks for integrating and pushing UI changes to GitHub.

Feature Prioritization: Decision to limit gamification scope and focus on core features for timely delivery.

Milestone 4 – Backend Integration, Gamification, and Security Enhancements

Pending Functionalities:

- Linking notification and achievement models to all features.
- Backend integration for dashboards.
- RBAC with token-based authentication.
- Gamification elements.

Feature Tracking: Added completion tracking for Good/Bad Touch modules and Pomodoro task progress.

Security Measures: Implemented hashing in RBAC and modularized backend code

for scalability.

Backend Review: Each member tasked with demonstrating backend work to the client.

Error Handling: All API endpoints to be wrapped in try-catch blocks with proper error codes/messages to support YAML-based API testing.

Testing Phase: Initiated pytest-based testing for early bug detection.

Milestone 5 – Testing, Bug Resolution, and Final Feature Review

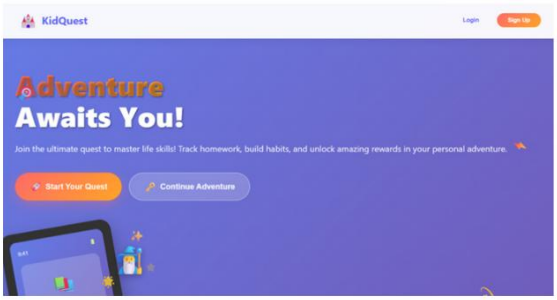
- **Feature Demonstrations:** Teacher Dashboard updates, Parent Dashboard near completion, Admin Dashboard development in progress.
- **Testing Documentation:** Members prepared pytest scripts and documentation for client review.
- **Issue Resolution:** Addressed JWT authentication problems and planned merging of frontend and main branch.
- **Pending Deliverables:** Analytics module and final dashboard integrations still under development.
- **Client Feedback Loop:** Meetings ended with actionable items to refine both features and test outcomes in line with client suggestions.

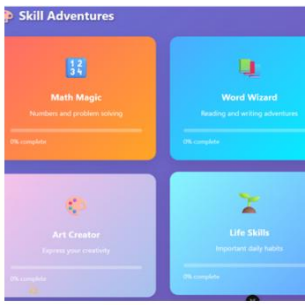
4. Project Design

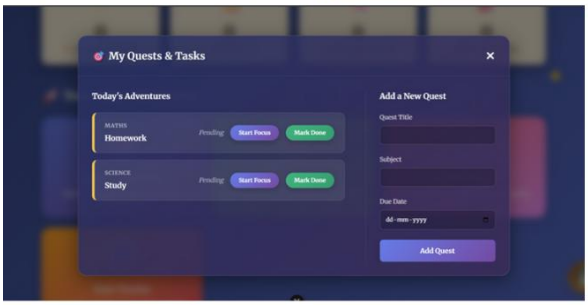
4.1 Frontend

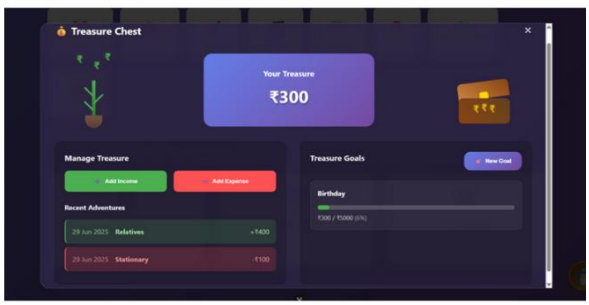
Child Dashboard

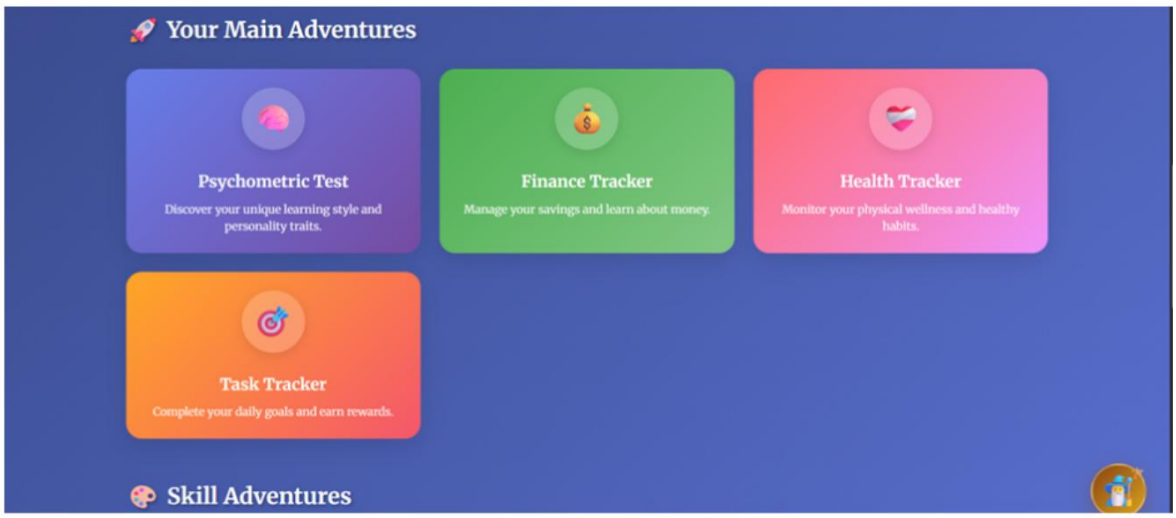

Pomodoro Timer
25:00
Focus Time

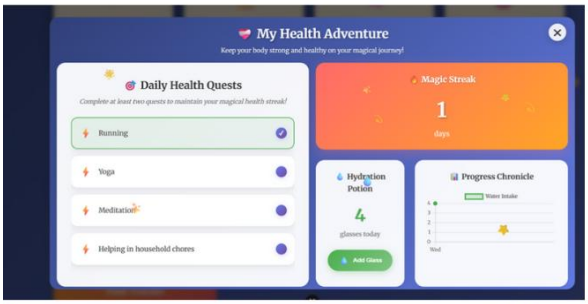

KidQuest
Login
Sign Up
Adventure
Awaits You!
Join the ultimate quest to master life skills! Track homework, build habits, and unlock amazing rewards in your personal adventure.
Start Your Quest
Continue Adventure

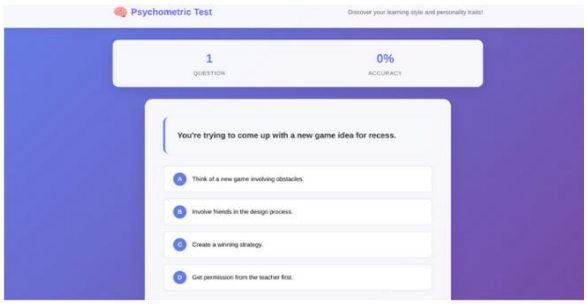

Skill Adventures
Math Magic
Word Wizard
Art Creator
Life Skills

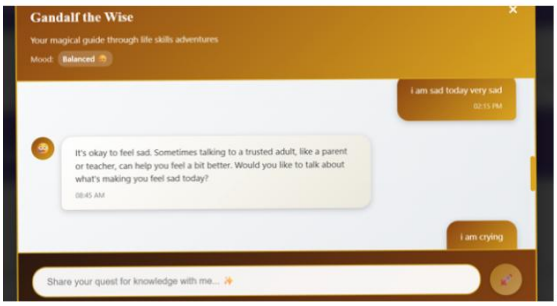

My Quests & Tasks
Today's Adventures
Homework
Study
Start Focus
Mark Done
Add a New Quest
Quest Title
Subject
Due Date
Add Quest


Treasure Chest
Your Treasure
₹300
Manage Treasure
Add Income
Add Expense
Recent Adventures
Treasure Goals
New Goal
Birthday

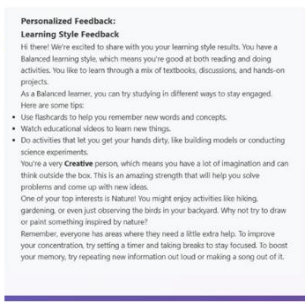

Your Main Adventures
Psychometric Test
Discover your unique learning style and personality traits.
Finance Tracker
Manage your savings and learn about money.
Health Tracker
Monitor your physical wellness and healthy habits.
Task Tracker
Complete your daily goals and earn rewards.
Skill Adventures


My Health Adventure
Keep your body strong and healthy on your magical journey!
Daily Health Quests
Magic Streak
1
days
Running
Yoga
Meditation
Helping in household chores
Hydration Potion
4
glasses today
Add Glass
Progress Chronicle
Water Intake

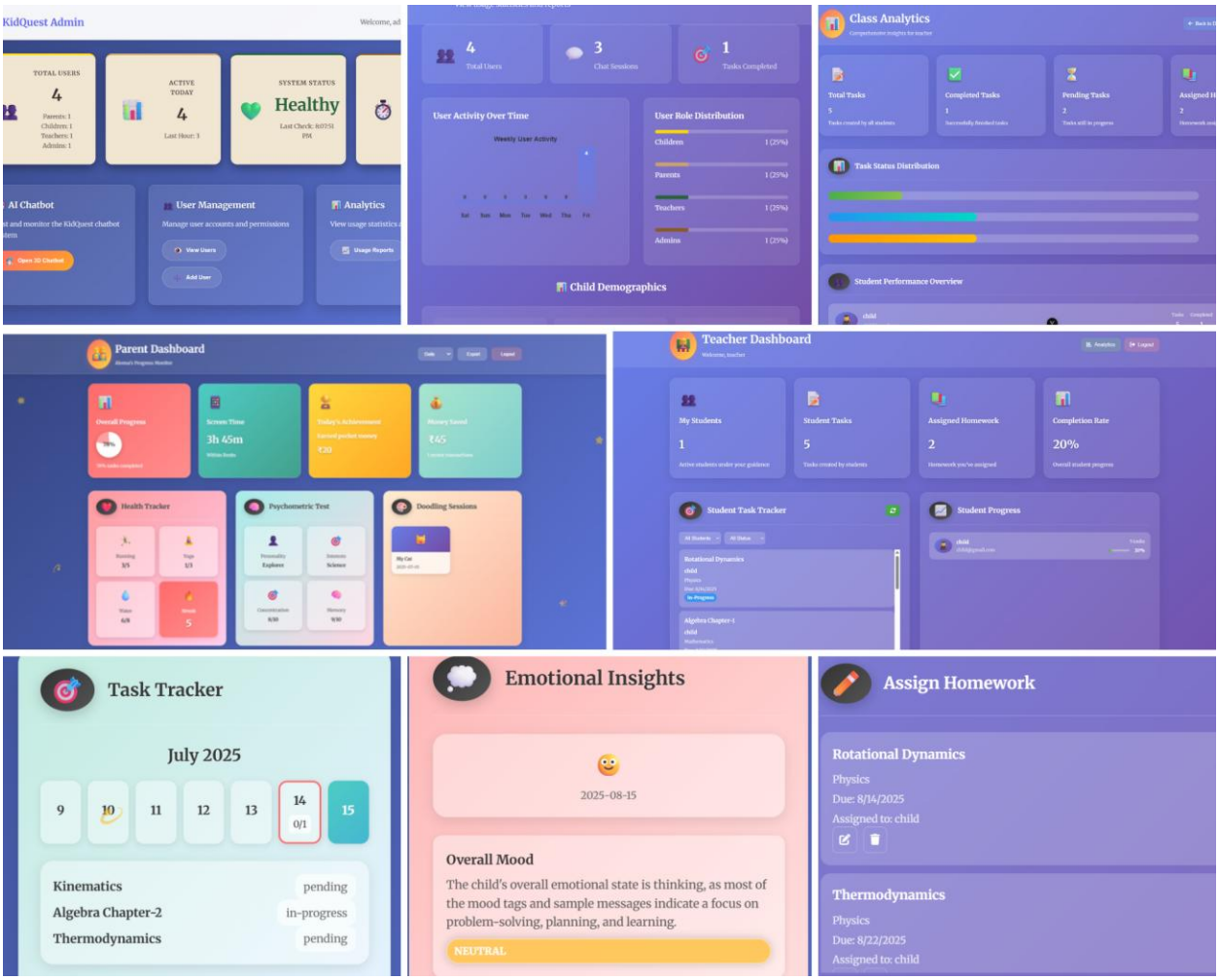

Psychometric Test
Discover your learning style and personality traits!
1
QUESTION
0%
ACCURACY
You're trying to come up with a new game idea for recess.
Think of a new game involving obstacles.
Involve friends in the design process.
Create a winning strategy.
Get permission from the teacher first.


Gandalf the Wise
Your magical guide through life skills adventures
Mood: Balanced
i am sad today very sad
02:55 PM
It's okay to feel sad. Sometimes talking to a trusted adult, like a parent or teacher, can help you feel a bit better. Would you like to talk about what's making you feel sad today?
08:45 AM
i am crying
Share your quest for knowledge with me...


Assessment Complete!
Balanced
LEARNING STYLE
Practical
PERSONALITY TYPE
0%
CONCENTRATION
33.3%
MEMORY
20
TOTAL QUESTIONS
89.5s
TEST DURATION
Category Breakdown
Concentration: 0% (0/1)
Memory: 33.3% (1/3)


Personalized Feedback:
Learning Style Feedback
Hi there! We're excited to share with you your learning style results. You have a Balanced learning style, which means you're good at both reading and doing activities. You like to learn through a mix of textbooks, discussions, and hands-on projects.
As a Balanced learner, you can try studying in different ways to stay engaged. Here are some tips:
Use flashcards to help you remember new words and concepts.
Watch educational videos to learn new things.
Do activities that let you get your hands dirty, like building models or conducting science experiments.
You're a very Creative person, which means you have a lot of imagination and can think outside the box. This is an amazing strength that will help you solve problems and come up with new ideas.
One of your top interests is Nature! You might enjoy activities like hiking, gardening, or even just observing the birds in your backyard. Why not try to draw or paint something inspired by nature?
Remember, everyone has areas where they need a little extra help. To improve your concentration, try setting a timer and taking breaks to stay focused. To boost your memory, try repeating new information out loud or making a song out of it.

Parent, Teacher and Admin Dashboards



4.2 Components of the Project

Frontend – *Vue.js-Based User Interface*

The KidQuest Life Skills App frontend is developed using Vue.js to deliver engaging, role-specific experiences for children, parents, and administrators. Its design focuses on fun, interactivity, and ease of navigation while ensuring safety and accessibility.

Key Pages & Features

Home Page

- For Children & Parents: Bright, welcoming login/start screen.
- Showcases key features: homework tracking, habit building, rewards.
- Buttons: “Start Your Quest” & “Continue Adventure” for smooth onboarding.

Sign-Up

- Step-by-step role selection (Child/Guardian) and registration form.
- Playful visuals, secure fields, and clear instructions for easy, safe signup.

Child Dashboard

- Displays progress stats: stars earned, quests completed, skills mastered, daily goals.
- Direct links to:
    - Psychometric Tests
    - Finance Tracker
    - Health Tracker
    - Task Tracker

Finance Tracker (*"Treasure Chest"*)

- Track income & expenses, set savings goals.
- View transaction history with child-friendly visuals.
- Encourages budgeting and smart money management.

Task Tracker

- Organize homework, chores, and personal tasks.
- Start focus sessions, mark tasks complete, add new ones.
- Promotes time management and responsibility.

Pomodoro Timer

- 25-minute work intervals with break reminders.
- Simple countdown interface for better focus.

Doodling Session (*"Creative Canvas"*)

- Drawing area with colors, brush sizes, and shape tools.
- Encourages creativity and relaxation between study tasks.

Health Tracker (*"My Health Adventure"*)

- Log daily activities: yoga, exercise, chores.
- Water intake tracker with 7-day history.
- Streak counters and badges for healthy habit motivation.

Psychometric Tests

- Interactive, age-friendly assessments for:

        - Learning styles
        - Strengths
        - Personality traits
- Personalized feedback and study improvement tips.

Emotion Chatbot (*"Gandalf the Wise"*)

- Listens to a child's emotions.
- Offers empathetic advice and suggests talking to adults.
- Mood display & safe conversation topics.

Skill Development Modules

- Themed learning adventures:
    - Math Magic
    - Word Wizard
    - Science Explorer
    - Art Creator
    - Life Skills
    - Safety Measures
- Progress tracking for motivational learning.

Parent Dashboard

- Overview of child's:
    - Task completion
    - Savings progress
    - Psychometric results
    - Mood & activity trends
- Recommendations for skill growth.

Teacher Dashboard

- Overview of child's:
    - Task Progress
    - Homework Assignment
    - Analytics on student progress

Admin Dashboard

- Tools for:
    - User account monitoring
    - Chatbot testing
    - Analytics & performance metrics
- Streamlined platform management.

Backend – *Flask-Based Middleware*

The backend is built with Flask (Python 3.10+) to act as the secure communication bridge between the Vue.js frontend, database, and AI/LLM services. It follows modular, maintainable, and role-secure architecture.

Core Backend Architecture

1. Request Handling

- Flask routes manage:
  - Tasks & homework
  - Finance tracking
  - Health data logging
  - Psychometric test workflows
  - AI chatbot interactions
- Flask-CORS enables secure frontend-backend communication.

2. Business Logic & Modular Services

- Each feature (Chatbot, Finance Tracker, Task Tracker, Health Tracker) is a separate service module.
- Middleware responsibilities:
  - Input validation
  - Business rule application
  - Database or AI service updates

3. Database Integration

- Flask-SQLAlchemy ORM for relational database handling (SQLite during development).
- Models include:
  - User, ChildProfile
  - Transaction, SavingGoal
  - HomeworkSchedule, PomodoroSession
  - Notification
- Strong parent-child relationships & foreign key constraints.

4. Authentication & Authorization

- Session-based authentication using Flask sessions.
- Password security with werkzeug.security hashing/verification.
- Role-based access control (RBAC) ensures:
  - Children → restricted learning features
  - Parents → monitoring & controls
  - Admins → full platform access

5. AI & LLM Integration

- ChatGroq-compatible APIs & OpenRouter APIs.
- AI-driven features:
    - Emotion Chatbot → empathetic advice via custom prompts.
    - Psychometric Engine → adaptive testing & categorization:
        - Cognitive abilities
        - Personality traits
        - Interest profiling

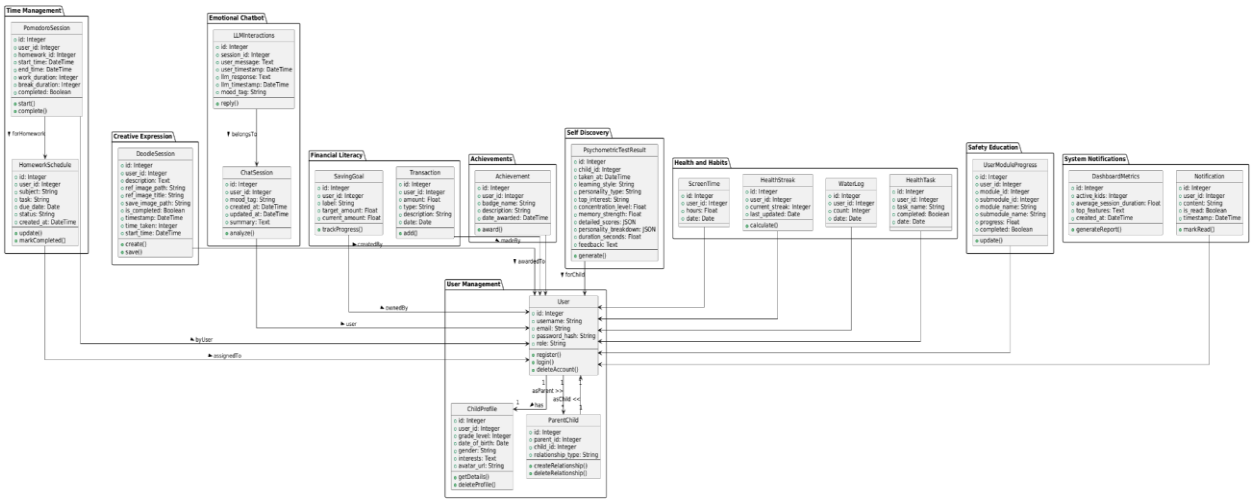
6. Utilities & Tooling

- Regex validation for clean, safe inputs.
- Session tracking for quizzes.
- Environment-specific configs for API keys & secrets.
- Auto admin account creation during initialization.

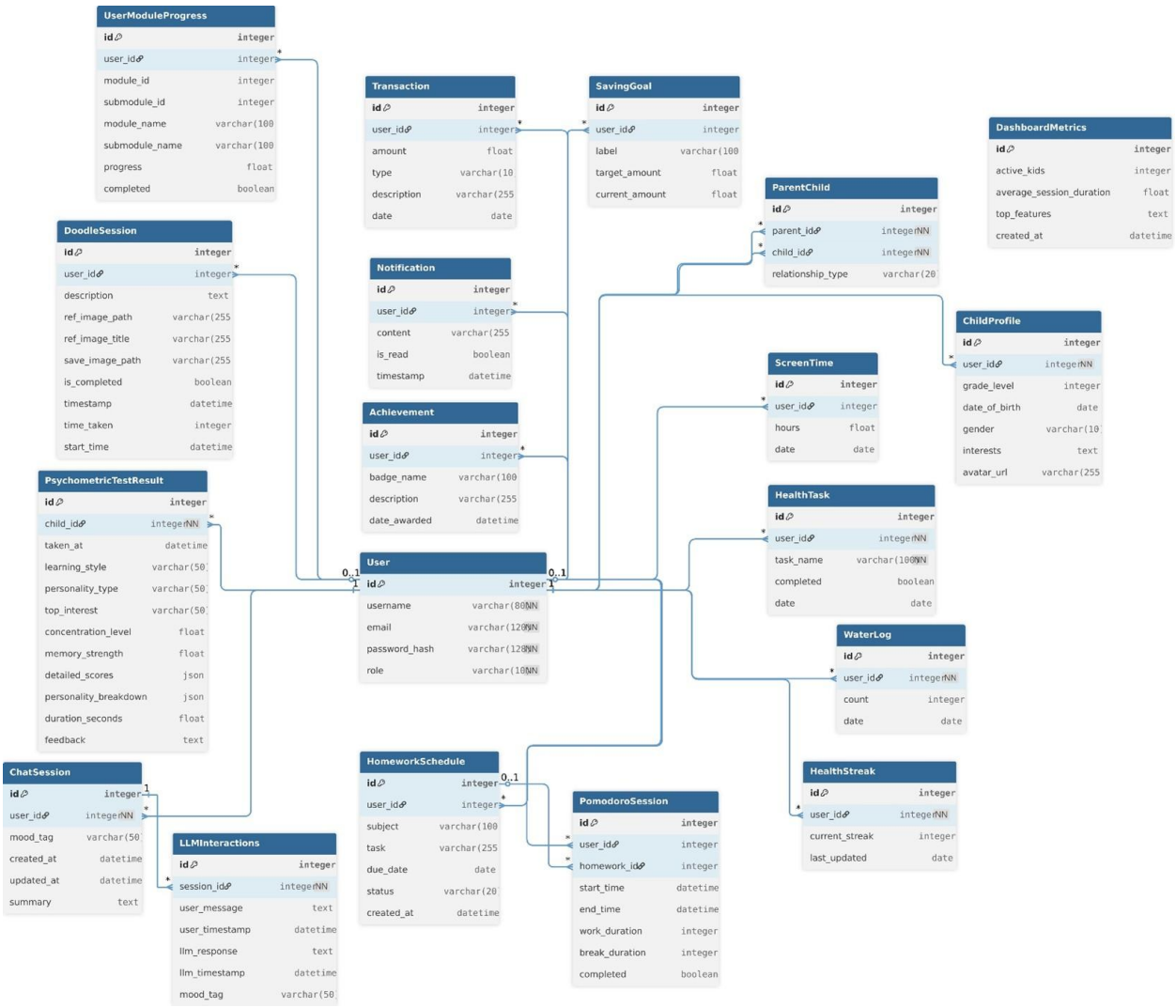
7. Security & Error Handling

- Strong secret keys with secrets.token_hex.
- Role-protected sensitive endpoints.
- Sanitized AI prompts for safe chatbot replies.
- Centralized error logging with structured tracebacks.

4.3 Class Diagram *(click here and zoom for clearer view)*



4.4 ER Diagram (*click here and zoom for clearer view*)



Milestone - 4

APIs | YAML

5. API Endpoints

Authentication and User

- POST /api/auth/register — Register a new user (child/parent/teacher) and optionally link relationships.
- POST /api/auth/login — Authenticate user and return a JWT with role and metadata.
- POST /api/auth/logout — Stateless logout endpoint; clears server-side artifacts like notifications where applicable.
- GET /api/auth/verify — Verify current JWT and return user plus token timing info.
- GET /api/user/profile — Get the current authenticated user's profile (JWT identity).
- GET /api/user/profile/{user_id} — Get a specific user's profile by id.

Chat and Sessions

- POST /api/chat — Send a message to the emotion chatbot, creating/using a session and returning reply+mood.
- GET /api/chat/sessions/{user_id} — List all chat sessions for a user with metadata (JWT required).
- GET /api/chat/session/{session_id} — Get a detailed chat session with messages (owner-only).
- PUT /api/chat/session/{session_id}/summary — Update the summary for a chat session (owner-only).
- GET /api/chat/history/{user_id} — Get the latest cross-session chat history for the user.
- POST /chatbot — Legacy chatbot endpoint (non-JWT) for backward compatibility.
- GET /chat-history/{user_id} — Legacy consolidated chat history (JWT).
- DELETE /clear-chat/{user_id} — Legacy route to clear all chat history for a user (JWT).
- GET /api/chat/mood-summary/{user_id} — Summarize today's moods from chat interactions, with sample messages.

Students/Teacher

- GET /api/students/available — List available students (children) for teacher registration.
- GET /api/teacher/students/{teacher_id} — Get students assigned to a teacher (Bearer header check).

- GET /api/teacher/student-tasks/{teacher_id} — Get all tasks created by students under a teacher (JWT, teacher-only).
- GET /api/teacher/homework/{teacher_id} — Get homework assigned by a teacher (temporary Bearer token check).
- POST /api/teacher/assign-homework — Assign homework to multiple authorized students (JWT teacher-only).

Health Tracker

- GET /api/health/tasks/{user_id} — Get/create today's health tasks for a user (default tasks if none).
- POST /api/health/tasks/{task_id}/toggle — Toggle completion of a specific health task (owner-only).
- GET /api/health/streak/{user_id} — Get current health streak days for a user.
- POST /api/health/water/{user_id} — Increment today's water intake count.
- DELETE /api/health/water/{user_id} — Decrement today's water intake count.
- GET /api/health/water/{user_id} — Get today's water intake count.
- GET /api/health/water/log/{user_id} — Get last 7–8 days water intake log.

Quotes and Streaks

- GET /api/quote/{user_id} — Get a motivational quote for the user (fallback if API fails).
- GET /api/login-streak/{user_id} — Get login streak, totals, longest streak, and last login date.

Child Dashboard and Achievements

- GET /api/child/stats/{user_id} — Get dashboard KPIs: stars, quests, skills, today's goals, streak, and level.
- POST /api/achievement/test — Create a simple test achievement for the authenticated user.
- GET /api/achievements/special/{user_id} — Compute three special achievement cards (knowledge, streak, tasks).
- POST /api/achievements/refresh/{user_id} — Force refresh of special achievements for testing.
- GET /api/achievements/{user_id} — List user-facing achievements, skipping module-progress pseudo-badges.
- POST /api/achievement — Create a new achievement record (open to any caller).
- POST /api/activity/complete — Award an achievement when a named activity is completed.

Child Quests

- GET /api/child/quests/{user_id} — Get today's mock quests for a child (sample data).

- POST /api/child/quest/{quest_id}/toggle — Toggle quest completion (mock response).

Tasks and Pomodoro

- GET /api/tasks/{user_id} — Get user's tasks with aggregated Pomodoro stats (owner-only).
- GET /api/tasks-for-parent/{user_id} — Parents get a child's task list if linked by relationship.
- POST /api/tasks — Create a new task (self-only), with due date validation.
- PUT /api/tasks/{task_id}/status — Update task status (owner or parent).
- DELETE /api/tasks/{task_id} — Delete a task (owner or assigning teacher).
- POST /api/pomodoro/start — Start a Pomodoro session for a task (self-only).
- PUT /api/pomodoro/complete/{session_id} — Complete a Pomodoro session with work/break durations (self-only).
- PUT /api/pomodoro/pause/{session_id} — Pause an active Pomodoro session (self-only).
- PUT /api/pomodoro/resume/{session_id} — Resume a paused Pomodoro session (self-only).
- PUT /api/pomodoro/abandon/{session_id} — Abandon a session, persisting partial durations (self-only).
- GET /api/pomodoro/last-session/{user_id}/{homework_id} — Fetch last Pomodoro session for a specific task (owner-only).
- GET /api/task-time/analytics/{user_id} — Get time analytics and recent Pomodoro sessions, optionally per homework.

Finance Tracker

- GET /api/finance/transactions/{user_id} — Get a user's transactions (self or parent).
- POST /api/finance/transaction — Add a new transaction (self-only).
- GET /api/finance/goals/{user_id} — Get a user's saving goals (self or parent).
- POST /api/finance/goal — Create a new saving goal (self-only).
- GET /api/parentchild — Return parent-child links (for debugging/inspection).

Screen Time

- POST /api/screen-time/log — Log screen time in hours for today (upserts aggregation).
- GET /api/screen-time/{user_id} — Get today's hours, weekly average, and status badge.

Module Progress

- POST /api/module/progress — Save a module or submodule progress entry (self-only with validation).
- GET /api/module/progress/{user_id}/{module_type} — Get progress for a specific module (with submodule breakdown).

- GET /api/module/progress/{user_id} — Get all module progress grouped with overall completion metrics.
- GET /api/modules/info — Get the modules catalog with submodule metadata.

Child Progress and Skills

- GET /api/child/progress/{user_id} — Get overall progress percentage from modules, tasks, and achievements (weighted).
- GET /api/child/skill-progress/{user_id} — Get per-module skill progress with icons.

Notifications

- GET /api/notifications/{user_id} — Get notifications (self; or parent if linked).
- POST /api/notifications/mark-read — Mark a set of owned notifications as read (self-only).

Stories (Creative Writing)

- GET /api/stories/{user_id} — Get all stories for the authenticated user id (owner-only).
- POST /api/stories — Create a story for the current user (title/content required).
- PUT /api/stories/{story_id} — Update an owned story by id.
- DELETE /api/stories/{story_id} — Delete an owned story by id.
- GET /api/stories/{story_id} — Get a single owned story by id.

Drawings / Doodle

- POST /api/drawings/save — Save a base64 image to disk and DB as a completed drawing session.
- GET /api/drawings/{user_id} — Get all drawings for a user (file existence flagged).
- POST /api/drawings/start-session — Start a new drawing session with reference image.
- GET /api/drawings/image/{drawing_id} — Get the base64 image data and metadata for a drawing.
- DELETE /api/drawings/delete/{drawing_id} — Delete a drawing file and record.
- GET /api/drawings/reference-images — List available reference images (auto-generate placeholders if missing).
- GET /api/drawings/random-reference — Return one random reference image.

Psychometric Assessment

- POST /api/psychometry/start — Initialize a new assessment, store state in session, return first question.
- POST /api/psychometry/submit — Submit an answer and return next question or final results.

- GET /api/psychometry/results/{child_id} — Get the latest psychometric result for a child.

Admin and Analytics

- GET /api/admin/dashboard-stats — Summary counts for admin dashboard (users, usage, activity).
- GET /api/admin/analytics — Comprehensive analytics: activity, screen time, health, creativity, financial, demographics.
- GET /api/admin/user-activity — Weekly activity breakdown and role activity with rates.
- GET /api/admin/task-stats — Task totals, completion rate, subject breakdown, recent completions.
- POST /api/admin/migrate-login-history — Populate LoginHistory from existing streaks (one-time migration tool).
- GET /api/admin/users — List all users (admin-only) with profile completeness info.
- POST /api/admin/users — Create a user (admin-only) with validation and uniqueness checks.
- DELETE /api/admin/users/{user_id} — Delete a user and cascade-delete associated data (admin-only).
- PUT /api/admin/users/{user_id} — Update user fields (admin-only) with validation and conflict checks.

Profiles

- GET /api/child-profile/{user_id} — Get a child's profile by user id.
- POST /api/child-profile — Create or update a child profile by payload.
- PUT /api/child-profile/{user_id} — Update a child profile for the given user id.

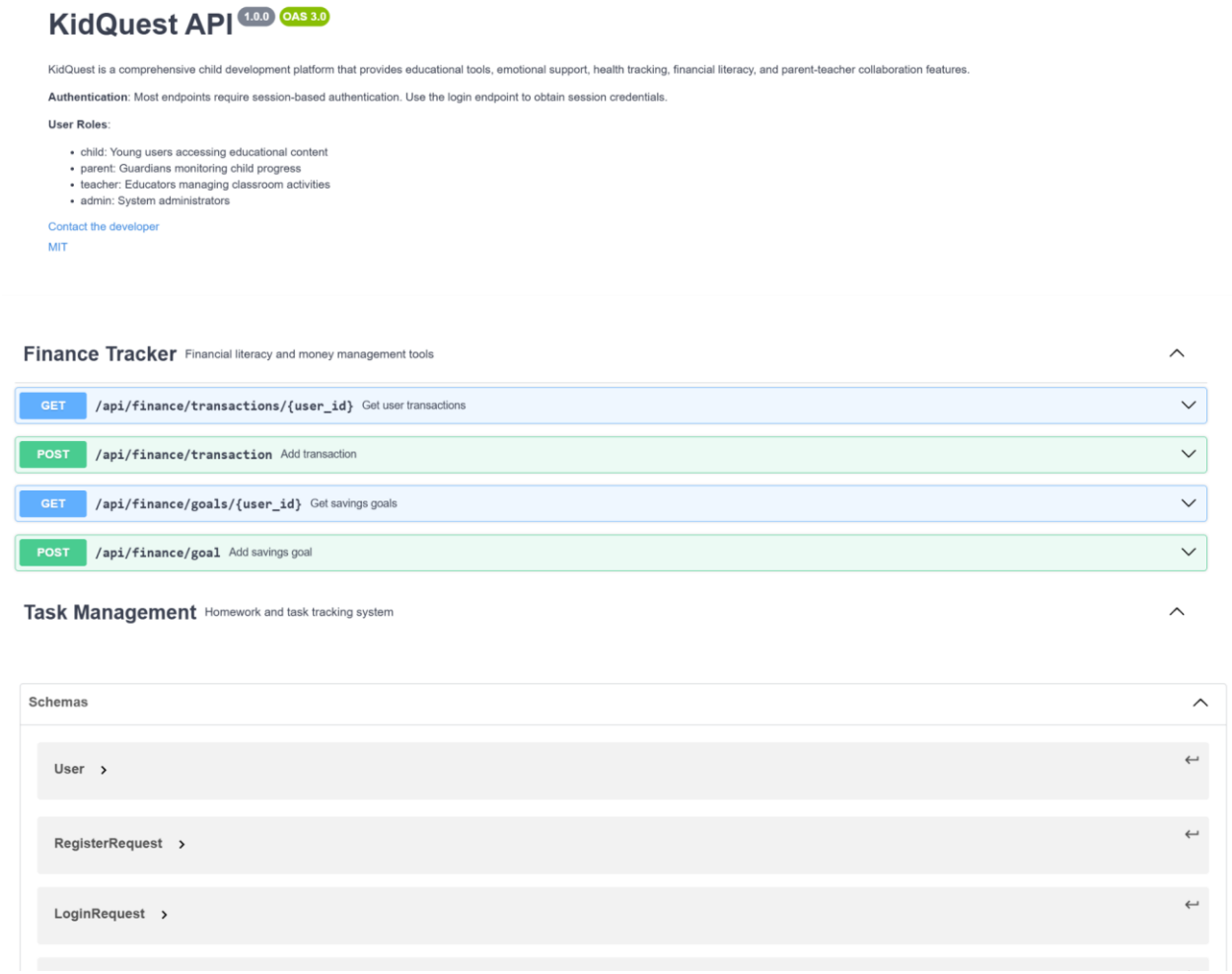
Demographics and System

- GET /api/admin/demographics — Aggregated demographics distributions and percentages.
- GET /api/health — System health check with DB connectivity and timestamp.

Testing Utilities

- DELETE /api/test/clear-user-data/{user_id} — Development-only endpoint to clear all data for a user (self-only).

5. YAML (Original YAML File)



Milestone - 5

Testing

6. KidQuest Platform – Pytest Test Suite Summary

1) Authentication & Core System

- test_user_authentication
    - Purpose: Verifies successful login and JWT issuance.
    - Input: POST /api/auth/login with {username, password}.
    - Expected: 200; success=true; access_token present.

2) Admin User Management

- test_create_parent_user
    - Purpose: Ensures admins can create parent users.
    - Input: POST /api/admin/users with username, email, password, role=parent.
    - Expected: 201; success=true; user.role=parent.
- test_security_validations
    - Purpose: Blocks creation of admin users via admin API (policy).
    - Input: POST /api/admin/users with role=admin.
    - Expected: 400; success=false; error mentions “admin”.

3) Teacher Dashboard APIs

- test_get_teacher_students_success
    - Purpose: Lists students assigned to a teacher.
    - Input: GET /api/teacher/students/{teacher_id}, Bearer token.
    - Expected: 200; success=true; students array includes assigned IDs.
- test_assign_homework_invalid_student
    - Purpose: Rejects assigning homework to unknown students.
    - Input: POST /api/teacher/assign-homework with student_id=999.
    - Expected: 404; success=false.
- test_assign_homework_unicode_character_handling
    - Purpose: Validates Unicode handling in subject/task payloads.
    - Input: POST /api/teacher/assign-homework with non-ASCII content.
    - Expected: 200; success=true (observed: 400 UnicodeDecodeError – needs fix).

- test_get_teacher_homework_sql_injection_attempt
    - Purpose: Guards against SQL injection via path parameter.
    - Input: GET /api/teacher/homework/1'; DROP TABLE homework; --
    - Expected: 400; error indicating invalid ID (observed: 200 – security gap).
- test_get_student_tasks_cross_teacher_data_leakage
    - Purpose: Ensures teacher isolation (no cross-account access).
    - Input: GET /api/teacher/student-tasks/{teacher_id} with other teacher's token.
    - Expected: 403; success=false (observed: 200 – authorization bug).

4) Parent Dashboard APIs

- test_get_tasks_for_parent_success
    - Purpose: Allows parent to view linked child's tasks.
    - Input: GET /api/tasks-for-parent/{child_id} with valid relationship.
    - Expected: 200; success=true; list of tasks.
- test_get_tasks_unauthorized_role
    - Purpose: Blocks non-parent role from parent-specific endpoint.
    - Input: GET /api/tasks-for-parent/{child_id} with child token.
    - Expected: 403; success=false; error indicates parent role required.
- test_get_tasks_no_relationship
    - Purpose: Enforces parent-child relationship.
    - Input: GET /api/tasks-for-parent/{child_id} without link.
    - Expected: 403; success=false; error cites missing relationship.
- test_get_mood_summary_success
    - Purpose: Returns mood summary with existing chat data.
    - Input: GET /api/chat/mood-summary/{child_id}; LLM mocked.
    - Expected: 200; success=true; latest_mood populated; sensible overall_mood.
- test_get_mood_summary_no_sessions
    - Purpose: Handles absence of chat data gracefully.
    - Input: GET /api/chat/mood-summary/{child_id} with no sessions.
    - Expected: 200; success=true; overall_mood=null; mood_tags=[].
- test_get_mood_summary_llm_failure
    - Purpose: Uses fallback text when LLM fails.
    - Input: GET /api/chat/mood-summary/{child_id}; LLM exception mocked.
    - Expected: 200; success=true; fallback overall_mood message; latest_mood from data.

5) Task Tracker System

- test_create_task_success

- Purpose: Creates a task with valid payload.
- Input: POST /api/homework/create with title, description, due_date, user_id.
- Expected: 201; success=true; task returned.

- test_create_task_invalid_date
  - Purpose: Validates due_date format.
  - Input: POST /api/homework/create with invalid due_date.
  - Expected: 400; success=false; error mentions date format.

6) LLM Chat Session System

- test_send_message_new_session
  - Purpose: Starts a new chat session and receives reply.
  - Input: POST /api/chat with message and user_id.
  - Expected: 200; success=true; session_id; response; timestamp.
- test_chat_rate_limiting
  - Purpose: Ensures rate limiting on rapid chat requests.
  - Input: 10 rapid POST /api/chat requests.
  - Expected: Some 429 responses (observed: none – rate limiting missing).

7) Doodling & Drawing APIs

- test_save_drawing_success
  - Purpose: Saves a base64-encoded drawing and metadata.
  - Input: POST /api/drawings/save with user_id, image_data, description, time_taken.
  - Expected: 200; success=true; drawing_id; filename.
- test_save_drawing_missing_data
  - Purpose: Validates mandatory image input.
  - Input: POST /api/drawings/save without image_data.
  - Expected: 400; success=false; error mentions missing required field.

8) Psychometry Assessment

- test_submit_assessment_success
  - Purpose: Submits a full assessment result successfully.
  - Input: POST /api/psychometry/submit with essential result fields.
  - Expected: 201; success=true; result_id.
- test_get_results_invalid_id
  - Purpose: Validates child_id format on retrieval.
  - Input: GET /api/psychometry/results/invalid_id.
  - Expected: 404; success=false; error mentions invalid.

- test_submit_assessment_complete_data
    - Purpose: Accepts extended fields including detailed_scores.
    - Input: POST /api/psychometry/submit with full dataset.
    - Expected: 201; success=true; result_id; confirmation message.
- test_submit_assessment_service_error
    - Purpose: Handles upstream service failure robustly.
    - Input: POST /api/psychometry/submit with mocked exception.
    - Expected: 500; error="Failed to submit answer"; message carries cause.

9) Notifications System

- test_get_notifications_success
    - Purpose: Fetches notifications for a user.
    - Input: GET /api/notifications/{user_id}.
    - Expected: 200; list of notifications.
- test_mark_read_unauthorized
    - Purpose: Prevents marking another user's notifications as read.
    - Input: POST /api/notifications/mark-read with foreign notification_id.
    - Expected: 403; success=false; unauthorized error.

10) Finance Module

- test_add_transaction_success
    - Purpose: Records a transaction for the authenticated user.
    - Input: POST /api/finance/transaction with user_id, amount, type, description.
    - Expected: 201; success=true; transaction payload.
- test_unauthorized_transaction
    - Purpose: Prevents creating a transaction for a different user.
    - Input: POST /api/finance/transaction with mismatched user_id.
    - Expected: 403; error="Unauthorized".

11) Health Tracker APIs

- test_toggle_task_success
    - Purpose: Toggles completion of a valid health task.
    - Input: POST /api/health/tasks/{task_id}/toggle.
    - Expected: 200; success=true; completed boolean.
- test_toggle_nonexistent_task
    - Purpose: Errors on toggling non-existent task.
    - Input: POST /api/health/tasks/99999/toggle.
    - Expected: 404; success=false; not found.

12) Additional Core APIs

- test_get_special_achievements_with_data
    - Purpose: Returns computed special achievements.
    - Input: GET /api/achievements/special/{user_id} with seeded data.
    - Expected: 200; success=true; achievements list populated.

13) Module Progress Tracking

- test_get_module_progress_with_data
    - Purpose: Retrieves module progress for a user with seeded progress.
    - Input: GET /api/progress/modules/{user_id}.
    - Expected: 200; success=true; progress entries with completion_percentage.

14) Login Streak Management

- test_get_login_streak_existing_user
    - Purpose: Returns current streak metrics for user with seeded streak.
    - Input: GET /api/login-streak/{user_id}.
    - Expected: 200; success=true; current_streak, total_logins, longest_streak.

15) Motivational Quotes

- test_get_quote_api_failure_fallback
    - Purpose: Uses fallback quote when external API fails.
    - Input: GET /api/quote/{user_id}; external API mocked down.
    - Expected: 200; success=false; fallback quote string.

16) Pomodoro Timer

- test_start_pomodoro_session_success
    - Purpose: Starts a Pomodoro session successfully.
    - Input: POST /api/pomodoro/start with user_id, task_name, duration_minutes.
    - Expected: 201; success=true; session_id; duration echoed.

17) Child Dashboard Statistics

- test_get_child_stats_with_achievements
    - Purpose: Aggregates dashboard KPIs including quests and stars.
    - Input: GET /api/child/stats/{user_id} with achievements seeded.
    - Expected: 200; success=true; stats with expected counts.

Sample Tests

👨‍👩‍👧 Parent Dashboard APIs

Module Overview

Test File: `test_parent_dashboard.py`
APIs Tested: `/api/tasks-for-parent/{child_id}`, `/api/chat/mood-summary/{child_id}`
Authentication: JWT Bearer Token Required
Authorization: Parent role and parent-child relationship validation

Test Case: Get Tasks for Parent - Success

API: `GET /api/tasks-for-parent/{child_id}`

Inputs:

- Child ID: 1
- Authorization: Bearer {parent_jwt_token}
- Database Setup: Valid parent-child relationship

Expected Output:

```
{
  "success": true,
  "tasks": [
    {
      "subject": "Science",
      "task": "Read chapter 5",
      "status": "pending",
      "user_id": "1"
    }
  ]
}
```

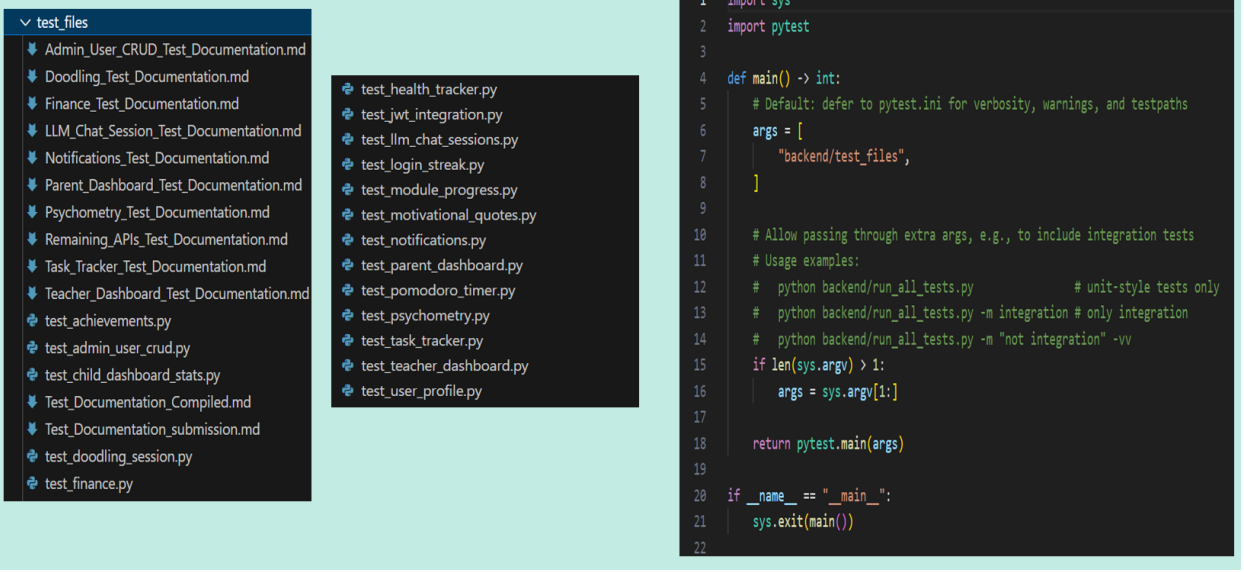
Pytest Code:

```
def test_get_tasks_for_parent_success():
    client = app.test_client()
```

```
    # Setup parent user and relationship
    parent_user = User(
        username='test_parent',
        email='parent@example.com',
        password_hash='hashed_password',
        role='parent'
    )
    db.session.add(parent_user)
    db.session.commit()

    # Create parent-child relationship
    relationship = ParentChild(
        parent_id=parent_user.id,
        child_id=1
    )
    db.session.add(relationship)

    # Create homework task for child
    task = HomeworkSchedule(
        user_id=1,
        subject='Science',
        task='Read chapter 5',
        status='pending'
    )
    db.session.add(task)
    db.session.commit()

    # Generate parent JWT token
    parent_token = create_access_token(identity=str(parent_user.id))

    headers = {
        "Authorization": f"Bearer {parent_token}"
    }

    response = client.get('/api/tasks-for-parent/1', headers=headers)

    assert response.status_code == 200
    response_data = response.get_json()
    assert response_data['success'] == True
    assert len(response_data['tasks']) >= 1
    assert response_data['tasks'][0]['subject'] == 'Science'
```

Result: Success ✅

Testing


test_files
Admin_User_CRUD_Test_Documentation.md
Doodling_Test_Documentation.md
Finance_Test_Documentation.md
LLM_Chat_Session_Test_Documentation.md
Notifications_Test_Documentation.md
Parent_Dashboard_Test_Documentation.md
Psychometry_Test_Documentation.md
Remaining_APIs_Test_Documentation.md
Task_Tracker_Test_Documentation.md
Teacher_Dashboard_Test_Documentation.md
test_achievements.py
test_admin_user_crud.py
test_child_dashboard_stats.py
Test_Documentation_Compiled.md
Test_Documentation_submission.md
test_doodling_session.py
test_finance.py
test_health_tracker.py
test_jwt_integration.py
test_llm_chat_sessions.py
test_login_streak.py
test_module_progress.py
test_motivational_quotes.py
test_notifications.py
test_parent_dashboard.py
test_pomodoro_timer.py
test_psychometry.py
test_task_tracker.py
test_teacher_dashboard.py
test_user_profile.py
backend > run_all_tests.py > ...
import sys
import pytest
def main() -> int:
# Default: defer to pytest.ini for verbosity, warnings, and testpaths
args = [
"backend/test_files",
]
# Allow passing through extra args, e.g., to include integration tests
# Usage examples:
# python backend/run_all_tests.py # unit-style tests only
# python backend/run_all_tests.py -m integration # only integration
# python backend/run_all_tests.py -m "not integration" -vv
if len(sys.argv) > 1:
args = sys.argv[1:]
return pytest.main(args)
if __name__ == "__main__":
sys.exit(main())

Milestone - 6

Steps to run app | Tech Stack | Issues & Pull Requests

7. How to run the Application?

Prerequisites

- Python 3.10+ installed
- Node.js and npm installed

1) Run the Backend (Flask)

1. Open a terminal and navigate to the backend directory:
    - cd backend
2. (Recommended) Create and activate a virtual environment:
    - python -m venv venv
    - On Windows: venv\Scripts\activate
    - On macOS/Linux: source venv/bin/activate
3. Install Python dependencies:
    - pip install -r requirements.txt
4. Start the Flask server:
    - python app.py
5. Notes:
    - The app initializes the database in backend/instance/app.db on first run.
    - Default demo credentials (as per README):
        - Admin: admin / admin123
        - Child: child / childchild
        - Parent: parent / parentparent
        - Teacher: teacher / teacherteacher

2) Run the Frontend (Vue.js)

1. Open a new terminal and navigate to the frontend directory:
    - cd frontend
2. Install Node dependencies:
    - npm install
3. Start the development server:

    - npm run dev
4. Open the provided local URL in a browser (typically http://localhost:5173).

3) End-to-End Usage

- Ensure the backend (Flask) is running on http://localhost:5000.
- Ensure the frontend (Vite dev server) is running (e.g., http://localhost:5173).
- Log in using one of the demo credentials to explore child/parent/teacher/admin flows.

8. Tech Stack Used

Frontend

- Framework: Vue.js (SPA with role-specific dashboards for child, parent, admin)
- UI/UX: Child-friendly components, animations, interactive modules (Pomodoro, Doodling Canvas, Finance/Health trackers)
- Auth: JWT Bearer tokens in Authorization headers
- Data Viz: Child-friendly charts/graphs for progress and trends

Backend

- Runtime/Language: Python 3.10+
- Web Framework: Flask
- CORS: Flask-CORS for secure cross-origin frontend-backend communication
- Authentication/Authorization: flask-jwt-extended (JWT issuance, verification, role-based access)
- Password Security: werkzeug.security (hashing, verification)
- Session Management: Flask session (used for psychometry flow state)
- Input Validation: Regex-based checks; endpoint-level validation

AI/LLM Integration

- Providers: ChatGroq-compatible APIs and OpenRouter APIs
- Use Cases:
    - Emotion Chatbot with prompt-engineering and mood tagging
    - Psychometry Engine for adaptive questions, scoring, and feedback
- Client: openai Python client pointed at Groq/OpenRouter endpoints

Database & ORM

- ORM: Flask-SQLAlchemy

- DB Engine: SQLite (development)
- Models: User, ChildProfile, ParentChild, HomeworkSchedule, PomodoroSession, Transaction, SavingGoal, HealthTask, HealthStreak, WaterLog, Achievement, ChatSession, LLMInteractions, Notification, LoginStreak, LoginHistory, PsychometricTestResult, UserModuleProgress

APIs & Services

- REST Endpoints: Modular routes for Auth, Chat, Tasks, Pomodoro, Finance, Health, Achievements, Stories, Doodles, Psychometry, Notifications, Admin Analytics
- External API: ZenQuotes for motivational quotes (with graceful fallback)
- File Handling: Base64 image handling and filesystem storage for Doodling

Security & Safety

- JWT error handlers: Expired/invalid/missing/revoked token responses
- RBAC: Role-protected endpoints for child/parent/teacher/admin
- Prompt Sanitization: Safe AI responses
- Secret Management: Config-driven keys (e.g., secrets.token_hex for secret keys)
- Data Integrity: Parent-child relationships, foreign key constraints

Analytics & Admin

- Aggregations: Dashboard stats, task analytics, user activity metrics, demographics
- Historical Tracking: LoginHistory and streak-based metrics
- Admin Tools: User CRUD (admin-only), cascade deletes, migration utilities

Dev & Testing

- Testing Framework: Pytest
- Test Patterns: In-memory SQLite, fixtures, JWT setup, mocking external services
- Dev Server: Flask debug mode on localhost:5000
- Static/Media: Local static folder for reference images and saved drawings

Utilities & Libraries

- Standard Libraries: datetime, time, os, glob, base64, json, random, traceback, re
- Imaging (optional fallback): Pillow (PIL) for generating placeholder reference images

Configuration

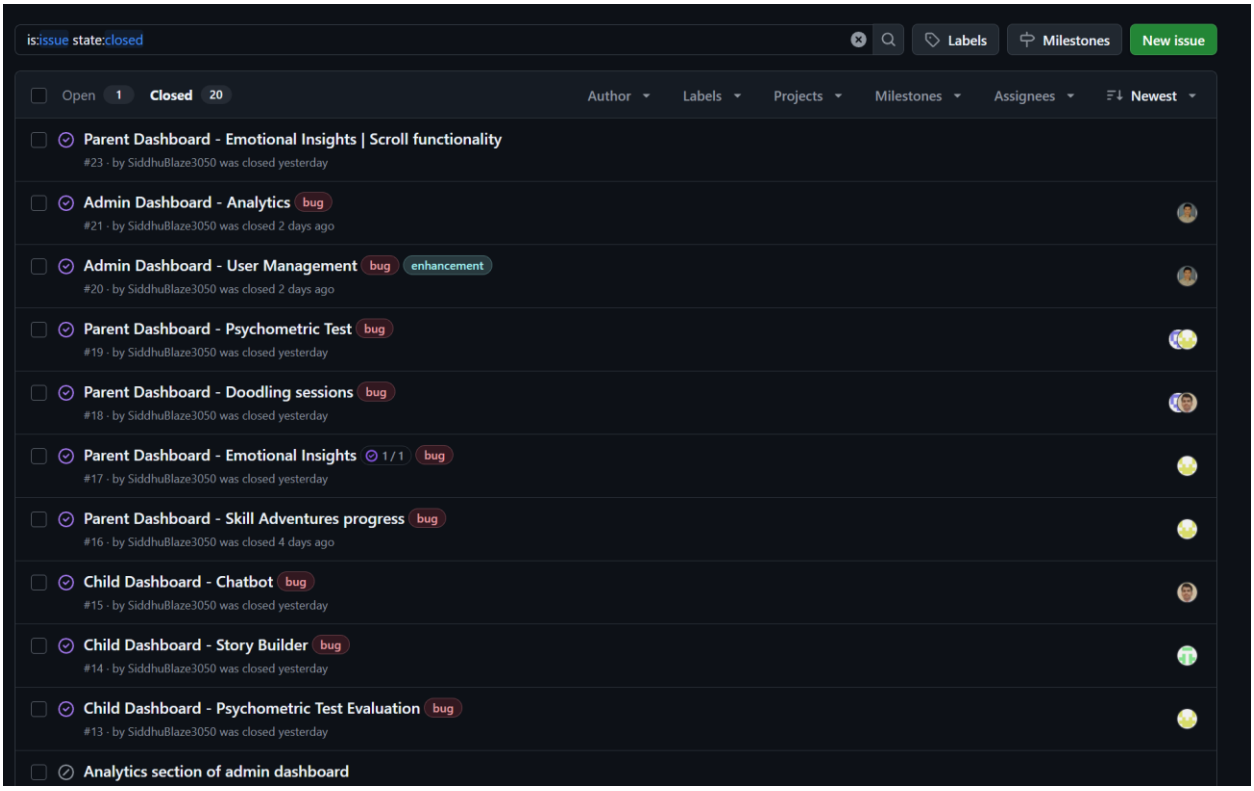
- Environment-driven: API keys (Groq/OpenRouter), base URLs, secret keys via Config
- Instance Directory: Created on startup for persistence when needed

9. Issue Reporting & Pull Requests

Issues summary:

- Focus areas: Most issues center on Parent Dashboard (Emotional Insights, Psychometric Test, Doodling Sessions, Skill Adventures progress) and Child Dashboard (Chatbot, Story Builder, Psychometric Test Evaluation), plus Admin Dashboard (Analytics, User Management).
- Types: Predominantly bug fixes, with at least one enhancement (Admin Dashboard – User Management).
- Status: 20 closed issues are visible; items show recent closure (yesterday to a few days ago), indicating active triage and resolution.
- Parent Dashboard:
    - Emotional Insights: Multiple entries, including a scroll functionality bug.
    - Psychometric Test: Stability/UX fixes.
    - Doodling Sessions: Reliability or data display issues.
    - Skill Adventures progress: Tracking/progress calculation issues.
- Child Dashboard:
    - Chatbot: Bug affecting interaction or responses.
    - Story Builder: Content creation/edit flow issues.
    - Psychometric Test Evaluation: Result handling/accuracy fixes.

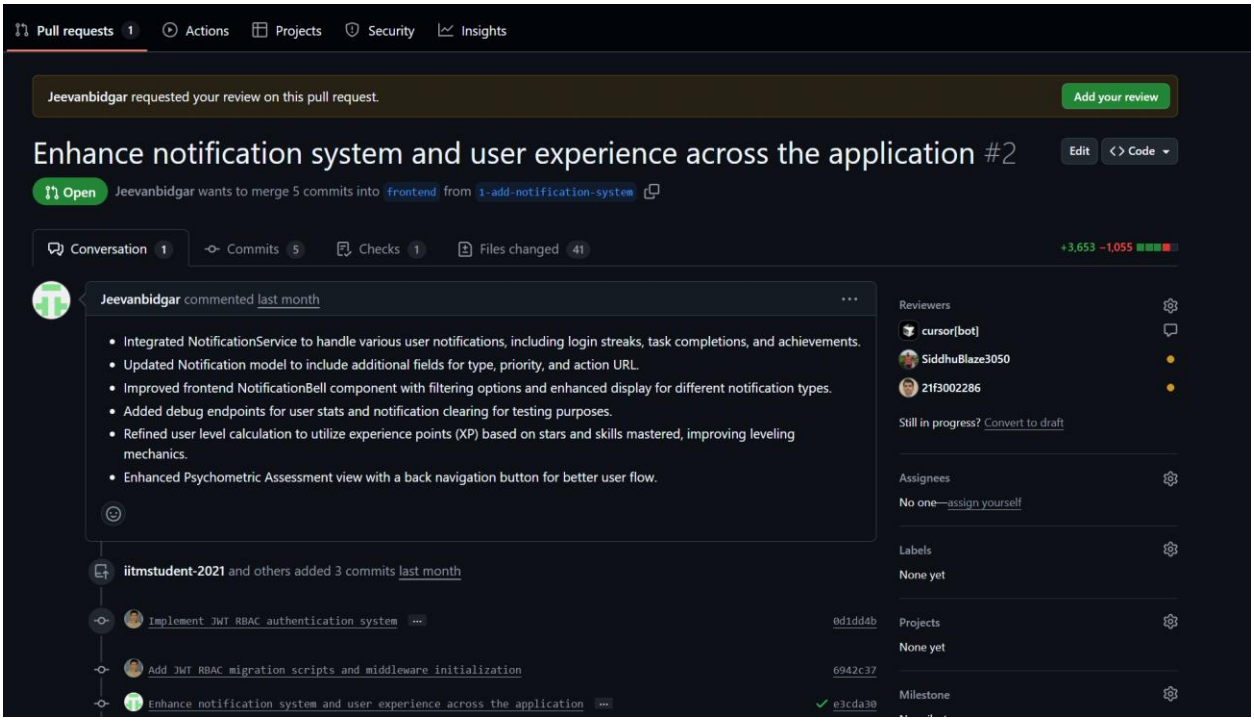
- Admin Dashboard:
    - Analytics: Data aggregation/visualization bugs.
    - User Management: Marked enhancement—feature improvements rather than defects.



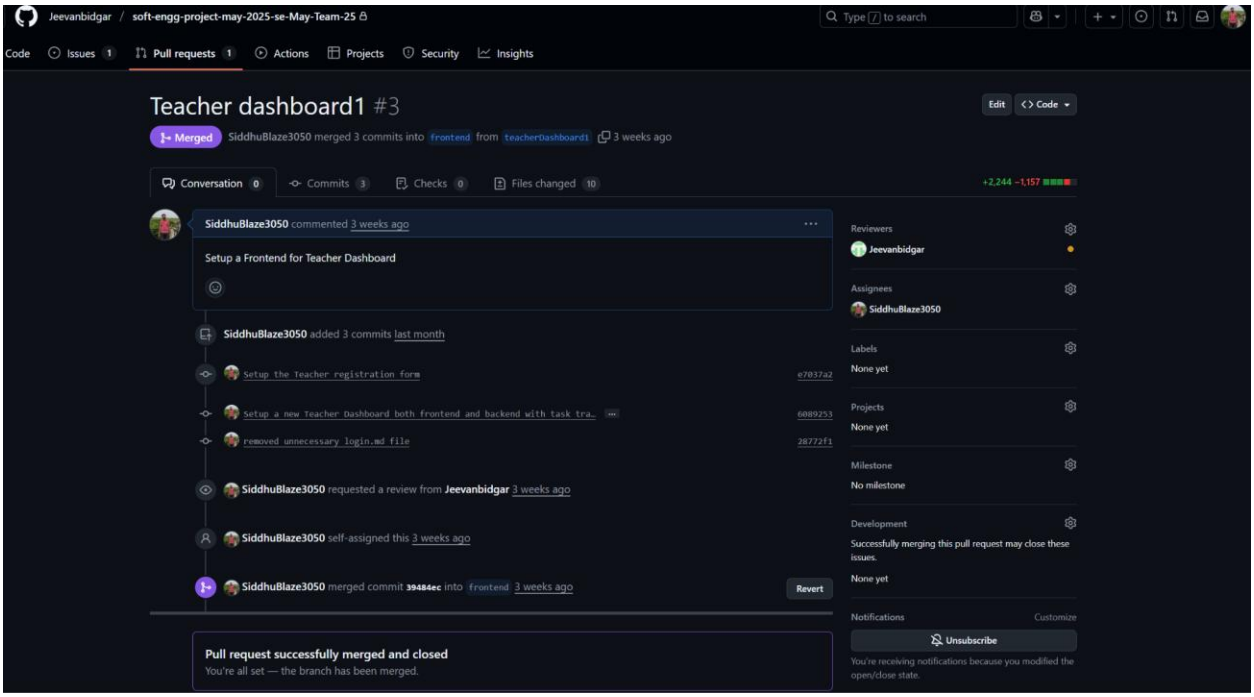
Pull Requests summary:

1. Enhance notification system and user experience across the application (Closed)
    - Objective: Introduce a unified NotificationService and elevate UX across notifications and learning flows.
    - Key changes:
        - Backend: Extended Notification model (type, priority, action URL), added debug endpoints for user stats and notification clearing.
        - Frontend: Improved NotificationBell with filtering and clearer displays for different notification types.
        - Engagement: Refined user leveling to use XP derived from stars earned and skills mastered.

- UX polish: Enhanced Psychometric Assessment view with a back-navigation button for smoother flow.
- Activity: 5 commits; 41 files changed; reviewers listed; checks present; proposed merge into frontend from feature branch; currently open.



2. Teacher dashboard1 (Merged)
    - Objective: Deliver the initial Teacher Dashboard and related flows.
    - Key changes:
        - Teacher registration form setup.
        - New Teacher Dashboard integrating frontend and backend with task tracking.
        - Cleanup of unnecessary login-related file.
    - Status: Successfully merged into frontend; branch closed after merge.
    - Activity: 3 commits; 10 files changed; review requested and completed about three weeks ago.


Jeevanbidgar / soft-engg-project-may-2025-se-May-Team-25
Type / to search
Code
Issues 1
Pull requests 1
Actions
Projects
Security
Insights
Teacher dashboard1 #3
Edit
Code
Merged
SiddhuBlaze3050 merged 3 commits into frontend from teacherDashboard1 3 weeks ago
Conversation 0
Commits 3
Checks 0
Files changed 10
+2,244 −1,157
SiddhuBlaze3050 commented 3 weeks ago
Setup a Frontend for Teacher Dashboard
SiddhuBlaze3050 added 3 commits last month
Setup the Teacher registration form
e7037a2
Setup a new Teacher Dashboard both frontend and backend with task tra…
6089253
removed unnecessary login.md file
28772f1
SiddhuBlaze3050 requested a review from Jeevanbidgar 3 weeks ago
SiddhuBlaze3050 self-assigned this 3 weeks ago
SiddhuBlaze3050 merged commit 39484ec into frontend 3 weeks ago
Revert
Pull request successfully merged and closed
You're all set — the branch has been merged.
Reviewers
Jeevanbidgar
Assignees
SiddhuBlaze3050
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.