**Logo, company name

Description automatically generated**

**COMSATS University Islamabad (CUI)**

Software Design Description   
(SDS DOCUMENT)

for

**BrainBee**

Version 1.0

***By***

**Nasir Shahzad CUI/SP22-BCS-077/ISB**

**Sohail Faiz Rasool CUI/SP22-BCS-095/ISB**

***Supervisor*Mr. Tanveer Ahmed**

*Bachelor of Science in Computer Science (2022-2026)*

**1. Introduction**

The BrainBee project aims to develop a mobile-based, AI-powered learning platform tailored for Pakistani students in grades 5–12. The system is designed to address gaps in accessibility, engagement, and personalized support in the education sector by leveraging artificial intelligence and gamification. The goal is to make high-quality education affordable, engaging, and effective for students in urban and rural regions alike.

**1.1 Modules**

This subsection lists the modules of BrainBee, along with their functionalities.

**1.1.1 Student App**

**1.1.1.1 User Management**

Manages user registration, profile updates, account deletion, grade selection, subject preferences, and premium upgrades.

**1.1.1.2 Peer Competition Module**

Enables students to participate in AI-generated quiz battles, track performance, maintain battle history, and view leaderboard positions.

**1.1.1.3 Learning Module**

Provides AI-generated summaries and explanations, interactive chatbot support, and personalized flashcards.

**1.1.1.4 Assessment Module**

Generates personalized quizzes, enables quiz and assignment submission, provides instant feedback, and maintains report cards and leaderboard metrics.

**1.1.1.5 Achievement and Rewards Module**

Tracks and displays badges, certificates, redeemable coins, and coin quests to gamify the learning experience.

**1.1.1.6 Performance Analytics Module**

Offers detailed performance reports across books, report cards, and leaderboard standings over time.

**1.1.1.7 AI Powered Assistant Module**

Allows students to ask questions and receive AI-generated answers with chat history maintenance.

**1.1.2 Parent App**

**1.1.2.1 User Management**

Manages parent user accounts including registration, profile updates, and deletion.

**1.1.2.2 Monitor Children**

Allows parents to add/remove children, monitor performance, view leaderboard rankings, and receive reports and notifications.

**1.1.2.3 Goals Module**

Enables parents to set, track, and reward academic goals for their children.

**1.1.2.4 Communication Module**

Facilitates chat with teachers and feedback submission to admins.

**1.1.3 Teacher App**

**1.1.3.1 User Management**

Enables account management for teachers, including registration, profile updates, and deletion.

**1.1.3.2 Class Management**

Allows creation and management of classes, students, study material, assignments, and quizzes.

**1.1.3.3 Assessment Module**

Provides performance monitoring and insights through ML algorithms, auto-grading of assignments, and personalized learning path generation.

**2. Design Methodology and Software Process Model**

This section outlines the chosen design methodology and software process model for the BrainBee educational application.

**2.1 Design Methodology**

BrainBee employs **Object-Oriented Programming (OOP)** as its design methodology due to the modular and scalable nature of the application. OOP provides a structured way to represent complex real-world entities such as students, parents, teachers, quizzes, rewards, and performance analytics through classes and objects. This approach ensures high code **reusability**, **maintainability**, and **extensibility**, which is essential for a system that involves multiple interacting modules and diverse user types.

Encapsulation and abstraction enable developers to isolate features like assessments, goal tracking, and peer competition, making the application more manageable and easier to debug. OOP's use of inheritance and polymorphism also supports future feature expansions without major codebase refactoring. This design methodology aligns well with BrainBee’s goal of building a flexible and robust platform that supports continuous improvement and evolving educational needs.

**2.2 Software Process Model**

BrainBee follows the **Agile Software Development** process model due to its iterative, collaborative, and adaptive nature. Agile is particularly well-suited for BrainBee because the project involves AI integration, dynamic feedback from students, teachers, and parents, and frequent improvements based on user behavior and evolving academic needs.

Agile allows the development team to work in **sprints**, delivering incremental updates with fully functional modules such as the Student Quiz Module, AI Chatbot, or Parent Monitoring Dashboard. Regular feedback from stakeholders ensures that any required changes are identified early and implemented quickly. This reduces risk, enhances product quality, and improves alignment with end-user expectations. Agile’s emphasis on working software, adaptability, and close collaboration makes it ideal for an education-focused application aiming for continuous improvement and user engagement.

This figure illustrates the agile development process model and its stages.

A diagram of a software development process

AI-generated content may be incorrect.

Figure 1 Agile Development Model

**3. System Overview**

This section provides a high-level summary of the overall system architecture and functionalities of **BrainBee**.

**3.1 Functionality**

**BrainBee** is an AI-powered mobile educational platform designed to enhance academic learning for students in grades 5 to 12 across Pakistan. It offers features such as user management for students, parents, and teachers; personalized learning content; AI-generated quizzes and explanations; gamified peer competition; performance analytics; and a reward system. The platform also provides tools for parental monitoring, goal setting, and teacher-based class and assignment management. Real-time feedback and AI tutoring ensure a customized, engaging, and efficient learning experience. The system promotes academic success through individualized learning paths, gamification, and collaborative support among users.

**3.2 Context**

Operating within the educational ecosystem of Pakistan, **BrainBee** addresses key challenges such as the lack of personalized instruction, rote learning, absence of student motivation, and limited parental and teacher engagement. The platform integrates artificial intelligence to personalize study material and assessments while encouraging healthy competition and progress tracking. By supporting students, parents, and teachers, BrainBee bridges the educational gap between urban elite and under-resourced communities, especially in Punjab. It aims to make quality education accessible, affordable, and interactive through AI and mobile technology.

**3.3 Design**

**BrainBee** utilizes a **multi-tiered architecture** that separates the system into distinct layers:

* **Presentation Layer** for user interfaces across mobile apps (Flutter-based)
* **Business Logic Layer** for managing core functionalities such as quizzes, performance tracking, and reward systems
* **Data Access Layer** for interacting with the MongoDB database securely

This design promotes **modularity**, **reusability**, and **scalability**, allowing different modules (e.g., student assessments, parent goals, teacher analytics) to work independently yet cohesively. RESTful APIs connect the layers to ensure seamless communication across different system components and user apps. The AI components are integrated through Python-based machine learning models that assist in generating personalized content, performance predictions, and quiz recommendations.

**3.4 Architectural Design**

**BrainBee** is designed with a modular architecture to support maintainability, extensibility, and integration of new features. The system is divided into key modules:

* **User Management:** Handles registration, profile management, and user type switching
* **Assessment Module:** Manages AI-generated quizzes, assignments, feedback, and report cards
* **Learning Module:** Provides summaries, flashcards, and AI chatbot explanations
* **Peer Competition Module:** Enables quiz battles, leaderboard tracking, and win/loss history
* **Performance Analytics:** Tracks detailed academic data and AI-based progress insights
* **Parent & Teacher Modules:** Allow monitoring, goal setting, and personalized teaching interventions
* **Reward Module:** Handles badges, coins, certificates, and redemption tracking

This modular approach ensures that updates or changes in one part of the system do not affect others. The **multi-tiered architectural style** supports separation of concerns and eases testing, deployment, and scaling.

**5. Data Design**

This section outlines the structure and organization of the data used in **BrainBee**, detailing the schema, data models, and relationships essential for efficient data management and retrieval.

**5.1 Data Dictionary**

These tables show the data dictionary used in the **BrainBee** system.

**User (Common Base for Student, Parent, Teacher):**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| userID | String | Unique user identifier |
| fullName | String | Full name of the user |
| email | String | Registered email address |
| phoneNumber | String | User's phone number |
| passwordHash | String | Hashed password |
| role | Enum | Student, Parent, or Teacher |
| accountType | Enum | Standard or Premium |
| profilePicture | String | URL to profile image |
| registrationDate | DateTime | Account creation timestamp |
| isActive | Boolean | Account status |

**Student (Extends User):**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| gradeLevel | String | Current grade (e.g., 9th) |
| enrolledSubjects | Array | List of selected subjects |
| quizHistory | Array | Past quizzes taken |
| leaderboardRank | Number | Rank on leaderboard |
| streakScore | Number | Daily learning streak |

**Parent (Extends User)**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| childrenIDs | Array | List of studentIDs linked to the parent |
| goals | Array | Goals assigned to children |
| notifications | Array | Real-time updates (e.g., quiz complete) |

**Teacher (Extends User)**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| classIDs | Array | List of managed class IDs |
| assignedQuizzes | Array | Quizzes created for students |
| uploadedMaterials | Array | Uploaded notes or assignments |

**Quiz**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| quizID | String | Unique quiz identifier |
| createdBy | String | studentID or teacherID |
| subject | String | Subject of the quiz |
| topic | String | Topic or chapter |
| questions | Array | List of questions (with options, answers) |
| generatedByAI | Boolean | Indicates if AI generated it |
| attemptHistory | Array | List of studentIDs who attempted |
| dateCreated | DateTime | Creation date |

**Battle**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| battleID | String | Unique identifier |
| challengerID | String | studentID of initiator |
| opponentID | String | studentID of opponent |
| quizData | Object | Quiz used in battle |
| winnerID | String | ID of winner |
| battleDate | DateTime | Date of battle |
| durationSeconds | Number | Time taken |
| winLossRecord | Object | Result summary for both students |

**FlashCard**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| cardID | String | Unique ID for flashcard |
| subject | String | Subject name |
| topic | String | Chapter/topic name |
| content | String | AI-generated explanation |
| studentID | String | Owner of flashcard |
| createdAt | DateTime | Creation timestamp |

**Assignment**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| assignmentID | String | Unique ID |
| classID | String | Class it belongs to |
| uploadedBy | String | Teacher ID |
| description | String | Assignment details |
| deadline | DateTime | Due date |
| submissions | Array | List of studentIDs with files/status |

**Reward**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| rewardID | String | Unique reward identifier |
| type | Enum | Badge, Certificate, CoinQuest |
| name | String | Title of the reward |
| awardedTo | String | studentID |
| awardedBy | String | System or parentID |
| dateAwarded | DateTime | Timestamp |
| coinValue | Number | If redeemable coins |

**Goal**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| goalID | String | Unique goal ID |
| parentID | String | Assigned by this parent |
| studentID | String | For this student |
| description | String | Academic task or milestone |
| status | Enum | Pending, Completed, Failed |
| rewardCoins | Number | Coins to award upon completion |
| dueDate | DateTime | Goal deadline |

**ReportCard**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| reportCardID | String | Unique ID |
| studentID | String | Student owner |
| bookName | String | Book title (e.g., Chemistry 10th) |
| score | Number | Percentage or numeric score |
| weakAreas | Array | Topics needing improvement |
| evaluationDate | DateTime | Auto-generated on quiz completion |

**ChatHistory**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| chatID | String | Unique chat identifier |
| studentID | String | Associated student |
| messages | Array | List of Q&A messages |
| createdAt | DateTime | Chat timestamp |
| topic | String | Chapter or concept discussed |

**Class**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| classID | String | Unique identifier |
| teacherID | String | Managed by this teacher |
| className | String | Display name |
| studentList | Array | List of enrolled studentIDs |
| materials | Array | Lecture notes, PDFs, videos |

**5.2 Data Schema**

**5.2.1 User Schema (Base Schema)**

const UserSchema = new mongoose.Schema({

userID: { type: String, unique: true, required: true },

fullName: String,

email: { type: String, unique: true, required: true },

phoneNumber: String,

passwordHash: String,

role: { type: String, enum: ['Student', 'Parent', 'Teacher'], required: true },

accountType: { type: String, enum: ['Standard', 'Premium'], default: 'Standard' },

profilePicture: String,

registrationDate: { type: Date, default: Date.now },

isActive: { type: Boolean, default: true }

});

**5.2.2 Student Schema (Extends User)**

const StudentSchema = new mongoose.Schema({

userID: { type: String, ref: 'User' },

gradeLevel: String,

enrolledSubjects: [String],

quizHistory: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Quiz' }],

leaderboardRank: Number,

streakScore: Number

});

**5.2.3 Parent Schema (Extends User)**

const ParentSchema = new mongoose.Schema({

userID: { type: String, ref: 'User' },

childrenIDs: [{ type: String, ref: 'Student' }],

notifications: [String]

});

**5.2.4 Teacher Schema (Extends User)**

const TeacherSchema = new mongoose.Schema({

userID: { type: String, ref: 'User' },

classIDs: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Class' }],

assignedQuizzes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Quiz' }],

uploadedMaterials: [String]

});

**5.2.5 Quiz Schema**

const QuizSchema = new mongoose.Schema({

quizID: { type: String, unique: true },

createdBy: String,

subject: String,

topic: String,

questions: [

{

questionText: String,

options: [String],

correctAnswer: String

}

],

generatedByAI: { type: Boolean, default: true },

attemptHistory: [{ studentID: String, score: Number, attemptedAt: Date }],

dateCreated: { type: Date, default: Date.now }

});

**6. Battle Schema**

const BattleSchema = new mongoose.Schema({

battleID: { type: String, unique: true },

challengerID: String,

opponentID: String,

quizData: { type: Object }, // Could embed quiz structure

winnerID: String,

battleDate: Date,

durationSeconds: Number,

winLossRecord: {

[String]: { type: String, enum: ['Win', 'Loss', 'Draw'] }

}

});

**7. FlashCard Schema**

const FlashCardSchema = new mongoose.Schema({

cardID: { type: String, unique: true },

studentID: String,

subject: String,

topic: String,

content: String,

createdAt: { type: Date, default: Date.now }

});

**8. Assignment Schema**

const AssignmentSchema = new mongoose.Schema({

assignmentID: { type: String, unique: true },

classID: String,

uploadedBy: String,

description: String,

deadline: Date,

submissions: [{

studentID: String,

fileURL: String,

status: { type: String, enum: ['Pending', 'Submitted', 'Graded'] }

}]

});

**9. Reward Schema**

const RewardSchema = new mongoose.Schema({

rewardID: { type: String, unique: true },

type: { type: String, enum: ['Badge', 'Certificate', 'CoinQuest'] },

name: String,

awardedTo: String,

awardedBy: String,

dateAwarded: { type: Date, default: Date.now },

coinValue: Number

});

**10. Goal Schema**

const GoalSchema = new mongoose.Schema({

goalID: { type: String, unique: true },

parentID: String,

studentID: String,

description: String,

status: { type: String, enum: ['Pending', 'Completed', 'Failed'], default: 'Pending' },

rewardCoins: Number,

dueDate: Date

});

**11. Report Card Schema**

const ReportCardSchema = new mongoose.Schema({

reportCardID: { type: String, unique: true },

studentID: String,

bookName: String,

score: Number,

weakAreas: [String],

evaluationDate: { type: Date, default: Date.now }

});

**12. Chat History Schema**

const ChatHistorySchema = new mongoose.Schema({

chatID: { type: String, unique: true },

studentID: String,

topic: String,

messages: [{

sender: { type: String, enum: ['student', 'AI'] },

text: String,

timestamp: { type: Date, default: Date.now }

}],

createdAt: { type: Date, default: Date.now }

});

**13. Class Schema**

const ClassSchema = new mongoose.Schema({

classID: { type: String, unique: true },

teacherID: String,

className: String,

studentList: [{ type: String, ref: 'Student' }],

materials: [{

title: String,

fileURL: String,

uploadedAt: Date

}]

});

**6. Human Interface Design**

This section outlines the user interface design principles and components for **BrainBee**, ensuring an intuitive and seamless experience for both mobile app users and admin dashboard operators.

**6.1 Screen Images**

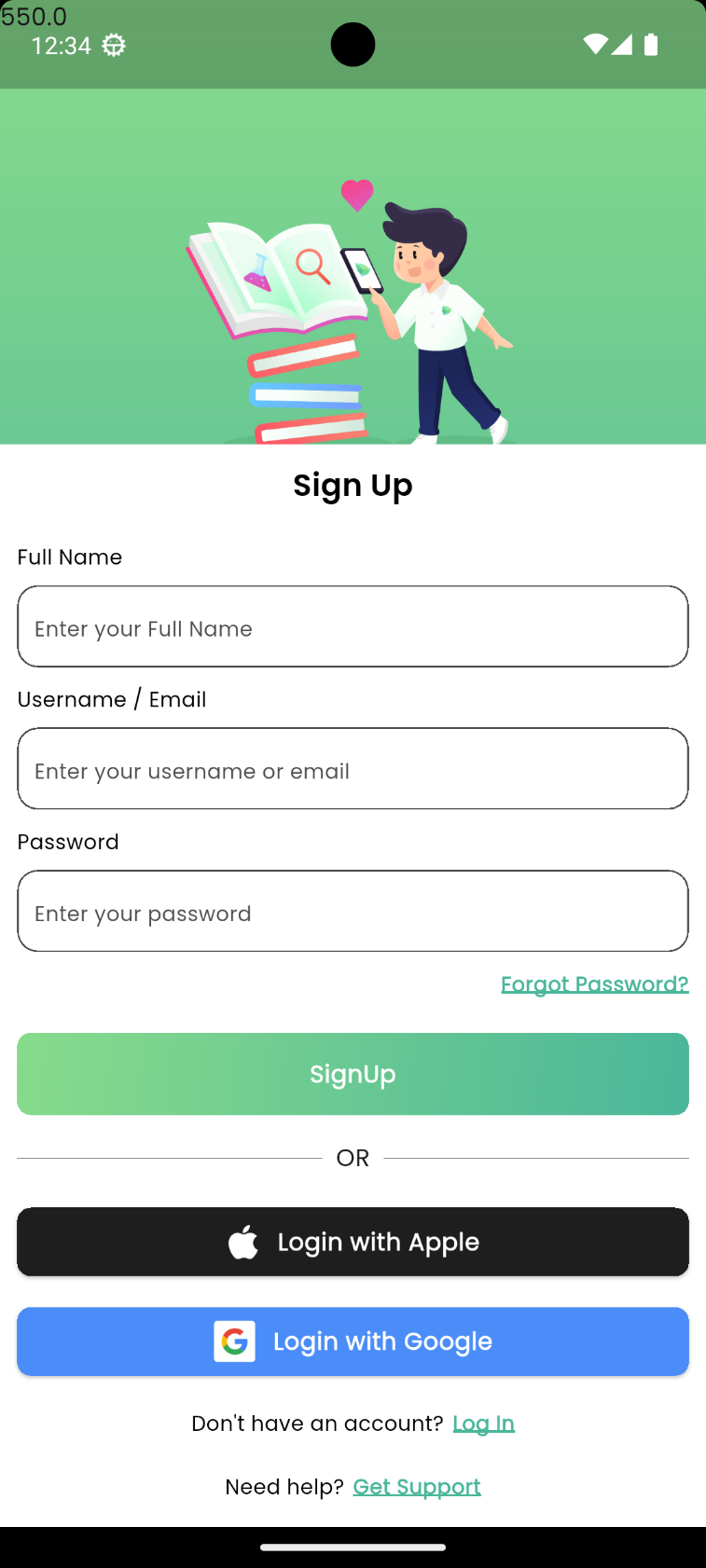
This subsection presents the screen images showing the user interface for the SafarRehnuma platform.

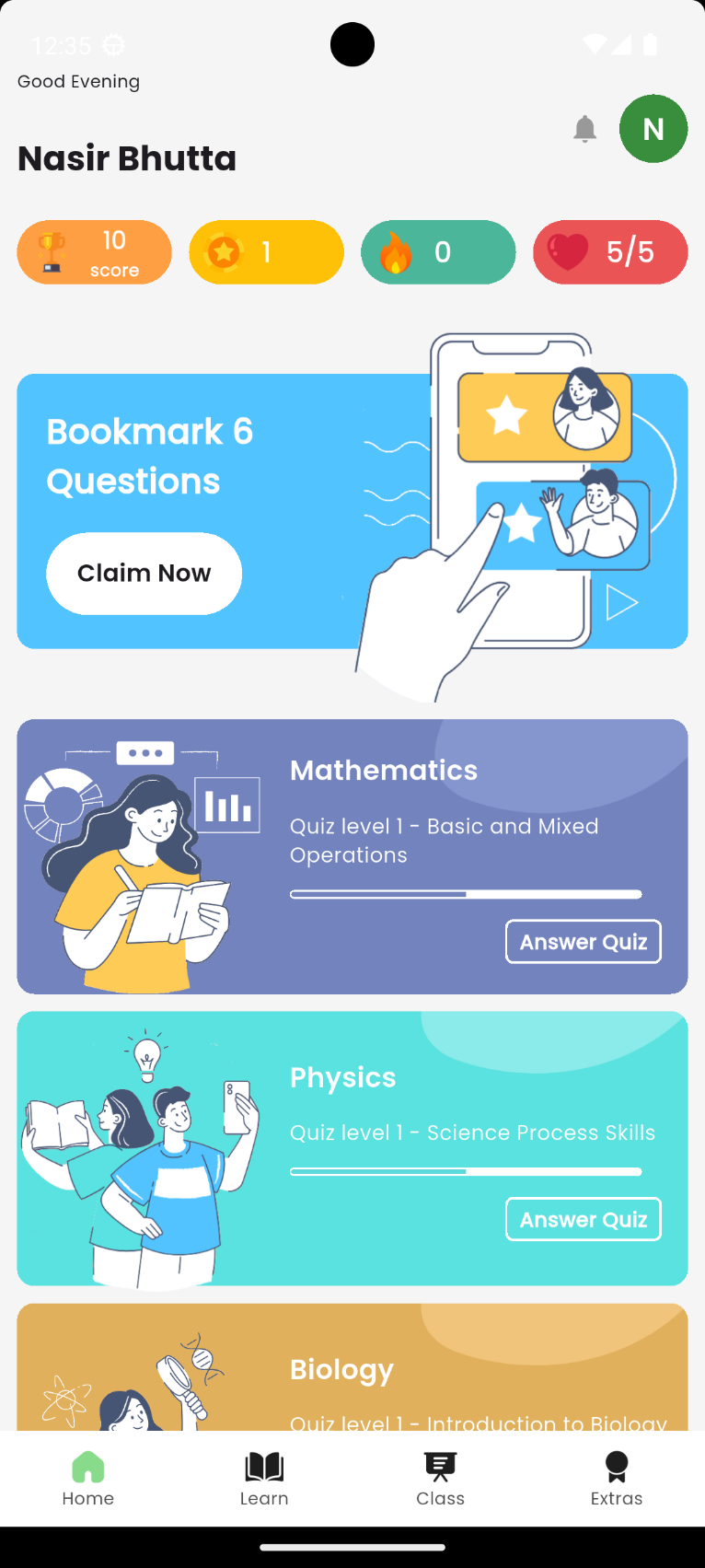
**Login screen**

**A screenshot of a login form

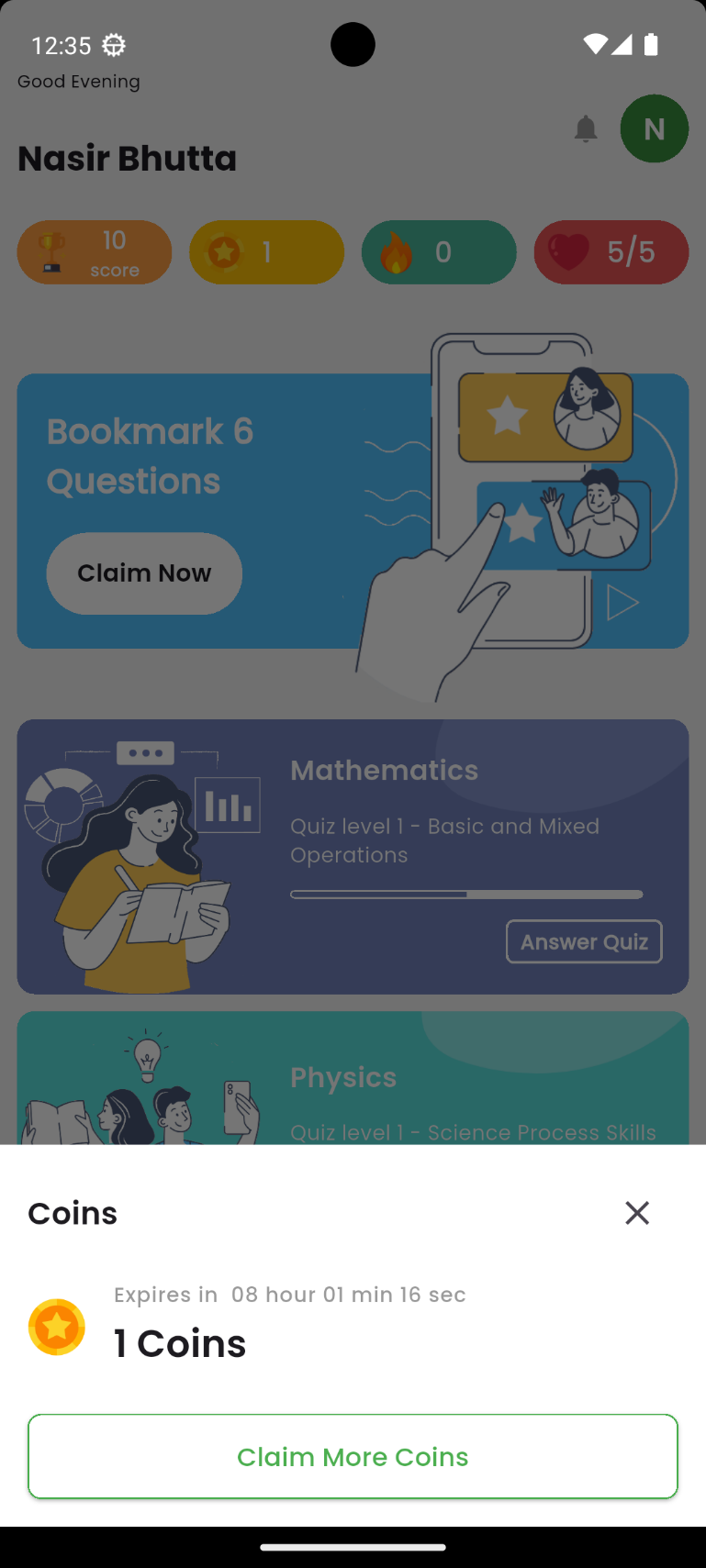
AI-generated content may be incorrect.**

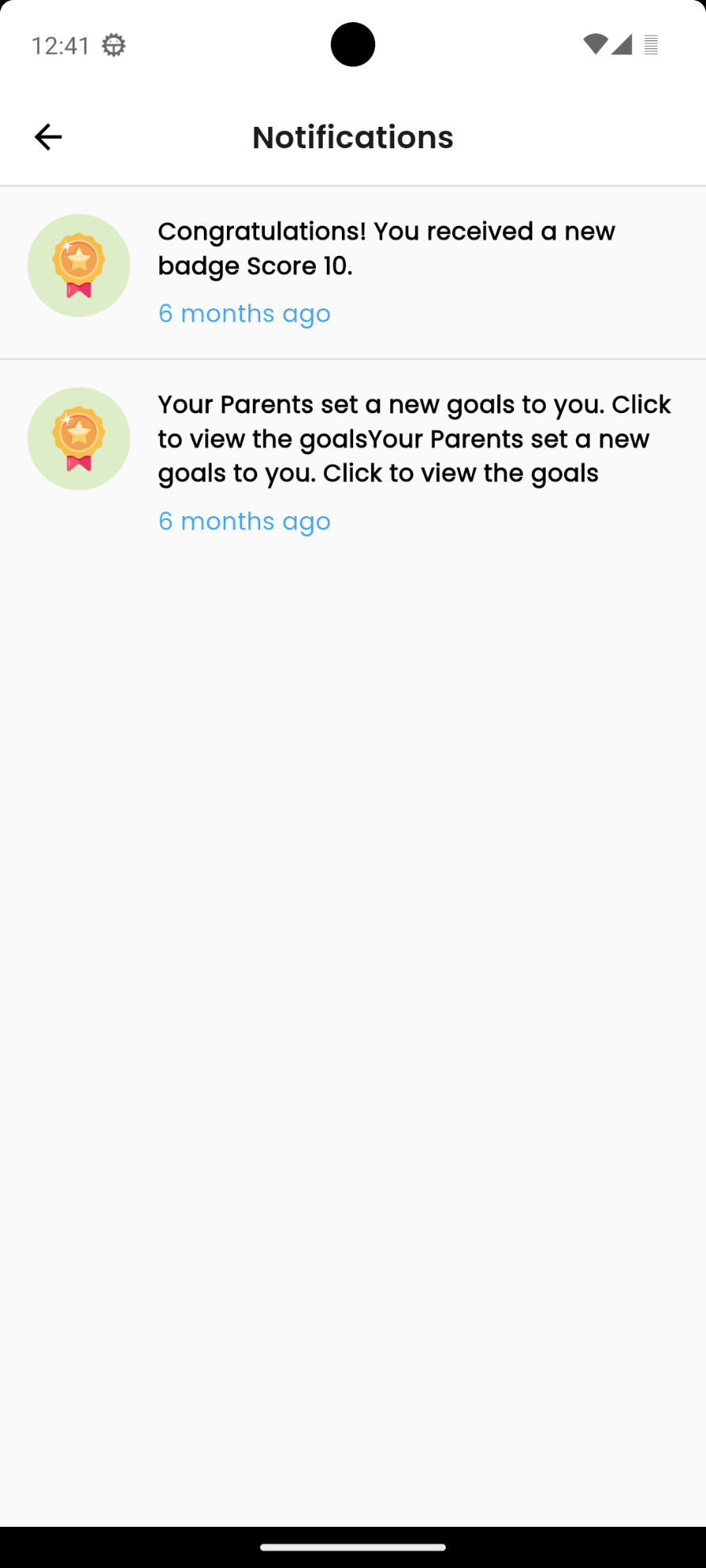
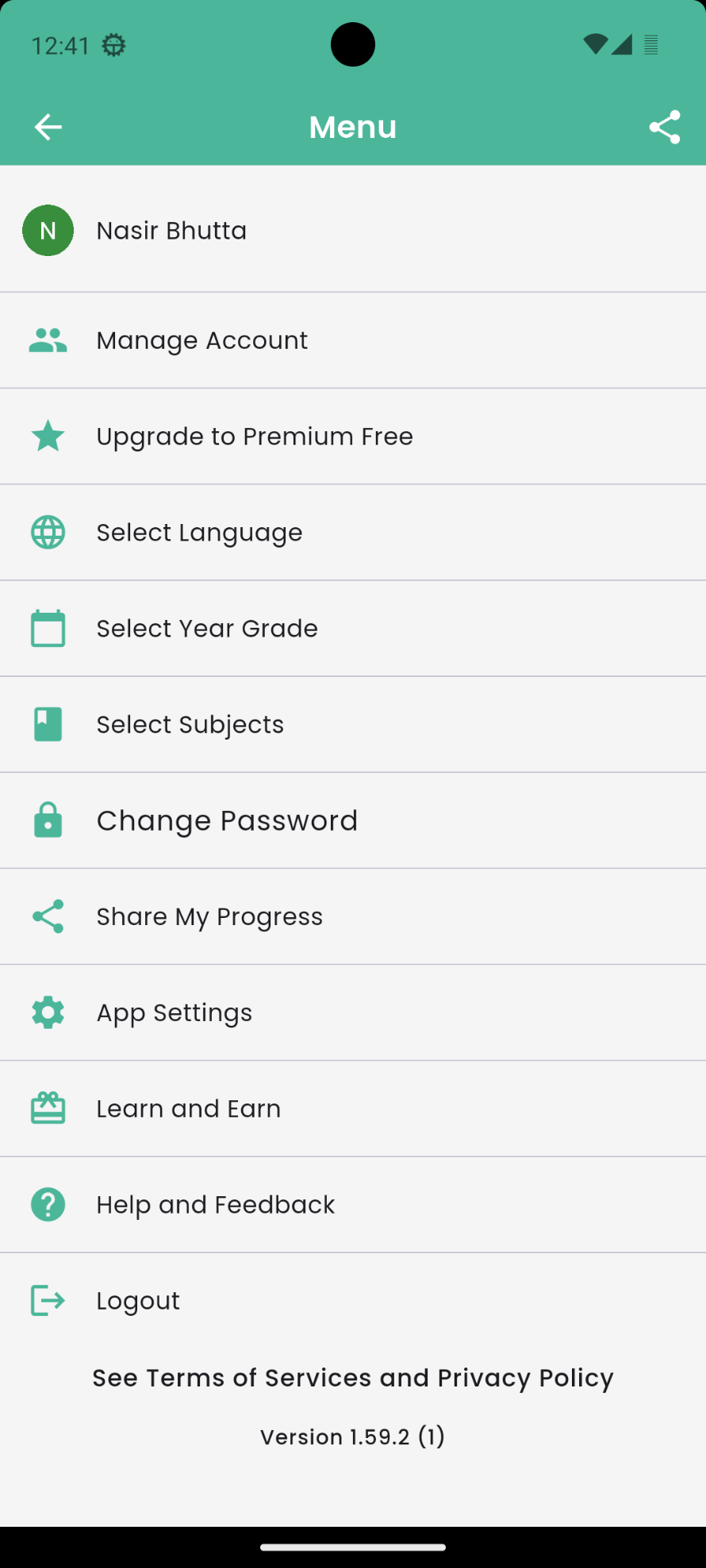
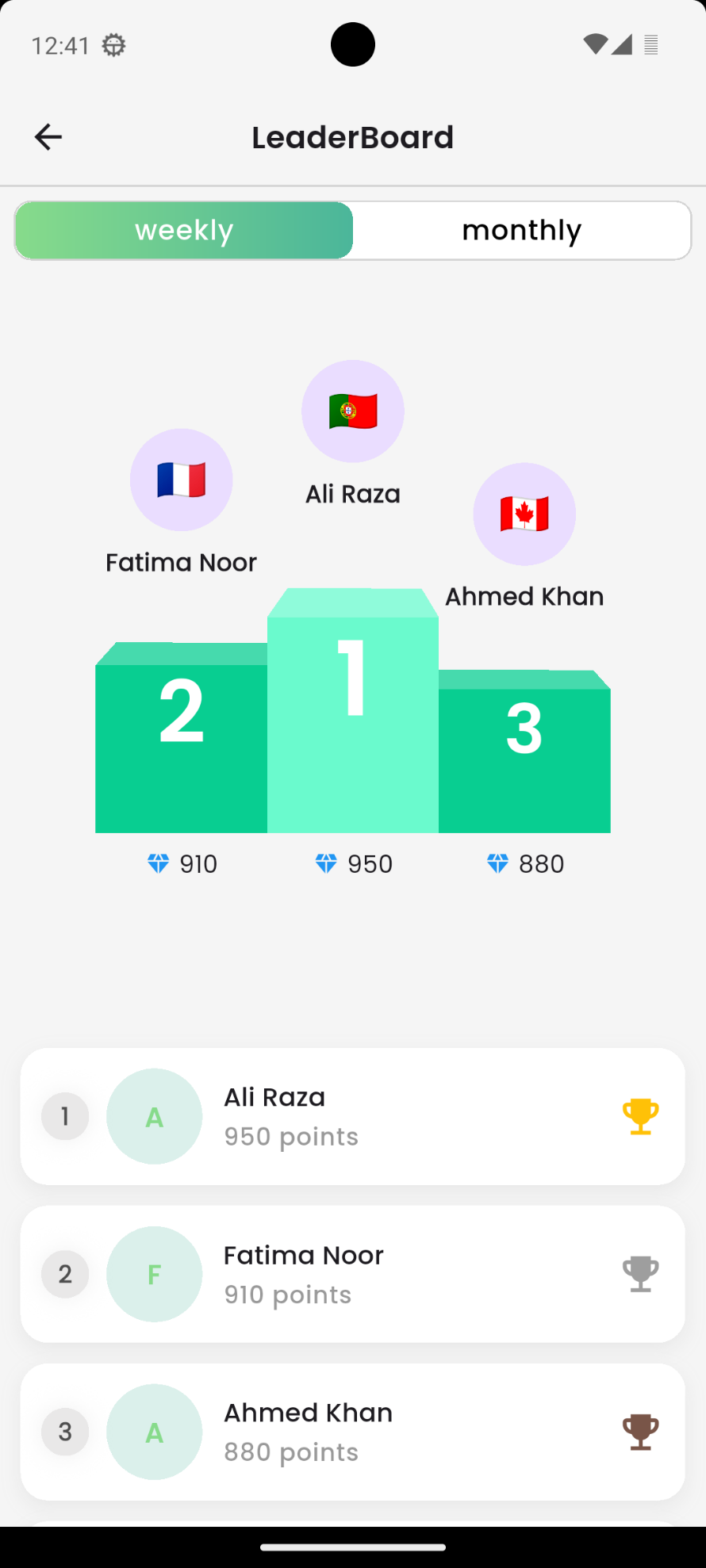
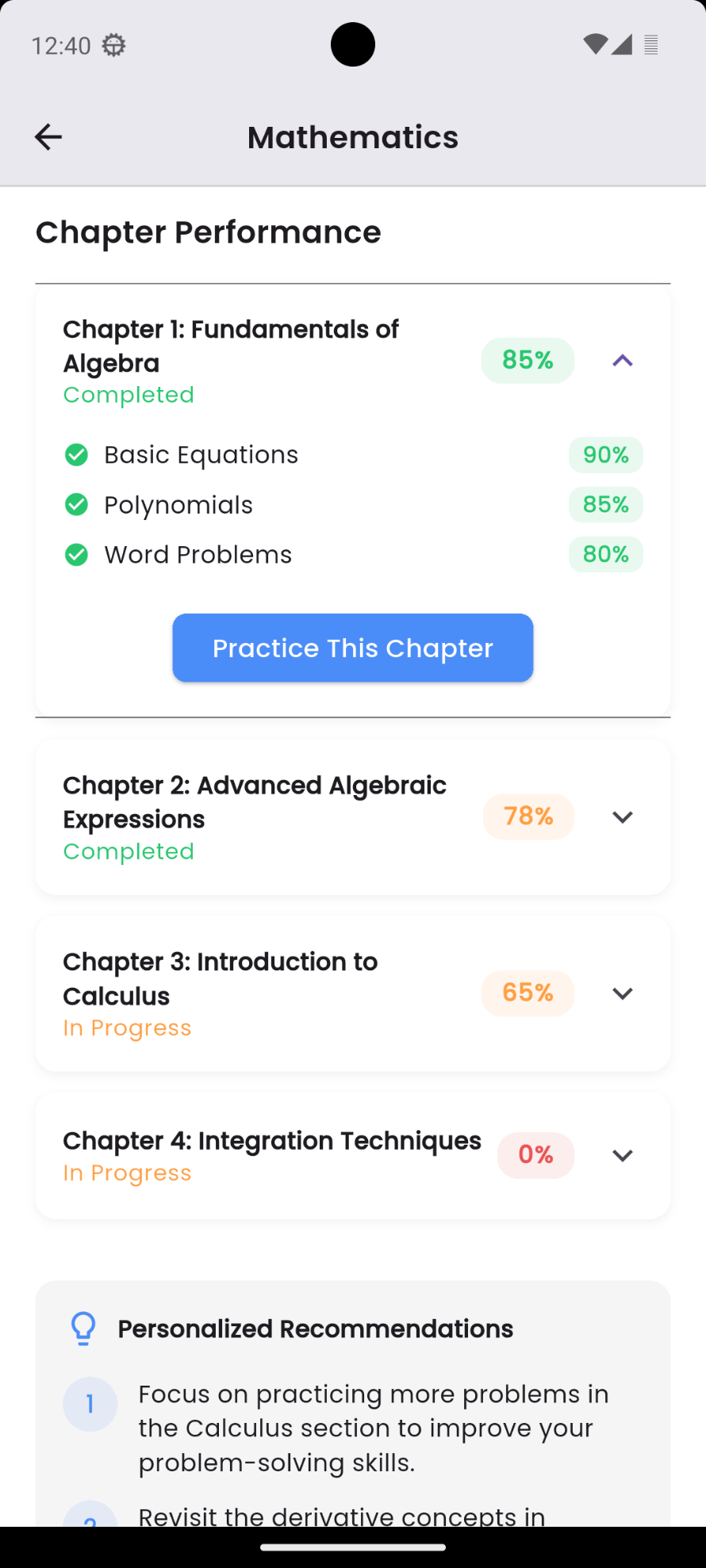
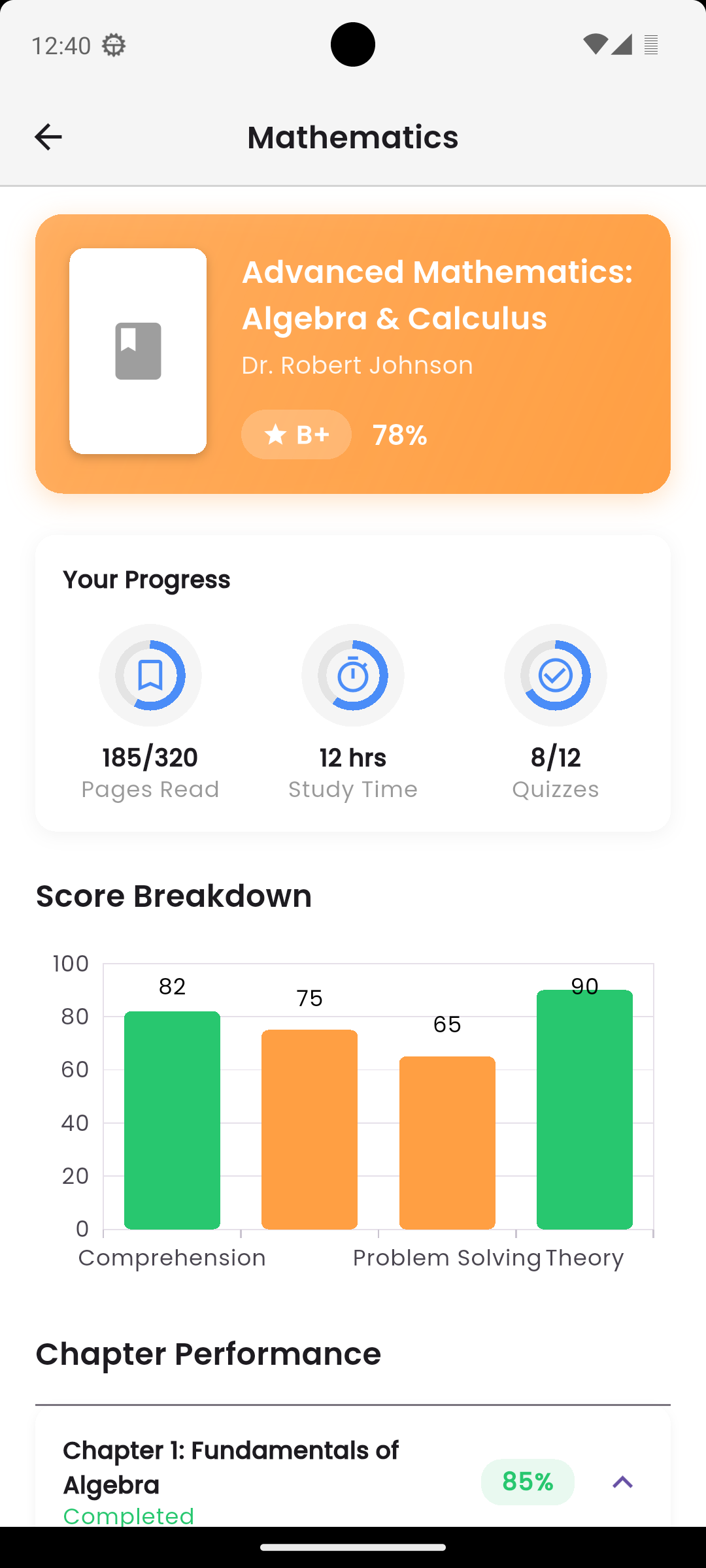
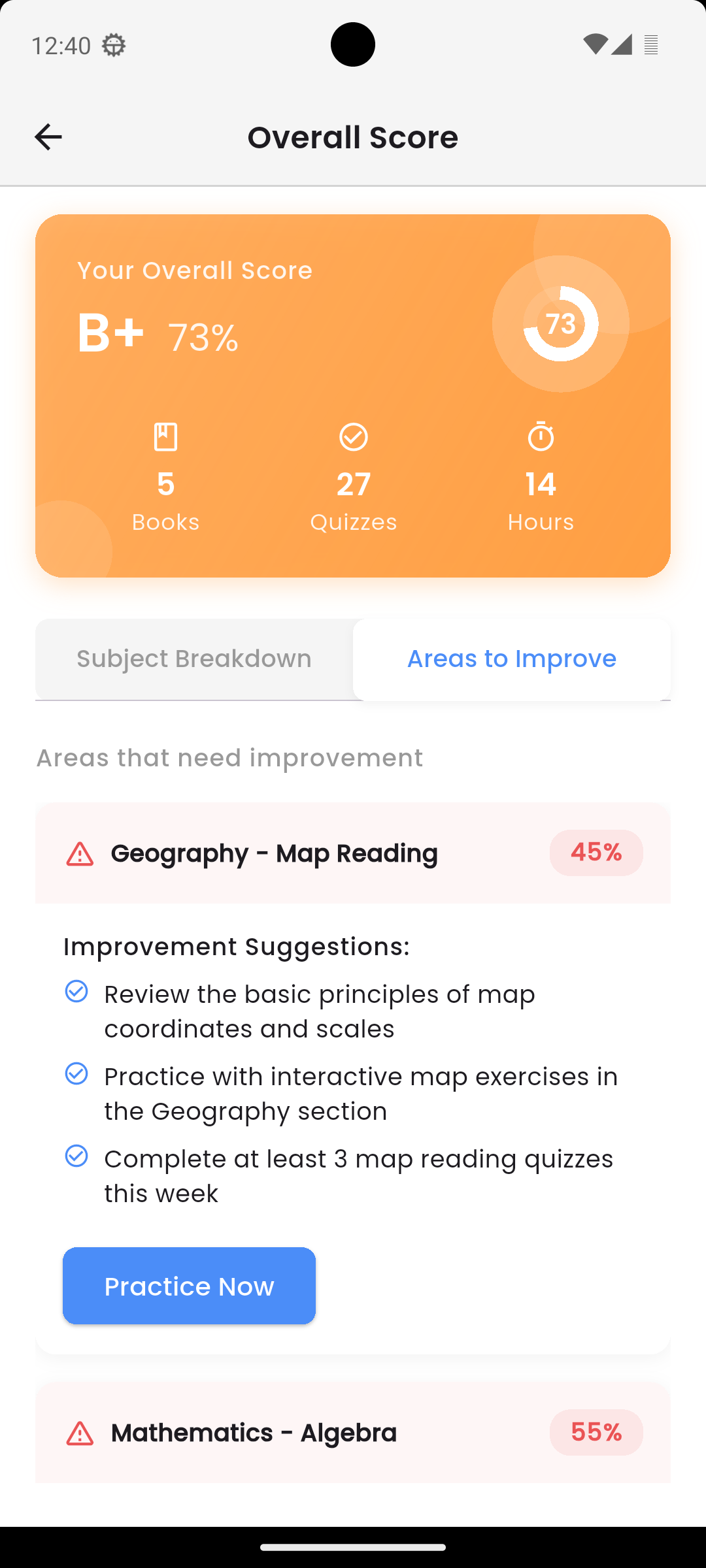
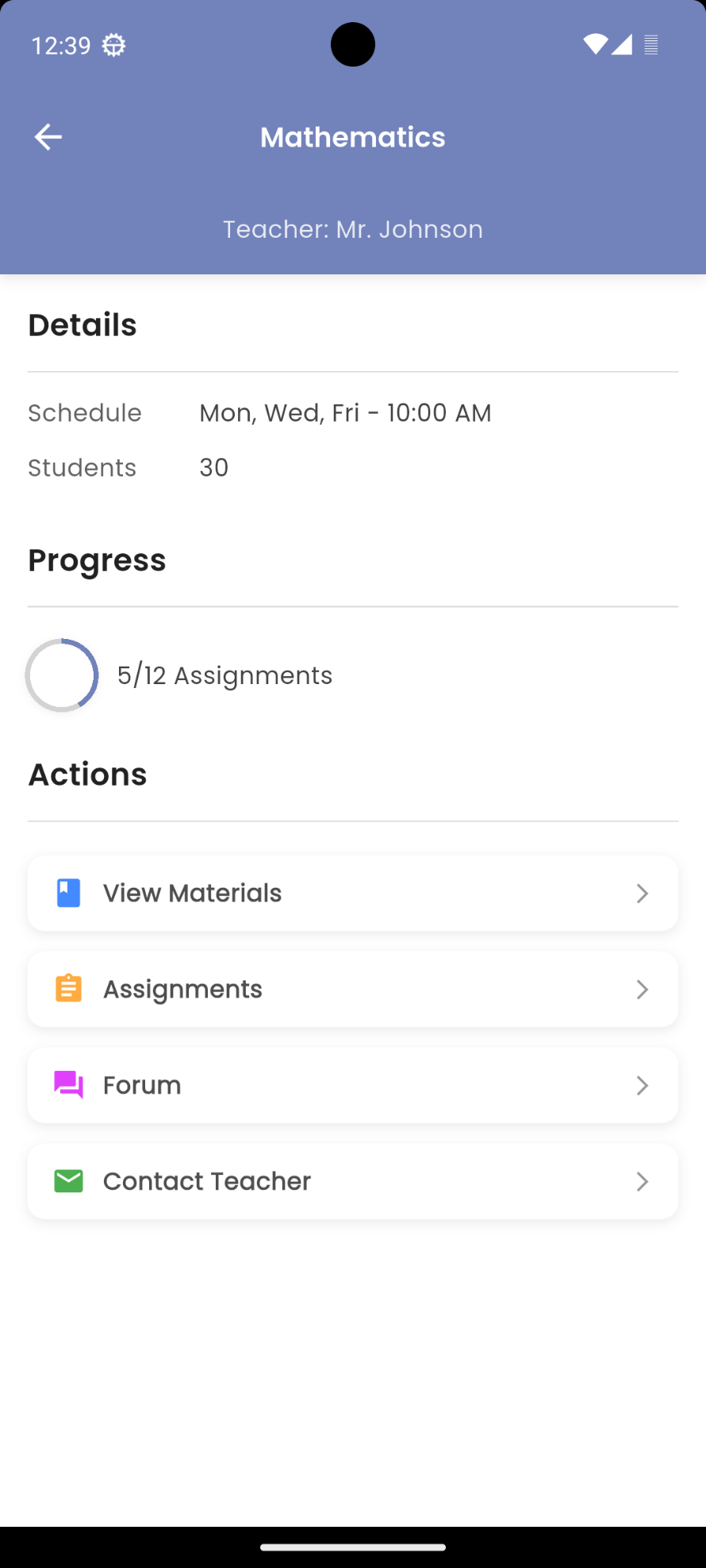
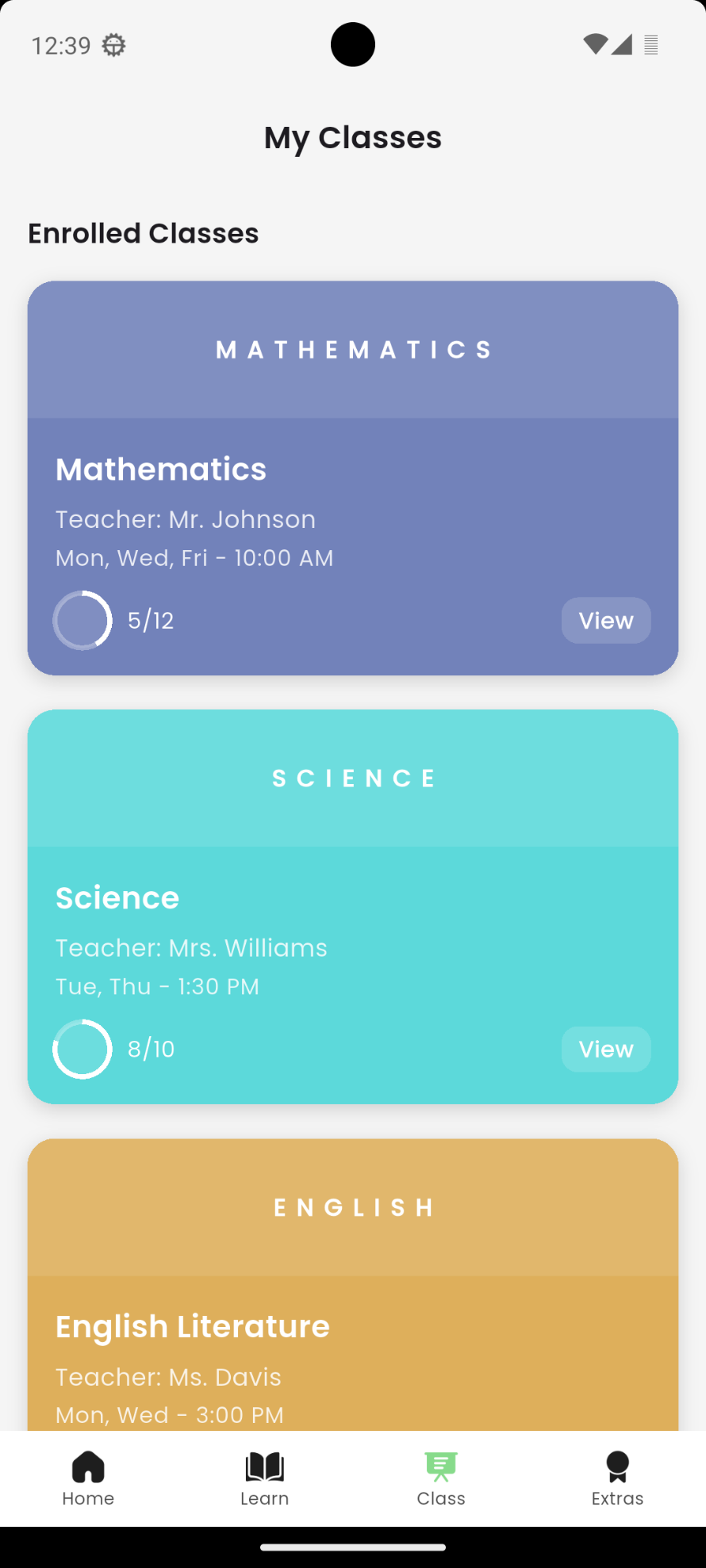
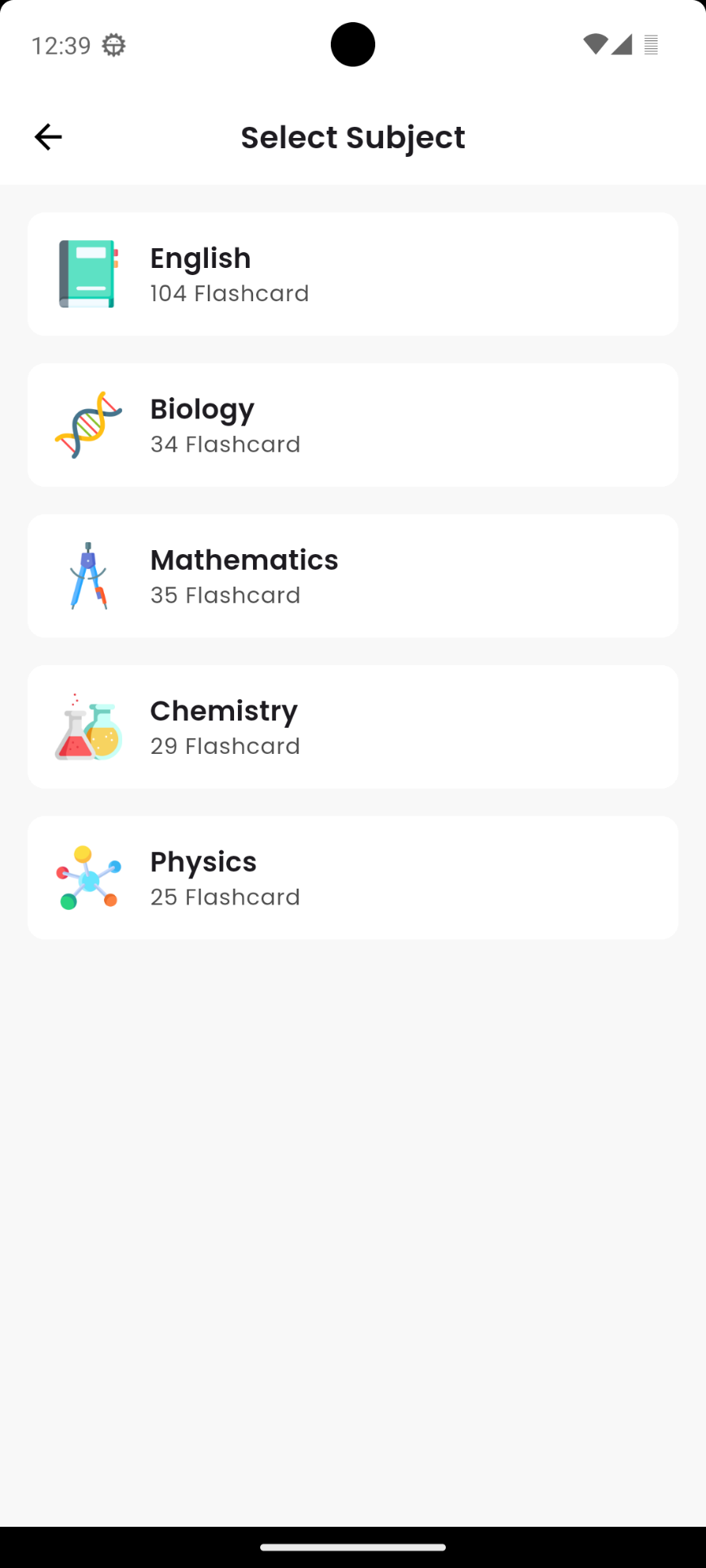
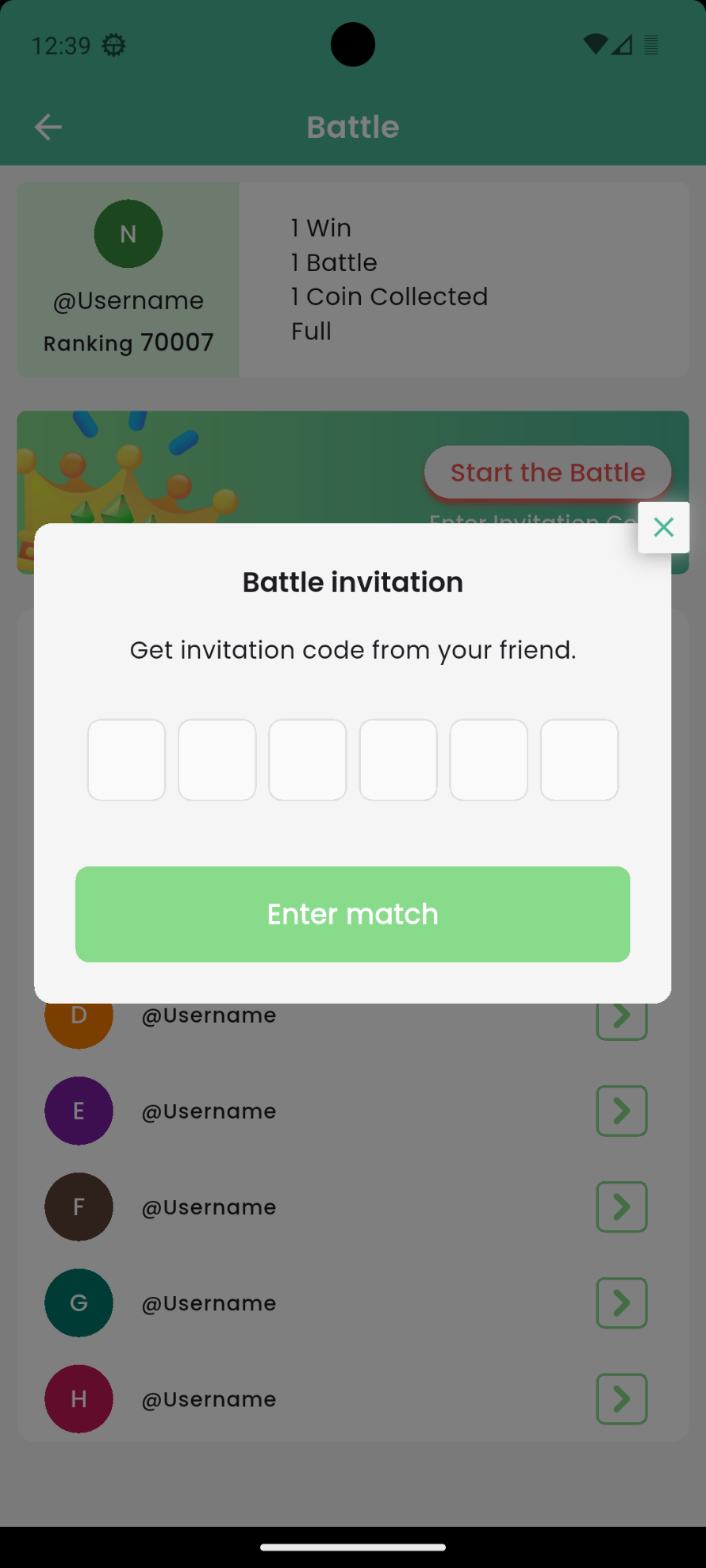
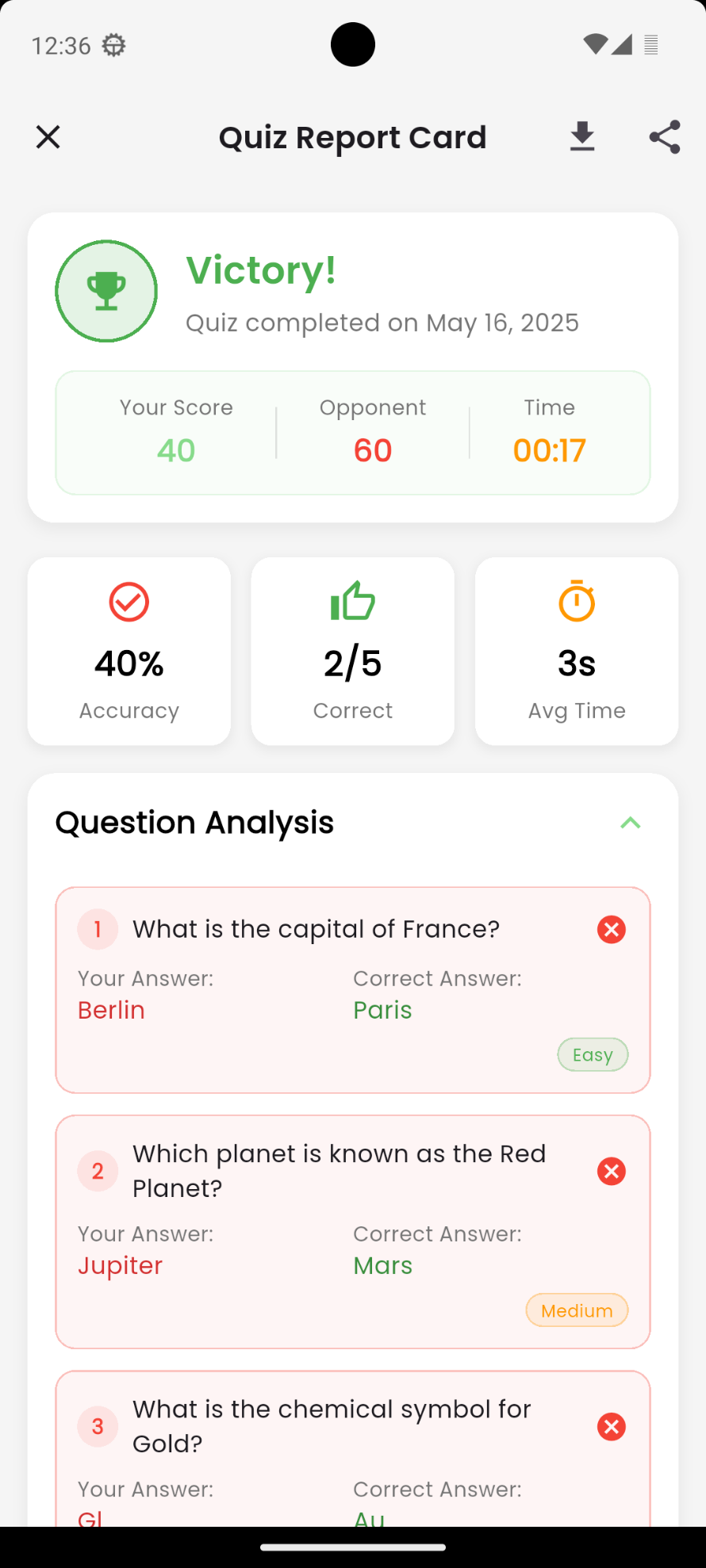
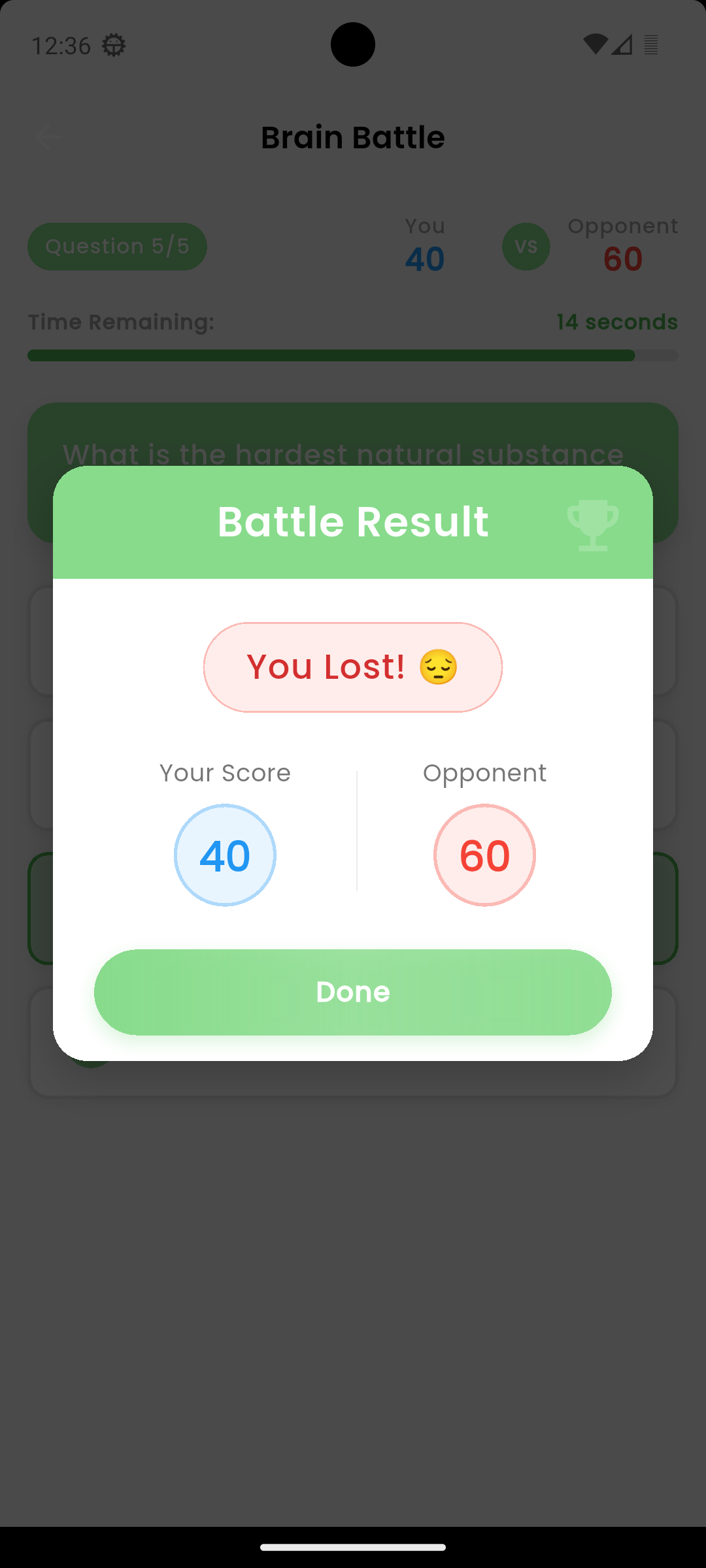
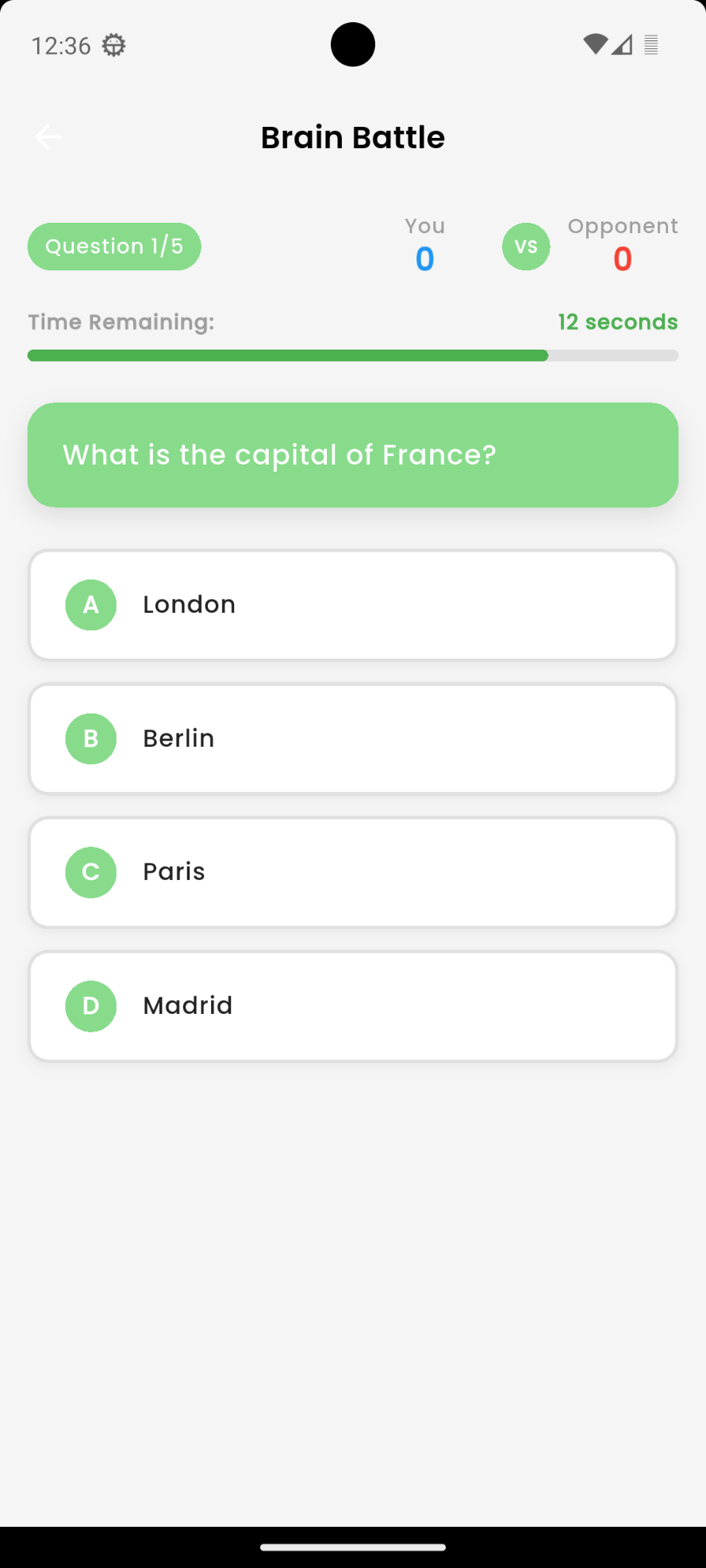
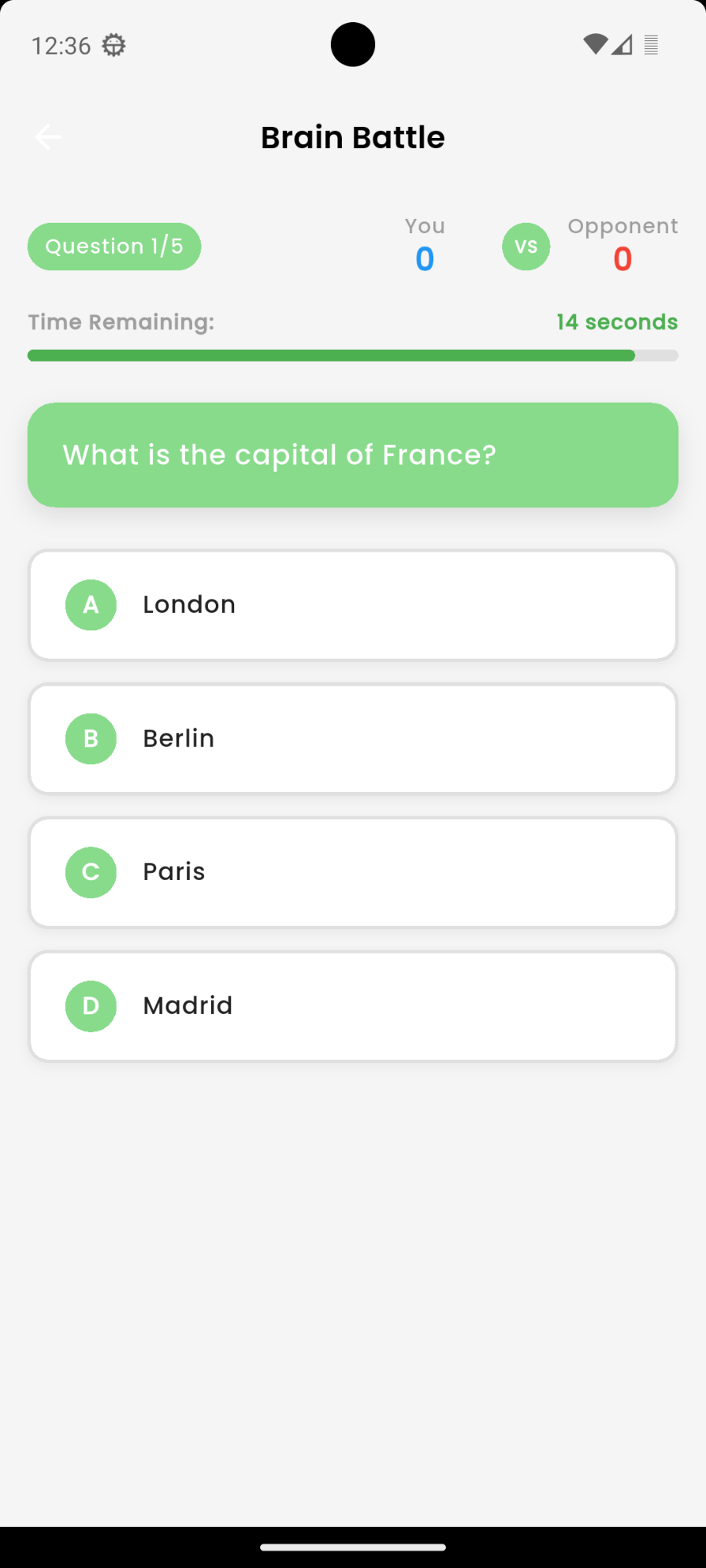
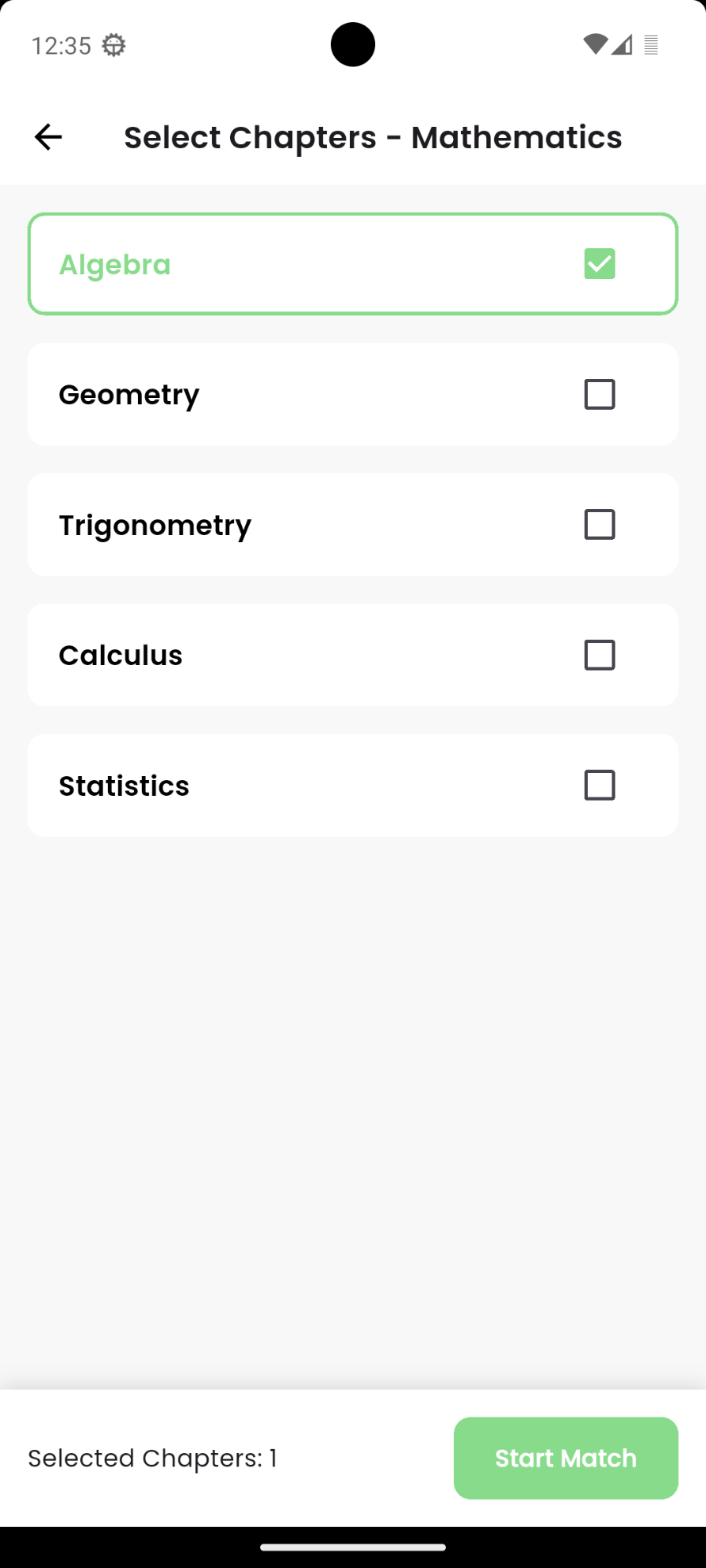
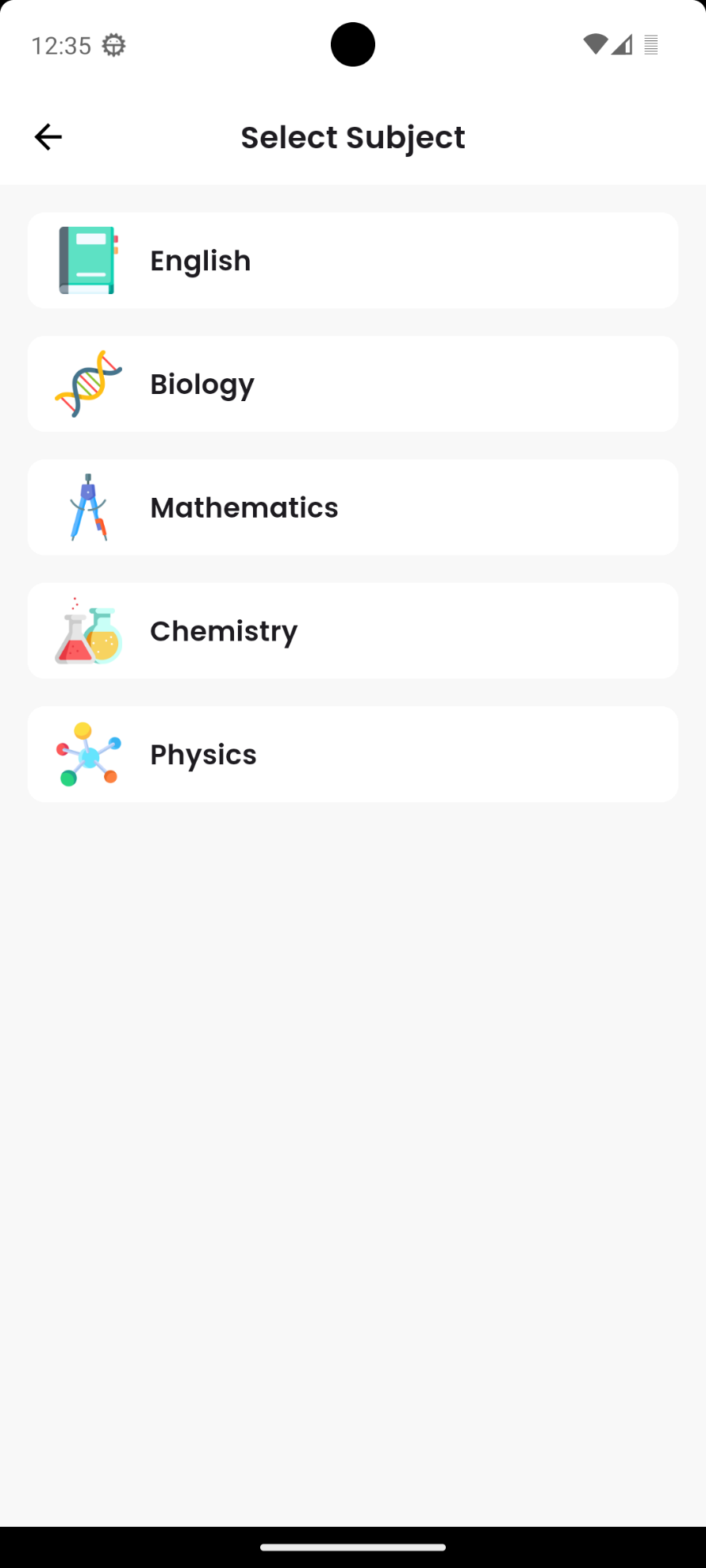
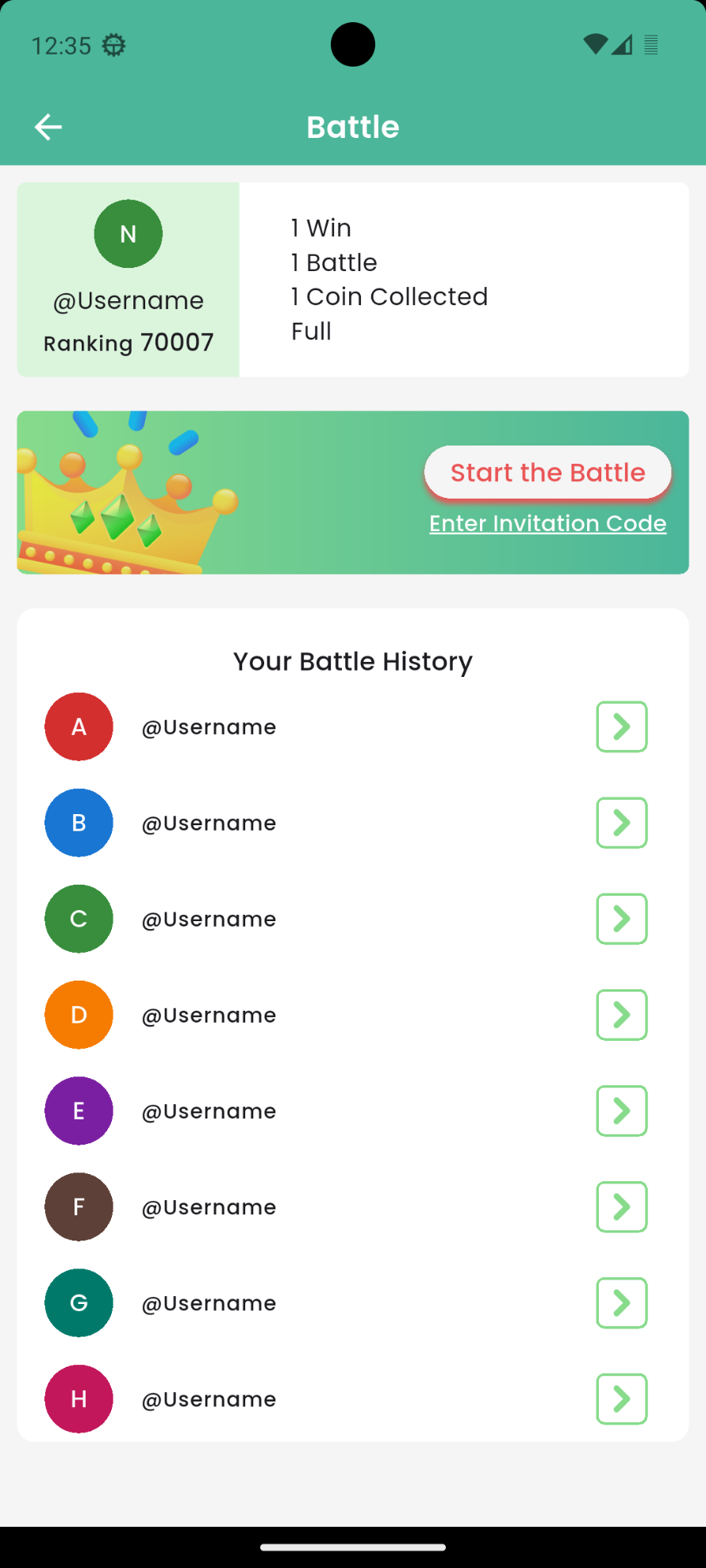
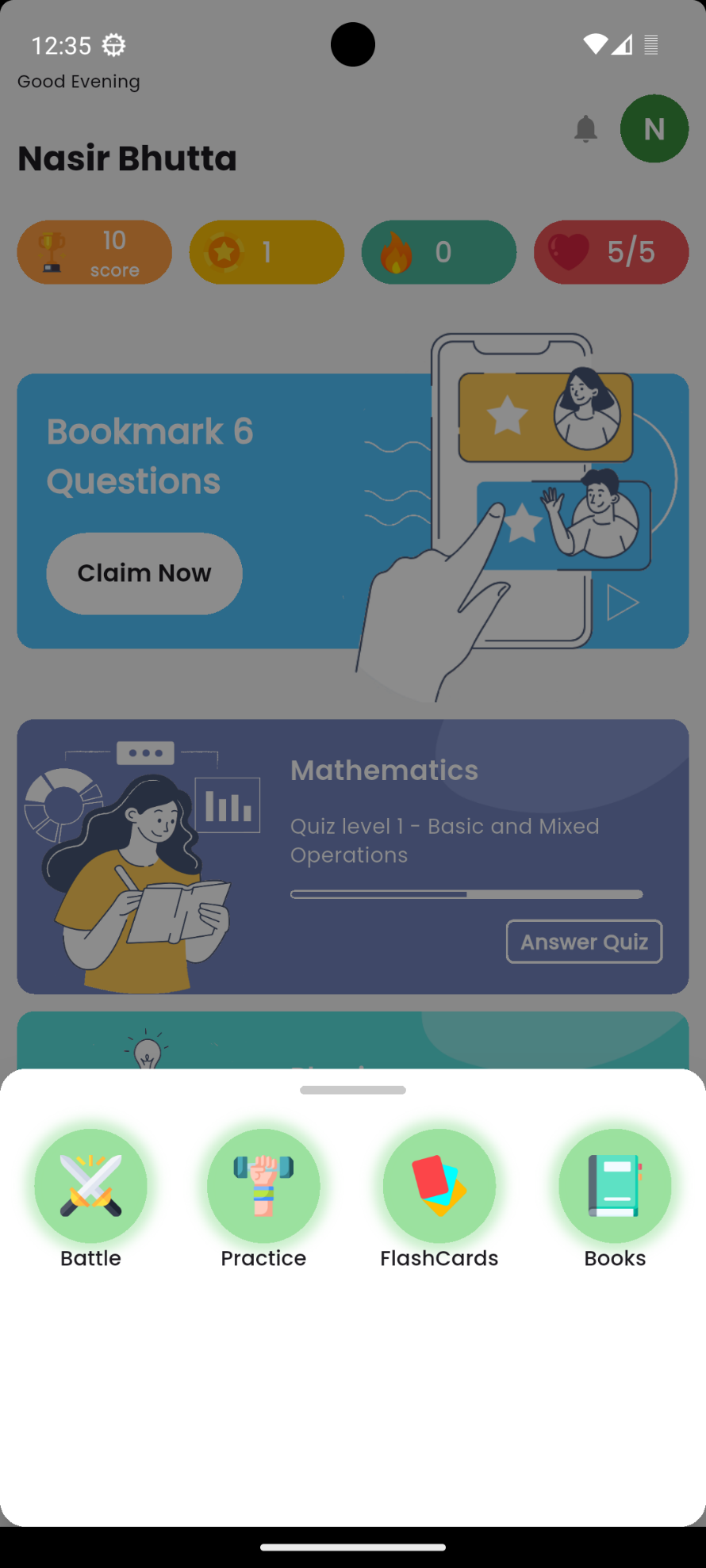
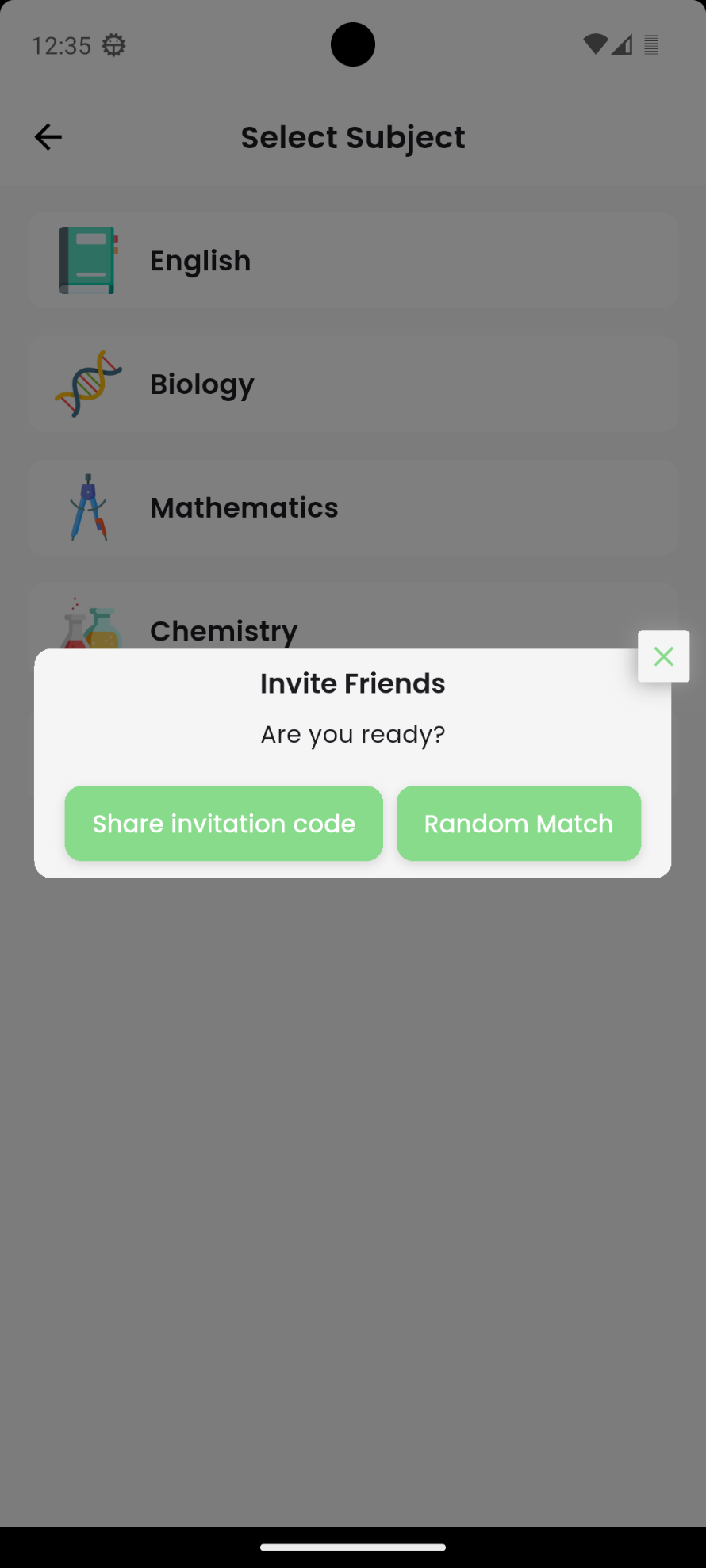
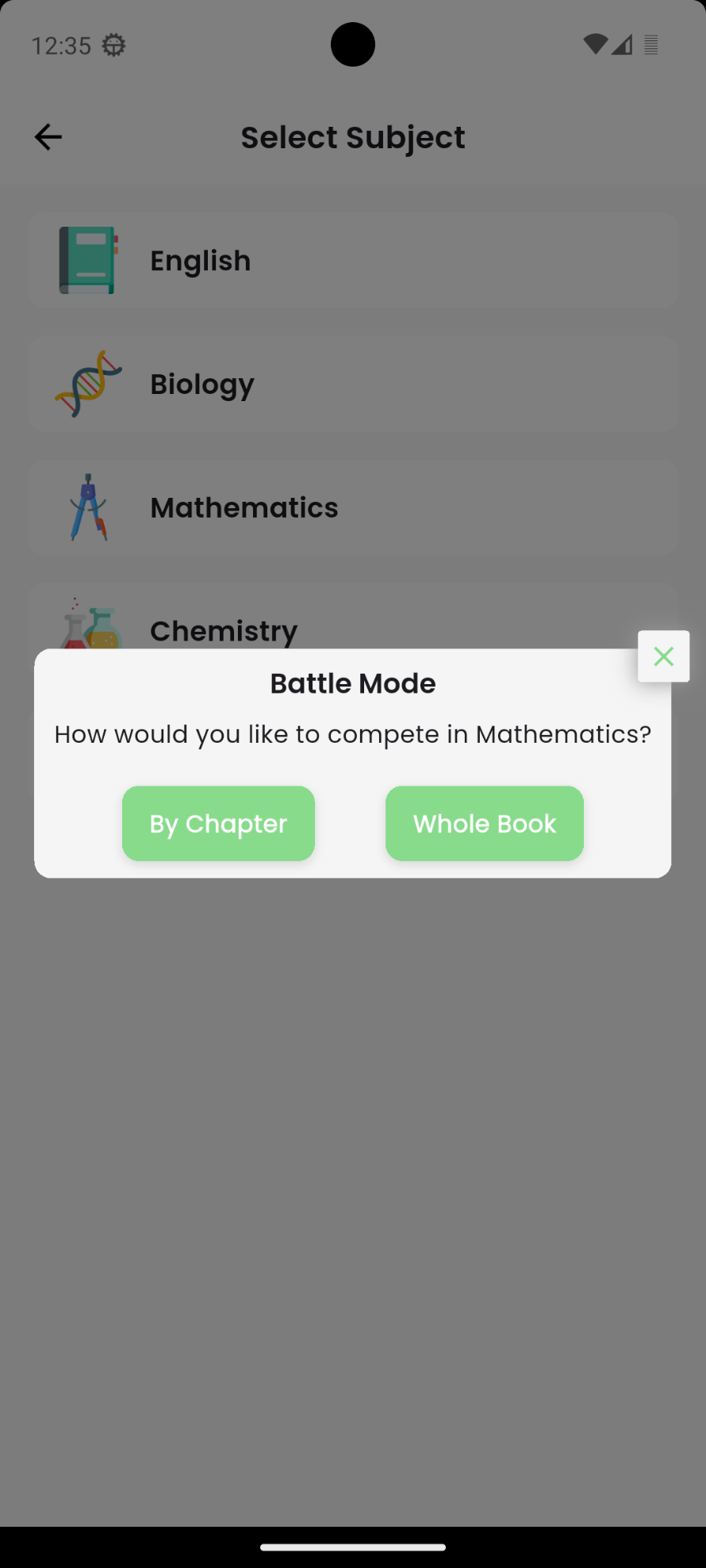
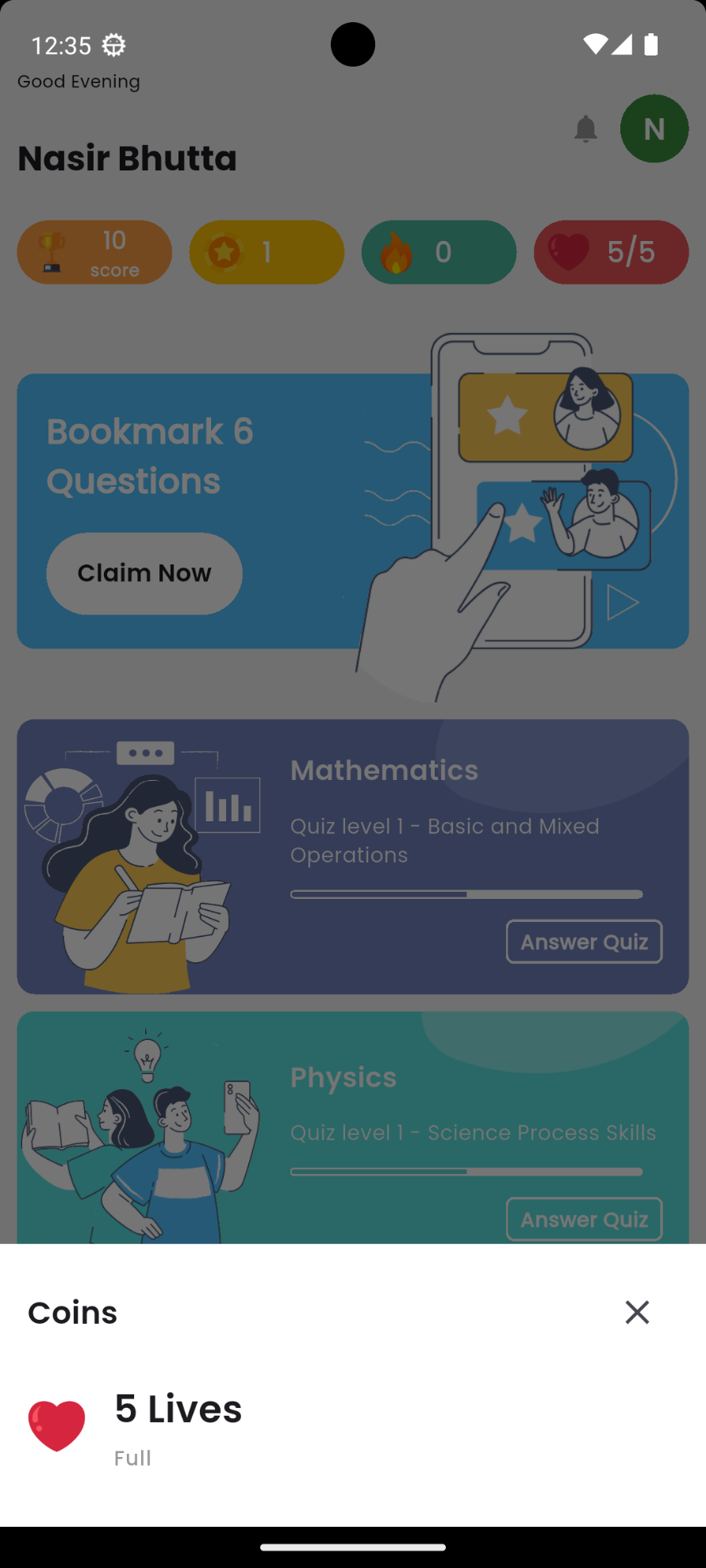
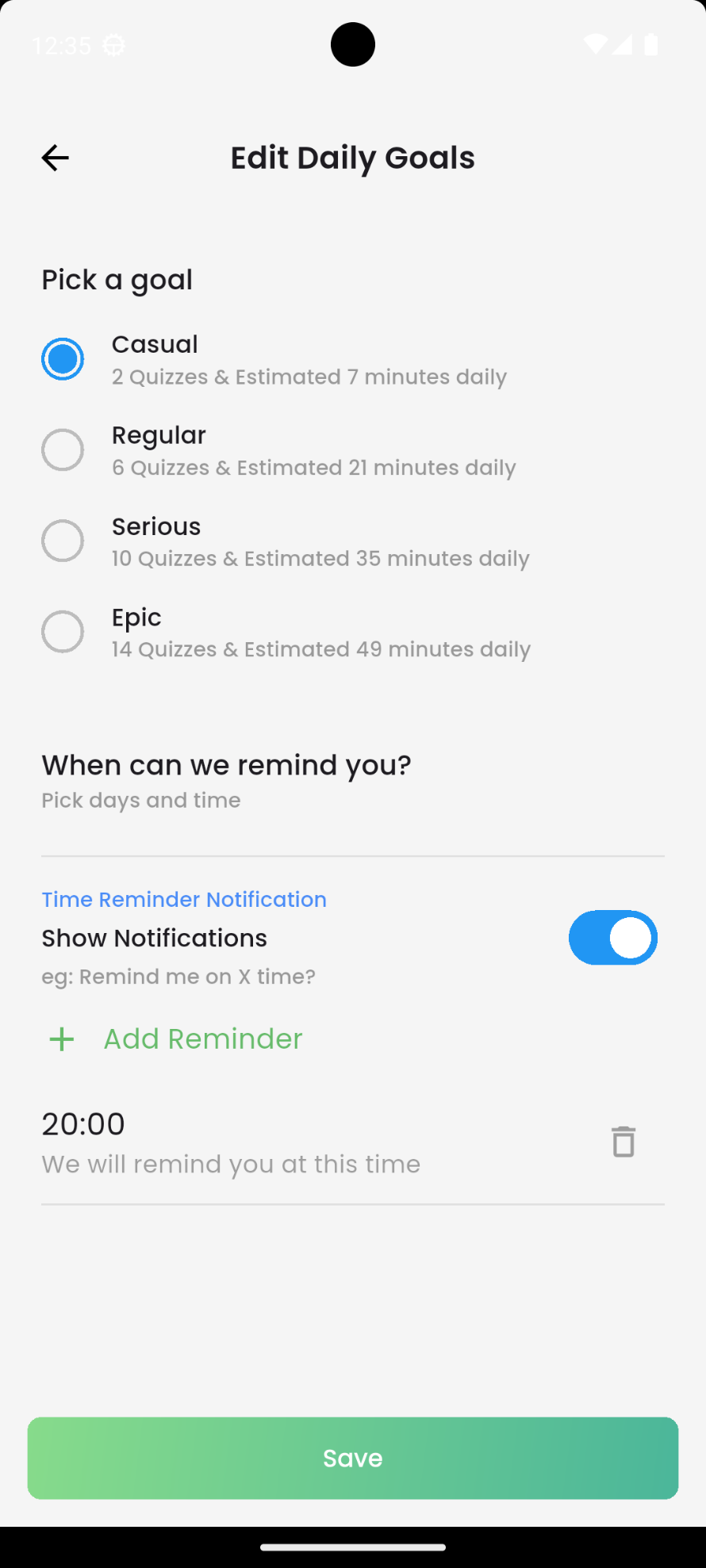
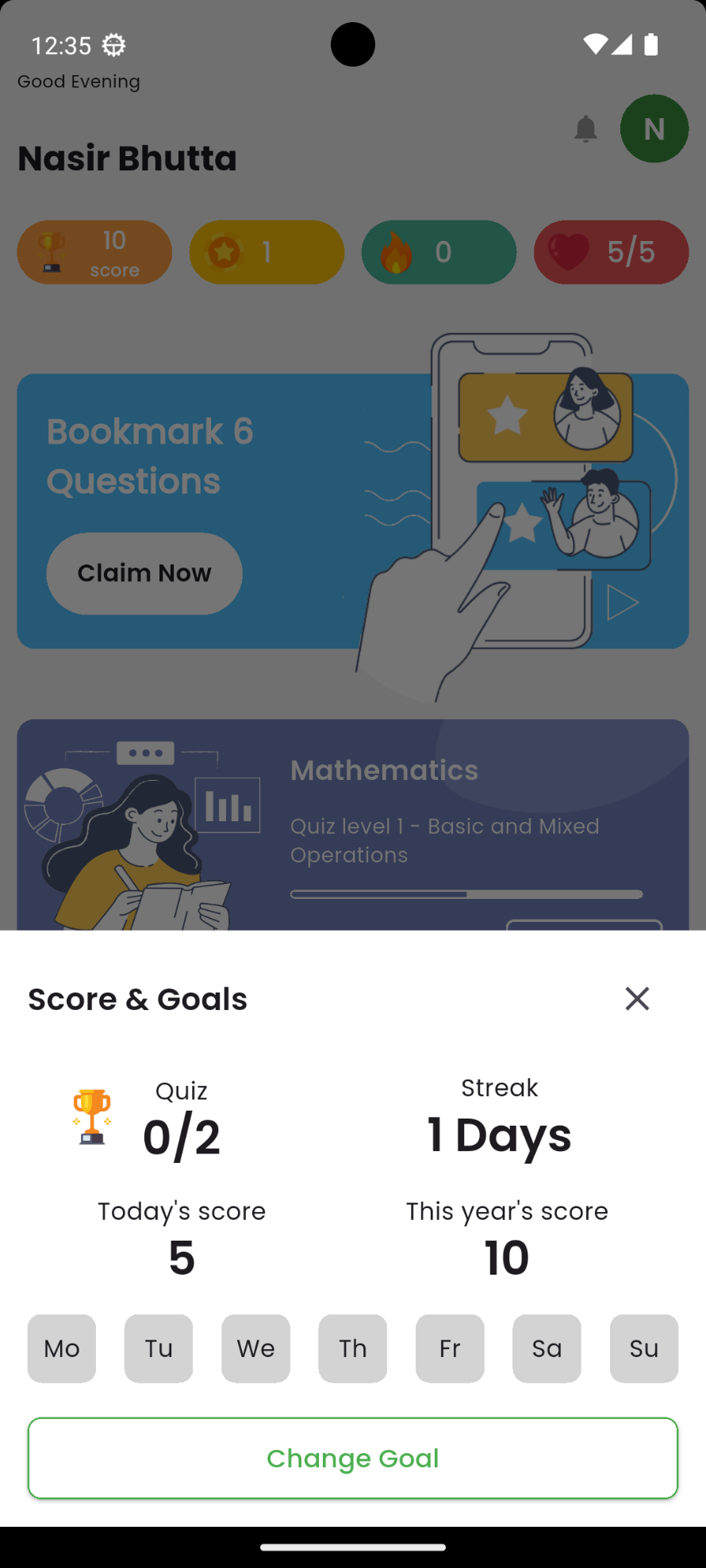
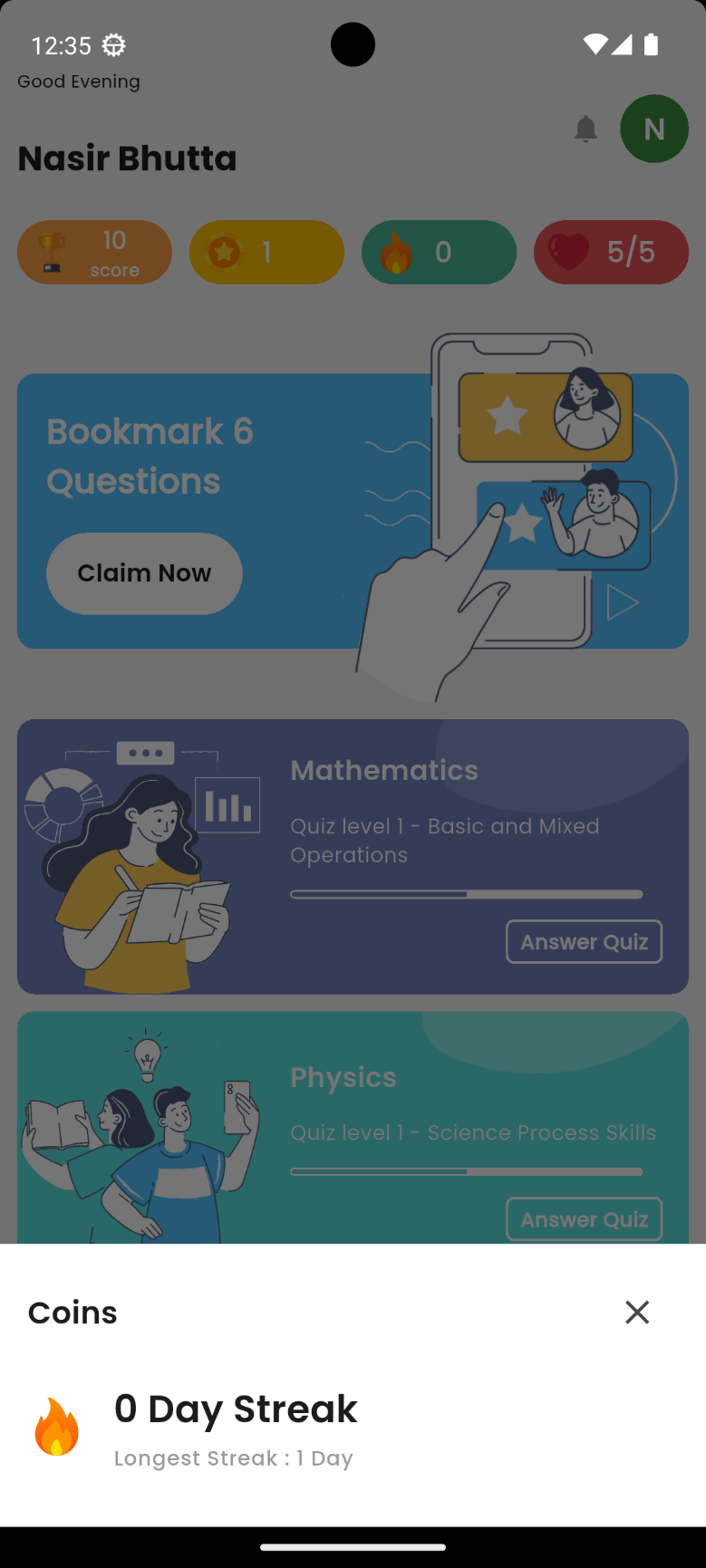
**SignUp screen**

****

**Dashboard**

**Goals Screen**

****

****

**7. Implementation**

**7.1 Algorithms**

This subsection explains the algorithms implemented within **BrainBee**, highlighting their roles in optimizing routes, calculating dynamic pricing, and managing ride requests efficiently.

**Algorithm 1.2 Get Flashcards**

| **Input: Book ID, Chapter ID** |
| --- |
| **Output:** List of flashcards |

function GetFlashcards(bookID, chapterID)

flashcards ← DB.query("SELECT \* FROM Flashcards WHERE bookID = ? AND chapterID = ?", bookID, chapterID)

return flashcards

end

**Algorithm 1.3 Match Peer For Battle**

| **Input: Student ID** |
| --- |
| **Output:** Matching result message |

function MatchPeerForBattle(studentID)

waitingList ← GetWaitingPeers()

for each peer in waitingList do

if IsCompatible(peer, studentID) then

CreateBattleRoom(studentID, peer)

return "Matched with " + peer.name

return "No match found. Please wait."

end

**Algorithm 1.4 Generate Quiz**

| **Input: Chapter ID, Difficulty Level** |
| --- |
| **Output:** Quiz (list of questions) |

function GenerateQuiz(chapterID, difficultyLevel)

allQuestions ← GetQuestions(chapterID)

suitable ← FilterByDifficulty(allQuestions, difficultyLevel)

quiz ← RandomSelect(suitable, 10)

return quiz

end

**Algorithm 1.5 Generate Notes From Video**

| **Input: Video ID** |
| --- |
| **Output:** Summarized notes |

function GenerateNotesFromVideo(videoID)

video ← LoadVideo(videoID)

transcript ← Transcribe(video)

notes ← Summarize(transcript)

return notes

end

**Algorithm 1.6 Assign Goal**

| **Input: Child ID, Goal Text, Deadline** |
| --- |
| **Output:** Goal assignment confirmation |

function AssignGoal(childID, goalText, deadline)

goal ← CreateGoal(goalText, deadline)

AssignToChild(childID, goal)

return "Goal assigned"

end

**Algorithm 1.7 Enroll In Class**

| **Input: Student ID, Class ID** |
| --- |
| **Output:** Enrollment status |

function EnrollInClass(studentID, classID)

class ← GetClass(classID)

if class.capacity > class.enrolled then

Enroll(studentID, classID)

return "Enrolled"

return "Class Full"

end

**Algorithm 1.8 Calculate Progress**

| **Input: Student ID, Subject ID** |
| --- |
| **Output:** Progress percentage |

function CalculateProgress(studentID, subjectID)

totalTasks ← GetAllTasks(subjectID)

completed ← GetStudentCompletions(studentID, subjectID)

progress ← (completed / totalTasks) \* 100

return progress

end

**Algorithm 1.9 Check Rewards**

| **Input: Child ID** |
| --- |
| **Output:** Rewards issued (if any) |

function CheckRewards(childID)

goals ← GetGoals(childID)

for goal in goals do

if goal.status == "Completed" and goal.rewardGiven == false then

GiveReward(childID, goal.reward)

MarkRewardGiven(goal)

end

**Algorithm 1.10 Evaluate Quiz**

| **Input: Quiz ID, Student Answers** |
| --- |
| **Output:** Score and feedback |

function EvaluateQuiz(quizID, studentAnswers)

correctAnswers ← GetCorrectAnswers(quizID)

score ← 0

for i in range(len(correctAnswers)) do

if studentAnswers[i] == correctAnswers[i] then

score += 1

feedback ← GenerateFeedback(score)

return score, feedback

end

**Algorithm 1.11 Get Parent Dashboard**

| **Input: Parent ID** |
| --- |
| **Output:** Dashboard data |

function GetParentDashboard(parentID)

children ← GetChildren(parentID)

dashboard ← []

for child in children do

stats ← GetChildStats(child.id)

dashboard.append(stats)

return dashboard

end

**Algorithm 1.12 Generate AI Flashcards**

| **Input: Chapter Text** |
| --- |
| **Output:** Flashcards (Q&A pairs) |

function GenerateAIFlashcards(chapterText)

keyPoints ← ExtractKeyConcepts(chapterText)

flashcards ← []

for point in keyPoints do

question ← GenerateQuestion(point)

answer ← GenerateAnswer(point)

flashcards.append((question, answer))

return flashcards

end

**Algorithm 1.13 Recommend Content**

| **Input: User ID** |
| --- |
| **Output:** Recommended content list |

function RecommendContent(userID)

history ← GetUserHistory(userID)

similarUsers ← FindSimilarUsers(userID)

preferredTopics ← AnalyzeWeakAreas(history)

recommended ← []

for topic in preferredTopics do

content ← FetchContent(topic)

recommended.append(content)

return recommended

end

**Algorithm 1.14 Match For Battle**

| **Input: Student ID** |
| --- |
| **Output:** Matched Peer or Waiting Message |

function MatchForBattle(studentID)

userLevel ← GetUserSkillLevel(studentID)

availablePeers ← GetWaitingStudents()

matchedPeer ← null

for peer in availablePeers do

if abs(GetUserSkillLevel(peer) - userLevel) ≤ threshold then

matchedPeer ← peer

break

if matchedPeer ≠ null then

CreateBattleRoom(studentID, matchedPeer)

return matchedPeer

return "No suitable peer found. Waiting in queue."

end

**Algorithm 1.15 Calculate Overall Score**

| **Input: Student ID** |
| --- |
| **Output:** Overall score percentage |

function CalculateOverallScore(studentID)

books ← GetAllBooksAttempted(studentID)

totalScore ← 0

totalQuestions ← 0

for book in books do

(score, questions) ← GetBookScore(studentID, book.id)

totalScore += score

totalQuestions += questions

if totalQuestions == 0 then

return 0

return (totalScore / totalQuestions) \* 100

end

**Algorithm 1.16 Calculate Book-Wise Scores**

| **Input: Student ID** |
| --- |
| **Output:** Dictionary of book-wise percentages |

function CalculateBookWiseScores(studentID)

books ← GetAllBooksAttempted(studentID)

scores ← {}

for book in books do

(score, questions) ← GetBookScore(studentID, book.id)

if questions > 0 then

percentage ← (score / questions) \* 100

scores[book.title] ← percentage

return scores

end

**Algorithm 1.17 Recommend Content (Hybrid)**

| **Input: User ID** |
| --- |
| **Output:** Ranked list of recommended items |

function RecommendContent(user\_id)

user\_profile ← GetUserPreferences(user\_id)

similar\_users ← FindSimilarUsersUsingCosineSimilarity(user\_profile)

content\_based\_recs ← RecommendBasedOnUserProfile(user\_profile)

collaborative\_recs ← RecommendFromSimilarUsers(similar\_users)

combined\_recs ← MergeAndRank(content\_based\_recs, collaborative\_recs)

return combined\_recs

end

**Algorithm 1.18 Generate Explanation**

| **Input: Topic Text** |
| --- |
| **Output:** Detailed Explanation |

function GenerateExplanation(topic\_text)

keywords ← ExtractKeyConcepts(topic\_text)

explanation ← []

for keyword in keywords do

step\_explanation ← UseGPTOrT5ToExplain(keyword)

explanation.append(step\_explanation)

return explanation

end

**Algorithm 1.19 Generate Summary**

| **Input: Content** |
| --- |
| **Output:** Summary |

function GenerateSummary(content)

cleaned\_text ← PreprocessText(content)

summary ← RunPegasusOrT5Summarizer(cleaned\_text)

return summary

end

**Algorithm 1.20 Check and Reward Goal**

| **Input: User ID, Goal ID** |
| --- |
| **Output:** Reward given (Boolean) |

function CheckAndRewardGoal(user\_id, goal\_id)

goal ← GetGoalDetails(goal\_id)

progress ← GetUserProgress(user\_id, goal.book, goal.chapter)

if progress ≥ goal.target then

reward ← AssignReward(user\_id)

return true

else

return false

end

## **7.2 External APIs/SDKs**

Describe the third-party APIs/SDKs used in the project implementation in the following table. Few examples of APIs are provided in the table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of API/SDK or Library** | **Description** | **Purpose of Usage** | **API Endpoint/Function Used** |
| **Cloudinary** | Image and video management solution | Uploading user-generated images (e.g., avatars, notes) to cloud storage | https://api.cloudinary.com/v1\_1/demo/image/upload |
| **Firebase Cloud Messaging (FCM)** | Messaging and push notification service | Sending notifications to users (e.g., quiz results, live class reminders) | FirebaseMessaging.onMessage, FirebaseMessaging.send() |
| **MongoDB Atlas Search** | Full-text search engine integrated with MongoDB | Searching books, topics, and flashcards across user data | db.collection.aggregate([{ $search: {...} }]) |
| **scikit-learn (Decision Tree)** | ML library for classification and regression | Classifying student performance and guiding personalized learning paths | DecisionTreeClassifier().fit(), .predict() |
| **scikit-learn (TF-IDF)** | Term Frequency–Inverse Document Frequency model | Identifying important keywords for AI flashcard and summary generation | TfidfVectorizer().fit\_transform() |
| **pyirt / IRT Model** | Item Response Theory model | Adaptive quiz difficulty and estimating student ability levels | irt\_model.fit(), irt\_model.predict() |
| **spaCy / Hugging Face Transformers** | Natural Language Processing (NLP) libraries | Summarizing educational content, generating explanations and paraphrased notes | nlp\_model(text), transformers.pipeline("summarization") |
| **Reinforcement Learning (Q-learning / Custom)** | Reward-based ML model | Used in dynamic battle matchmaking and progress optimization | env.step(), agent.learn() |
| **Socket.IO / Pusher** (Optional) | Real-time communication engine | Enables real-time features such as battles, collaborative learning, chat etc. | socket.emit(), socket.on() or Pusher.trigger() |

**8. Testing and Evaluation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Check the email field of login | Email: [abc@gmail.com](mailto:abc@gmail.com) | Validates email address and moves cursor to next textbox | Pass |
| 2 | Check the email field for invalid format | Email: abc.gmail.com | Highlights field and displays error message | Pass |
| 3 | Validate password length in signup | Password: 1234 | Displays error message for short password | Pass |
| 4 | Confirm password match in signup | Password: abc123, Confirm: abc1234 | Shows mismatch error | Pass |
| 5 | Ensure unique user registration | Email: [existing\_user@gmail.com](mailto:existing_user@gmail.com) | Returns "User already exists" | Pass |
| 6 | Authenticate user with correct credentials | Email: [user@gmail.com](mailto:user@gmail.com), Pass: abc123 | Redirects to dashboard | Pass |
| 7 | Authenticate user with wrong credentials | Email: [user@gmail.com](mailto:user@gmail.com), Pass: wrongpass | Shows "Invalid credentials" | Pass |
| 8 | Load flashcards for a chapter | Chapter ID: CH101 | Displays all flashcards for that chapter | Pass |
| 9 | Generate AI flashcard using NLP | Topic: Photosynthesis | Returns summary + flashcard text | Pending |
| 10 | Recommend book using Decision Tree | User performance: low on Book A | Suggests easier book based on DT rules | Pending |
| 11 | IRT-based score update | Correct/Incorrect answers logged | Updates latent trait (θ) for user | Pending |
| 12 | TF-IDF for video explanation summary | Transcript: 500 words | Returns top keywords & summary | Pending |
| 13 | Match user for battle | Skill Level: Beginner | Finds peer of similar skill level | Pass |
| 14 | Notification delivery via FCM | Message: "Class starts in 5 min" | Push notification received | Pass |
| 15 | Store battle result in MongoDB | User A: win, User B: lose | Battle result inserted in DB | Pass |
| 16 | Check token-based auth middleware | Token: valid JWT | Allows API access | Pass |
| 17 | Load enrolled live classes | User ID: 101 | Returns list of live class data | Pass |
| 18 | Fetch student book-wise score | Student ID: 101, Book: Science | Displays accurate percentage | Pass |
| 19 | Calculate total progress across all books | Student ID: 101 | Returns cumulative progress score | Pass |
| 20 | Test video notes download | Topic ID: T1001 | Downloads PDF notes of video transcript | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Attribute and value** | **Expected result** | **Result** |
| 1 | Validate Login screen email field accepts valid input | Email: abc@gmail.com | Field validates, focus shifts to password field | Pass |
| 2 | Check email field rejects invalid format | Email: abc.gmail.com | Shows error: “Enter valid email” | Pass |
| 3 | Verify password field hides input | Password: mypassword | Password is hidden with dots | Pass |
| 4 | Check Login button disables when fields are empty | Email: , Password: | Button remains disabled | Pass |
| 5 | Verify navigation from Login to Home on success | Tap Login → valid input | Navigates to Home page | Pass (mock navigation) |
| 6 | Check if Flashcards page opens from Dashboard | Tap Flashcards Icon | Navigates to Flashcards screen | Pass |
| 7 | Validate that loading indicator shows on long operation | Tap Login with delay | CircularProgressIndicator shown | Pass |
| 8 | Verify text field clears after login | After login success | Email and password fields clear | Pass |
| 9 | Validate tabs switch between “Live Classes” and “My Classes” | Tap tab | Corresponding screen content appears | Pass |
| 10 | Check Sign Up screen input validation | Name: empty, Email: invalid | Error shown below fields | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Attribute and value** | **Expected result** | **Result** |
| 1 | Complete Login flow | Valid Email + Password → Tap Login | Redirects to Dashboard | Pass |
| 2 | Access Book Detail page | From Book List → Tap Book Card | Opens Book Detail with info | Pass |
| 3 | Navigation from Book Detail to Flashcards | Tap “Flashcards” | Opens Flashcard screen | Pass |
| 4 | Try submitting empty Signup form | Tap Submit without input | Form displays validation errors | Pass |
| 5 | Open Sidebar Menu | Tap Menu icon | Drawer slides in with options | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Attribute and value** | **Expected result** | **Result** |
| 1 | Email must be unique (to be tested later via backend) | Duplicate Email | Shows “Email already exists” (not testable yet) | Pending |
| 2 | Password must be at least 6 characters | Input: 123 | Error: “Minimum 6 characters” | Pass |
| 3 | Cannot proceed to Flashcards without selecting book | Try opening Flashcards without selecting book | Displays “Select a book first” | Pass |
| 4 | Student cannot join Live Class without enrollment | Tap Join without selection | Error message shown | Pass |
| 5 | Check error when adding empty battle code | Input: | Shows “Enter valid code” | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Attribute and value** | **Expected result** | **Result** |
| 1 | Navigate Login → Dashboard | Valid login | Dashboard renders | Pass |
| 2 | Dashboard → Book List → Book Detail | Tap sequence | Proper screen transition | Pass |
| 3 | Book Detail → Flashcards | Tap Flashcards | Loads Flashcard UI | Pass |
| 4 | Home → Live Class tab | Tap tab | Loads Live Class screen | Pass |
| 5 | Live Class → Enrolled Class | Tap Enrolled | Shows list of enrolled classes | Pass |

Okay, here are test cases for the provided screens, following the format you specified and considering unit, functional, business, and integration testing perspectives for the frontend implementation, while assuming the authentication backend works.

**6.3 Test Cases**

This subsection provides test cases for the different screens and user interactions present in the BrainBee application, covering various testing types.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Test Case/Test Script | Attribute and value | Expected result | Result |
| 1 | **[Unit]** Check if the Login screen components render correctly. | Screen: Login (1). Components: Username/Email field, Password field, "Forgot Password?", "Sign In", Social login buttons. | All specified components should be visible and correctly rendered. | Pass |
| 2 | **[Functional]** Verify "Forgot Password?" link navigates. | Screen: Login (1). Action: Tap "Forgot Password?" link. | User should be navigated to the Forgot Password screen (assuming it exists). | Pass |
| 3 | **[Integration]** Test successful login redirects to Home. | Screen: Login (1). Action: Enter valid credentials, tap "Sign In". (Assume backend mocks successful response). | User should be successfully logged in and navigated to the Home Screen (3). | Pass |
| 4 | **[Functional]** Verify tapping "Sign Up" link navigates. | Screen: Login (1). Action: Tap "Sign Up" link. | User should be navigated to the Sign Up Screen (2). | Pass |
| 5 | **[Unit]** Check if the Sign Up screen components render correctly. | Screen: Sign Up (2). Components: Full Name field, Username/Email field, Password field, "Forgot Password?", "SignUp", Social login buttons. | All specified components should be visible and correctly rendered. | pass |
| 6 | **[Integration]** Test successful signup redirects to Home. | Screen: Sign Up (2). Action: Enter valid user details, tap "SignUp". (Assume backend mocks successful response). | User should be successfully signed up and navigated to the Home Screen (3). |  |
| 7 | **[Unit]** Check if the Home Screen displays user info and status correctly. | Screen: Home (3). Components: User name ("Nasir Bhutta"), Score, Coins, Lives, Notification icon, Profile icon. | User's name and current status (Score, Coins, Lives values) should be accurately displayed. |  |
| 8 | **[Functional]** Verify tapping on Score, Coins, Lives shows pop-ups. | Screen: Home (3). Action: Tap on Score (10), Coins (1), Lives (5/5). | Tapping should display the respective pop-up modals (Score & Goals (4), Coins (6), Lives (8)). |  |
| 9 | **[Integration]** Test tapping a subject card on Home navigates to quiz. | Screen: Home (3). Action: Tap "Answer Quiz" on the Mathematics card. | User should be navigated to the quiz screen for Mathematics Level 1. |  |
| 10 | **[Business]** Ensure core learning sections are accessible from Home. | Screen: Home (3). Components: Bookmark, Mathematics, Physics, Biology cards. | User should be able to initiate actions related to bookmarked questions and specific subjects from the main screen. |  |
| 11 | **[Unit]** Check if Score & Goals pop-up displays details correctly. | Screen: Score & Goals pop-up (4). Components: Quiz count, Streak, Today's/Year's score, day buttons. | The pop-up should show the current quiz progress, streak, and score figures accurately. |  |
| 12 | **[Functional]** Verify "Change Goal" button in pop-up navigates. | Screen: Score & Goals pop-up (4). Action: Tap "Change Goal". | User should be navigated to the Edit Daily Goals screen (5). |  |
| 13 | **[Functional]** Verify closing Coins pop-up works. | Screen: Coins pop-up (6). Action: Tap the 'X' icon. | The Coins pop-up should close, returning the user to the Home Screen. |  |
| 14 | **[Functional]** Verify closing Streak Status pop-up works. | Screen: Streak Status pop-up (7). Action: Tap the 'X' icon. | The Streak Status pop-up should close, returning the user to the Home Screen. |  |
| 15 | **[Functional]** Verify closing Lives Status pop-up works. | Screen: Lives Status pop-up (8). Action: Tap the 'X' icon. | The Lives Status pop-up should close, returning the user to the Home Screen. |  |
| 16 | **[Integration]** Test tapping Battle option from Menu navigates. | Screen: Learning Menu pop-up (9). Action: Tap "Battle" icon. | User should be navigated to the Battle Screen (10). |  |
| 17 | **[Functional]** Verify tapping "Start the Battle" on Battle screen navigates. | Screen: Battle (10). Action: Tap "Start the Battle". | User should be navigated to the Select Subject screen (11) for battle. |  |
| 18 | **[Integration]** Test selecting a subject for battle leads to mode selection. | Screen: Select Subject (11). Action: Tap "Mathematics". | The Battle Mode pop-up (12) should appear. |  |
| 19 | **[Functional]** Verify tapping "By Chapter" on Battle Mode pop-up navigates. | Screen: Battle Mode pop-up (12). Action: Tap "By Chapter". | User should be navigated to the Select Chapters screen (14). |  |
| 20 | **[Functional]** Verify tapping "Random Match" on Invite Friends pop-up initiates search. | Screen: Invite Friends pop-up (13). Action: Tap "Random Match". | The app should initiate a search process for a random opponent (frontend indicates loading/searching). |  |
| 21 | **[Functional]** Verify selecting chapters updates the count. | Screen: Select Chapters - Mathematics (14). Action: Check/uncheck chapter checkboxes. | The "Selected Chapters" count at the bottom should update to reflect the number of checked chapters. |  |
| 22 | **[Functional]** Verify tapping an answer option during Brain Battle. | Screen: Brain Battle (15/16). Action: Tap one of the answer options (A, B, C, D). | The selected option should be highlighted, and the app should process the answer (frontend logic). |  |
| 23 | **[Integration]** Test reaching end of Battle questions shows result. | Screen: Brain Battle (16 - Question 5/5). Action: Answer the final question. | The Battle Result pop-up (17) should appear after the last question is answered or time runs out. |  |
| 24 | **[Functional]** Verify tapping "Done" on Battle Result pop-up navigates. | Screen: Battle Result pop-up (17). Action: Tap "Done". | User should be navigated to the Quiz Report Card screen (18). |  |
| 25 | **[Business]** Ensure the Quiz Report Card provides detailed performance analysis. | Screen: Quiz Report Card (18, 19). Components: Scores, Accuracy, Correct/Incorrect answers, Time, Performance Metrics, Comparison. | The report should display comprehensive metrics about the battle performance and allow review of individual questions and overall trends. |  |
| 26 | **[Functional]** Verify tapping tabs on Overall Score screen switches views. | Screen: Overall Score (24). Action: Tap "Areas to Improve" tab. | The content area should switch to display "Areas that need improvement" and suggestions (Screen 25). |  |
| 27 | **[Business]** Ensure "Areas to Improve" highlights weak areas and suggests actions. | Screen: Areas to Improve (25). Components: Subject/Topic items with percentages, Improvement Suggestions, "Practice Now" button. | The screen should correctly identify subjects/topics with lower scores and provide relevant suggestions and a practice link. |  |
| 28 | **[Integration]** Test tapping "Practice Now" from Areas to Improve navigates. | Screen: Areas to Improve (25). Action: Tap "Practice Now". | User should be navigated to a practice session/quiz specifically for the highlighted topic (e.g., Geography - Map Reading). |  |
| 29 | **[Unit]** Check if Mathematics Subject Details screen renders metrics. | Screen: Mathematics (Subject Details) (26). Components: Progress circle, Pages Read, Study Time, Quizzes metrics. | The screen should display the user's progress metrics for the specific subject correctly. |  |
| 30 | **[Functional]** Verify expanding a chapter on Subject Details shows sub-topics. | Screen: Mathematics (Subject Details) (27). Action: Tap on "Chapter 1: Fundamentals of Algebra". | The chapter details should expand, showing the performance breakdown for sub-topics like "Basic Equations", "Polynomials", "Word Problems". |  |
| 31 | **[Integration]** Test tapping "Practice This Chapter" on Subject Details navigates. | Screen: Mathematics (Subject Details) (27). Action: Tap "Practice This Chapter". | User should be navigated to a practice session/quiz for the selected chapter. |  |
| 32 | **[Business]** Ensure Subject Details provides actionable insights for practice. | Screen: Mathematics (Subject Details) (27). Components: Chapter Performance, Practice This Chapter button, Recommendations. | The screen should clearly show chapter progress, provide an option to practice, and offer personalized recommendations. |  |
| 33 | **[Functional]** Verify tapping tabs on Leaderboard switches views. | Screen: LeaderBoard (28). Action: Tap "monthly" tab. | The leaderboard view should switch to display the monthly rankings. |  |
| 34 | **[Unit]** Check if Leaderboard displays top users and list correctly. | Screen: LeaderBoard (28). Components: Top 3 podium with flags/names/points, Ranked list below. | The podium and the list should render with user names, points, and ranks as expected. |  |
| 35 | **[Business]** Provide competitive element through weekly/monthly leaderboards. | Screen: LeaderBoard (28). Overall feature. | Users should be able to see how they rank against others on a weekly and monthly basis. |  |
| 36 | **[Unit]** Check if the Menu screen displays all options correctly. | Screen: Menu (29). Components: List items (Manage Account, Upgrade, etc.), Version info, Policy links. | All expected menu options and footer information should be visible. |  |
| 37 | **[Integration]** Test tapping a Menu item navigates to its respective screen. | Screen: Menu (29). Action: Tap "App Settings". | User should be navigated to the App Settings screen (not shown, assumed). This applies to all navigation items. |  |
| 38 | **[Integration]** Test tapping "Logout" initiates logout process. | Screen: Menu (29). Action: Tap "Logout". (Assume backend mocks logout success). | User should be logged out and navigated back to the Login or Welcome screen. |  |
| 39 | **[Functional]** Verify tapping on a notification item. | Screen: Notifications (30). Action: Tap on a notification item (e.g., "Congratulations! You received a new badge..."). | The notification item might expand, show details, or navigate to a relevant screen based on the notification type. |  |
| 40 | **[Business]** Ensure Notifications screen keeps users informed of events. | Screen: Notifications (30). Overall feature. | Users should be able to view past notifications about achievements, goals, etc. |  |