**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.

**5.1 Data Dictionary**

This table lists **alphabetically** the major system objects and their attributes.

**🔹 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**

**🔹 1. User Schema (Base Schema)**

js

CopyEdit

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 }

});

**🔹 2. Student Schema (Extends User)**

js

CopyEdit

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

});

**🔹 3. Parent Schema (Extends User)**

js

CopyEdit

const ParentSchema = new mongoose.Schema({

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

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

notifications: [String]

});

**🔹 4. Teacher Schema (Extends User)**

js

CopyEdit

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. Quiz Schema**

js

CopyEdit

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**

js

CopyEdit

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**

js

CopyEdit

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**

js

CopyEdit

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**

js

CopyEdit

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**

js

CopyEdit

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**

js

CopyEdit

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**

js

CopyEdit

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**

js

CopyEdit

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

}]

});

**1. Flashcard Retrieval and Display**

**Purpose**: To fetch flashcards for a selected book and chapter.

pseudo

CopyEdit

Algorithm GetFlashcards(bookID, chapterID)

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

2. return flashcards

**2. Peer Battle Matchmaking System**

**Purpose**: To connect a student with another available peer for a battle.

pseudo

CopyEdit

Algorithm MatchPeerForBattle(studentID)

1. waitingList ← GetWaitingPeers()

2. for each peer in waitingList do

3. if IsCompatible(peer, studentID) then

4. CreateBattleRoom(studentID, peer)

5. return "Matched with " + peer.name

6. return "No match found. Please wait."

**3. AI-Based Quiz Generation**

**Purpose**: To generate a quiz based on topic and difficulty.

pseudo

CopyEdit

Algorithm GenerateQuiz(chapterID, difficultyLevel)

1. allQuestions ← GetQuestions(chapterID)

2. suitable ← FilterByDifficulty(allQuestions, difficultyLevel)

3. quiz ← RandomSelect(suitable, 10)

4. return quiz

**4. Video-to-Notes Conversion**

**Purpose**: To generate transcript-based notes from video content.

pseudo

CopyEdit

Algorithm GenerateNotesFromVideo(videoID)

1. video ← LoadVideo(videoID)

2. transcript ← Transcribe(video)

3. notes ← Summarize(transcript)

4. return notes

**5. Goal Assignment & Tracking**

**Purpose**: To allow parents to assign goals to children and monitor progress.

pseudo

CopyEdit

Algorithm AssignGoal(childID, goalText, deadline)

1. goal ← CreateGoal(goalText, deadline)

2. AssignToChild(childID, goal)

3. return "Goal assigned"

**6. Live Class Enrollment**

**Purpose**: To enroll a student in a live class if space exists.

pseudo

CopyEdit

Algorithm EnrollInClass(studentID, classID)

1. class ← GetClass(classID)

2. if class.capacity > class.enrolled then

3. Enroll(studentID, classID)

4. return "Enrolled"

5. return "Class Full"

**7. Student Progress Calculation**

**Purpose**: To calculate progress in a subject based on completed flashcards, quizzes, and videos.

pseudo

CopyEdit

Algorithm CalculateProgress(studentID, subjectID)

1. totalTasks ← GetAllTasks(subjectID)

2. completed ← GetStudentCompletions(studentID, subjectID)

3. progress ← (completed / totalTasks) \* 100

4. return progress

**8. Reward Trigger System**

**Purpose**: Automatically issue rewards when goals are completed.

pseudo

CopyEdit

Algorithm CheckRewards(childID)

1. goals ← GetGoals(childID)

2. for goal in goals do

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

4. GiveReward(childID, goal.reward)

5. MarkRewardGiven(goal)

**9. Quiz Result Evaluation and Feedback**

**Purpose**: To calculate and display quiz result and provide feedback.

pseudo

CopyEdit

Algorithm EvaluateQuiz(quizID, studentAnswers)

1. correctAnswers ← GetCorrectAnswers(quizID)

2. score ← 0

3. for i in range(len(correctAnswers)) do

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

5. score += 1

6. feedback ← GenerateFeedback(score)

7. return score, feedback

**10. Parent Dashboard Data Aggregation**

**Purpose**: To summarize child performance, goals, and quiz stats.

pseudo

CopyEdit

Algorithm GetParentDashboard(parentID)

1. children ← GetChildren(parentID)

2. dashboard ← []

3. for child in children do

4. stats ← GetChildStats(child.id)

5. dashboard.append(stats)

6. return dashboard

**1. AI Flashcard Generation Algorithm**

**Purpose**: Automatically generate flashcards using AI from chapter text or transcript.

pseudo

CopyEdit

Algorithm GenerateAIFlashcards(chapterText)

1. keyPoints ← ExtractKeyConcepts(chapterText) // via NLP or transformer model

2. flashcards ← []

3. for point in keyPoints do

4. question ← GenerateQuestion(point)

5. answer ← GenerateAnswer(point)

6. flashcards.append((question, answer))

7. return flashcards

🔹 Uses NLP/transformer models to extract key concepts and form Q&A pairs.

**✅ 2. Recommendation Engine for Flashcards/Videos/Quizzes**

**Purpose**: Suggest relevant content based on past activity and performance.

pseudo

CopyEdit

Algorithm RecommendContent(userID)

1. history ← GetUserHistory(userID) // chapters viewed, scores, time spent

2. similarUsers ← FindSimilarUsers(userID)

3. preferredTopics ← AnalyzeWeakAreas(history)

4. recommended ← []

5. for topic in preferredTopics do

6. content ← FetchContent(topic)

7. recommended.append(content)

8. return recommended

🔹 Could use collaborative filtering or content-based filtering in the backend.

**✅ 3. Battle Matchmaking System (AI + Peer-Based)**

**Purpose**: Match users with peers for competitive battles.

pseudo

CopyEdit

Algorithm MatchForBattle(studentID)

1. userLevel ← GetUserSkillLevel(studentID)

2. availablePeers ← GetWaitingStudents()

3. matchedPeer ← null

4. for peer in availablePeers do

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

6. matchedPeer ← peer

7. break

8. if matchedPeer ≠ null then

9. CreateBattleRoom(studentID, matchedPeer)

10. return matchedPeer

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

🔹 This ensures fair battles based on student performance levels.

**✅ 5. Calculating Overall Books Score**

**Purpose**: To compute a student’s overall performance across all books.

pseudo

CopyEdit

Algorithm CalculateOverallScore(studentID)

1. books ← GetAllBooksAttempted(studentID)

2. totalScore ← 0

3. totalQuestions ← 0

4. for book in books do

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

6. totalScore += score

7. totalQuestions += questions

8. if totalQuestions == 0 then return 0

9. return (totalScore / totalQuestions) \* 100

🔹 Helps show cumulative progress in parent dashboards.

**✅ 6. Calculating Book-wise Score**

**Purpose**: To compute performance metrics for each book.

pseudo

CopyEdit

Algorithm CalculateBookWiseScores(studentID)

1. books ← GetAllBooksAttempted(studentID)

2. scores ← {}

3. for book in books do

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

5. if questions > 0 then

6. percentage ← (score / questions) \* 100

7. scores[book.title] ← percentage

8. return scores

🔹 Ideal for performance analytics per subject/book.

Algorithm RecommendContent(user\_id):

Input: user\_id

Output: recommended\_items

1. user\_profile ← GetUserPreferences(user\_id)

2. similar\_users ← FindSimilarUsersUsingCosineSimilarity(user\_profile)

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

4. collaborative\_recs ← RecommendFromSimilarUsers(similar\_users)

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

6. Return combined\_recs

**5. Explanation Generation by AI**

**Purpose:** Generate detailed, step-by-step explanations for difficult concepts.

plaintext

CopyEdit

Algorithm GenerateExplanation(topic\_text):

Input: topic\_text

Output: explanation

1. keywords ← ExtractKeyConcepts(topic\_text)

2. For each keyword:

step\_explanation ← UseGPTOrT5ToExplain(keyword)

explanation.append(step\_explanation)

3. Return explanation

**4. AI Summary Generation**

**Purpose:** Generate a brief summary of a topic using NLP.

plaintext

CopyEdit

Algorithm GenerateSummary(content):

Input: content

Output: summary

1. cleaned\_text ← PreprocessText(content)

2. summary ← RunPegasusOrT5Summarizer(cleaned\_text)

3. Return summary

**10. Goal Achievement Tracking and Reward Trigger**

plaintext

CopyEdit

Algorithm CheckAndRewardGoal(user\_id, goal\_id):

Input: user\_id, goal\_id

Output: reward\_given (boolean)

1. goal ← GetGoalDetails(goal\_id)

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

3. If progress ≥ goal.target:

reward ← AssignReward(user\_id)

Return True

4. Else:

Return False

| **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() |

| **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 |