Crowd-Sourced Competitive Exam Question Aggregator and Practicing Platform

Group - 3

IT Lab Group Project

Debjit Dhar (002210501106)
Soham Lahiri (002210501107)
Srinjoy Dutta (002210501122)
Chirantan Nath (101910501064)
Section A3

23 April 2025

Table of Contents

Requirement-Deliverable Mapping

Requirement	Deliverable
User Authentication: Users can sign up, log in, and contribute questions, options and explanations based on different exam categories.	Implemented role based authentication for users and admins. Admins have a superset of user functions which include verifying, editing and deleting existing questions
Question Bank: Aggregates user-submitted questions, categorized by subject and difficulty level.	Implemented filtering based on subject and difficulty. Added categories such as 'AI Generated' along with Keyword Searching.
Practice Mode: Users can attempt quizzes with instant feedback and explanations.	Quiz mode present, where users can take quizzes to practice.
Upvoting & Reporting (optional): Users can upvote good questions and report incorrect ones for moderation.	Implemented upvoting and downvoting for questions.
Leaderboard & Analytics (optional): Tracks user progress and ranks top contributors.	Implemented ranking of contributors based on most upvoted questions and most questions contributed.

Frontend

Technology Stack and Foundation

The QAggrPlatform frontend is built on a modern React ecosystem leveraging TypeScript for type safety and enhanced developer experience. The application utilizes Vite as its build tool and development server, providing fast hot module replacement and optimized production builds. The frontend adopts the latest React version (19.0.0) and incorporates Tailwind CSS for utility-first styling, enabling rapid UI development with consistent design patterns.

Core Architecture Components

Application Structure

1. The frontend follows a well-organized modular architecture with clear separation of concerns:
2. **Components Directory**: Contains reusable UI elements organized by functionality:
    1. auth/: Authentication-related components including ProtectedRoute for access control
    2. layout/: Structural components that define the application's visual framework
    3. questions/: Components for displaying and managing question data
    4. quizzes/: Components for quiz display, creation, and management
    5. Utility components like MarkdownRenderer.tsx for content formatting
3. **Pages Directory**: Contains full page components representing different application views:
    1. Homepage.tsx: Entry point for users with featured content
    2. QuestionDetailPage.tsx and QuestionsPage.tsx: For viewing and browsing questions
    3. QuizDetailPage.tsx and QuizzesPage.tsx: For viewing and browsing quizzes
    4. QuizAttemptPage.tsx: Interactive quiz-taking interface
    5. LeaderboardPage.tsx: Display of user rankings

6. Authentication pages (LoginPage.tsx, SignupPage.tsx)
7. User profile and admin management pages

4. **Services Directory**: Contains API interaction logic:
    1. api.ts: Central module defining all backend API endpoints with typed interfaces
    2. Implementation of request caching for performance optimization
    3. Organization of endpoints by functional domain (questions, quizzes, attempts, etc.)
5. **Contexts Directory**: Implements application-wide state management:
    1. AuthContext.tsx: Manages user authentication state, exposes login/logout methods
    2. Provides user information throughout the application
6. **Types Directory**: Contains TypeScript interface definitions for data models

7. Routing and Navigation

1. The application implements client-side routing using React Router v7, as defined in App.tsx. The routing system includes:
2. **Public Routes**: Accessible to all users (home, login, browse questions/ quizzes)
3. **Protected Routes**: Require authentication (create questions, attempt quizzes, profile)
4. **Admin Routes**: Limited to users with administrative privileges (quiz creation, verification)
5. **Error Handling**: Dedicated 404 page with redirect for non-existent routes
6. The routing implementation utilizes lazy loading through React's lazy() and Suspense to optimize initial load performance by code-splitting.

State Management and Data Flow

1. **Context API**: Used for global state management (authentication)
2. **Local Component State**: For UI-specific state within components
3. **Data Fetching**: Centralized in the api.ts service module with Axios
4. **Request Caching**: In-memory caching for GET requests to reduce redundant network calls
5. **Cache Invalidation**: Automatic clearing after mutating operations (POST/ PUT/DELETE)

UI/UX Implementation

The interface incorporates several modern design patterns:

1. **Responsive Design**: Tailwind CSS provides responsive utilities for different screen sizes
2. **Component Composition**: Building complex interfaces from smaller, reusable parts
3. **Loading States**: Skeleton loaders and spinners during async operations
4. **Error Handling**: User-friendly error messages and fallbacks
5. **Animations**: Subtle transitions for improved user experience
6. **Accessibility**: Semantic HTML and proper ARIA attributes

Advanced Features

1. **KaTeX Integration**: For mathematical notation rendering in questions
2. **Markdown Support**: Using react-markdown with remark and rehype plugins
3. **Form Validation**: Client-side validation before API submission
4. **Real-time Feedback**: Immediate response to user actions like voting
5. **Progress Tracking**: For quiz attempts and completions

API Integration

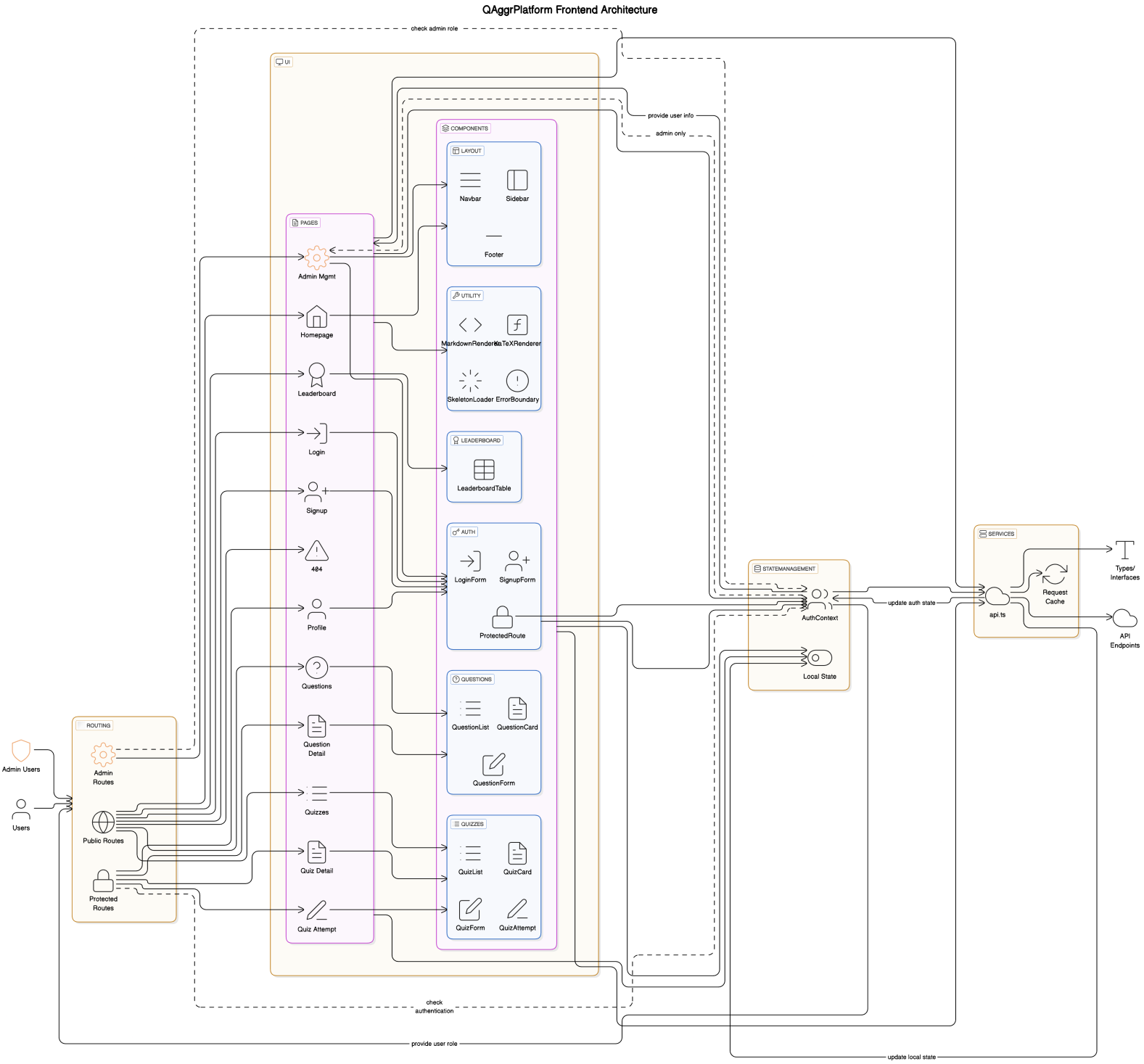
The frontend communicates with the backend through a comprehensive set of API endpoints organized by domain:

1. **Authentication**: Login, signup, logout, and session management
2. **Questions**: CRUD operations, search, filtering, and voting
3. **Quizzes**: Creation, management, and attempt tracking
4. **Subjects**: Browse and search functionality
5. **Users**: Profile information and statistics
6. **Leaderboards**: Rankings and achievements

API requests include proper error handling and loading state management to provide a seamless user experience even during network operations.

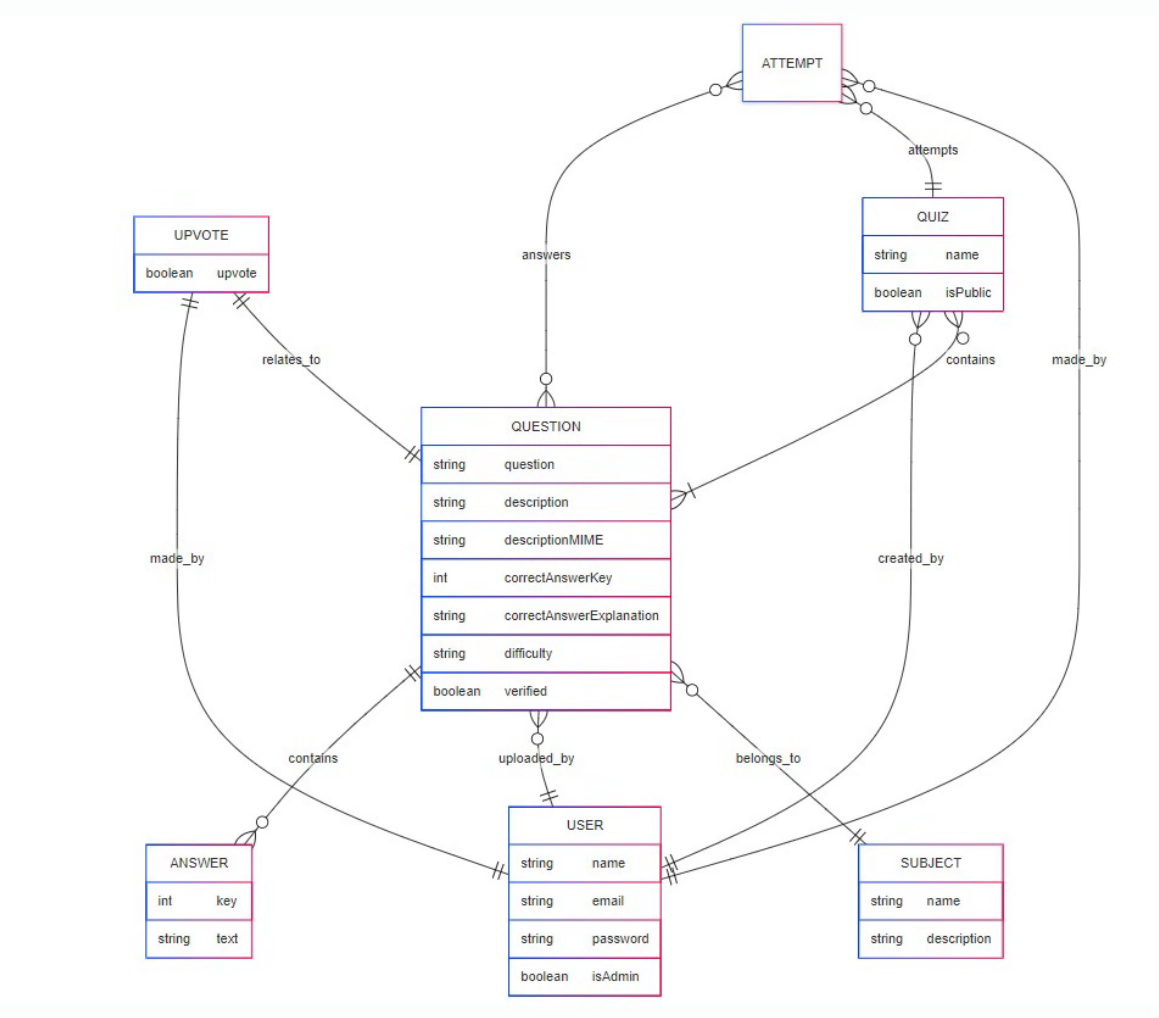
Performance Optimization

1. **Code Splitting**: Using React.lazy for on-demand loading of components
2. **Request Caching**: In-memory caching for frequently accessed data
3. **Suspense API**: For loading states during code splitting
4. **Optimized Rendering**: Careful component design to prevent unnecessary re-renders


QAggrPlatform Frontend Architecture
check admin role
UI
COMPONENTS
LAYOUT
Navbar
Sidebar
Footer
PAGES
Admin Mgmt
Homepage
Leaderboard
Login
Signup
404
Profile
Questions
Question Detail
Quizzes
Quiz Detail
Quiz Attempt
UTILITY
MarkdownRenderer
LaTeXRenderer
SkeletonLoader
ErrorBoundary
LEADERBOARD
LeaderboardTable
AUTH
LoginForm
SignupForm
ProtectedRoute
QUESTIONS
QuestionList
QuestionCard
QuestionForm
QUIZZES
QuizList
QuizCard
QuizForm
QuizAttempt
provide user info
admin only
STATEMANAGEMENT
AuthContext
Local State
update auth state
SERVICES
api.ts
Request Cache
Types/ Interfaces
API Endpoints
ROUTING
Admin Routes
Public Routes
Protected Routes
Admin Users
Users
check authentication
provide user role
update local state

Backend

Database Schema



MongoDB is used to manage the tables in a single database for the project. The above diagram is the E-R diagram for the project database.

Database Design

1. User Schema

```
const userSchema = new mongoose.Schema({
 name: { type: String, required: true },
 email: { type: String, required: true, unique: true },
 password: { type: String, required: true },
 isAdmin: { type: Boolean, required: true, default: false },
});
```

• Purpose: Stores user details.

• Fields:

o name: Full name of the user.

o email: Unique identifier and contact; must be unique.

o password: Hashed password.

o isAdmin: Boolean flag to distinguish normal users from admins.

2. Subject Schema

```
const subjectSchema = new mongoose.Schema({
 name: { type: String, required: true, trim: true, unique: true },
 description: { type: String, required: true, trim: true },
});
subjectSchema.index({ name: "text", description: "text" });
```

• Purpose: Represents the domain/category a question belongs to.

• Fields:

o name: Unique name of the subject.

o description: Overview of the subject.

• Indexing: Text index allows for full-text search on both name and description.

3. Answer Subdocument

```
const answerSchema = new mongoose.Schema({
 key: { type: mongoose.Schema.Types.Int32, required: true },
 text: { type: String, required: true, trim: true },
});
```

• Purpose: Defines each answer option within a question.

• Fields:

o key: Integer that identifies the option; unique only within its question.

o text: The actual answer text.

4. Question Schema

```
const questionSchema = new mongoose.Schema({
 question: { type: String, required: true, trim: true },
 description: { type: String, required: false, default: "" },
 descriptionMIME: { type: String, required: false, default: "text/plain" },
 subject: { type: mongoose.Schema.Types.ObjectId, ref: "Subject", required: true
},
 answers: [answerSchema],
 correctAnswerKey: { type: mongoose.Schema.Types.Int32, required: true },
 correctAnswerExplanation: { type: String, default: "" },
 uploader: { type: mongoose.Schema.Types.ObjectId, ref: "User", required:
true },
 difficulty: { type: String, enum: DIFFICULTY_LEVELS, required: true },
 verified: { type: Boolean, default: false, required: true },
});
```

```
questionSchema.index({ question: "text", description: "text" });
```

• Purpose: Represents individual questions.

• Fields:

o question: The main question statement.

o description: Additional info or background.

o descriptionMIME: MIME type for the description (e.g., HTML support).

o subject: Reference to Subject.

o answers: Array of possible answers (subdocuments).

o correctAnswerKey: The key of the correct answer.

o correctAnswerExplanation: Optional explanation.

o uploader: The user who created the question.

o difficulty: Enum value: EASY, MEDIUM, HARD.

o verified: If the question has been verified (moderated).

• Indexing: Text search on question and description.

5. Upvote Schema

```
const upvoteSchema = new mongoose.Schema({
 question: { type: mongoose.Schema.Types.ObjectId, ref: "Question", required:
true },
 user: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
 upvote: { type: Boolean, required: true },
}, {
 statics: {
   async countUpvotes(question_id) { ... },
   async countDownvotes(question_id) { ... },
   async countNetVotes(question_id) { ... }
 }
});
upvoteSchema.index({ question: 1, user: 1 }, { unique: true });
```

• Purpose: Tracks upvotes/downvotes on questions.

• Fields:

o question: Question being voted on.

o user: User who voted.

o upvote: true for upvote, false for downvote.

• Indexing:

o Unique index ensures a user can vote only once per question.

• Statics:

o countUpvotes, countDownvotes, and countNetVotes for voting stats.

6. Quiz Schema

```
const quizSchema = new mongoose.Schema({
 name: { type: String, required: true, unique: true, trim: true },
```

```
 questions: [{ type: mongoose.Schema.Types.ObjectId, ref: "Question", required:
true }],
 creator: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
 isPublic: { type: Boolean, required: true, default: false },
});
quizSchema.index({ name: "text" });
```

• Purpose: Represents a collection of questions.

• Fields:

o name: Unique quiz name.

o questions: Array of question references.

o creator: Admin user who created it.

o isPublic: Controls visibility.

• Indexing: Allows search by quiz name.

7. Attempt Schema

```
const attemptSchema = new mongoose.Schema({
 user: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
 quiz: { type: mongoose.Schema.Types.ObjectId, ref: "Quiz", required: true },
 answers: [{
   question: { type: mongoose.Schema.Types.ObjectId, ref: "Question", required:
true },
   answerKey: { type: mongoose.Schema.Types.Int32, required: true },
 }]
}, {
 timestamps: true,
 methods: {
   async countCorrectAnswers() { ... },
   async countIncorrectAnswers() { ... },
   async countUnanswered() { ... }
 }
});
```

• Purpose: Tracks a user's attempt at a quiz.

• Fields:

o user: The test-taker.

o quiz: Quiz being attempted.

o answers: List of answers with references to questions.

• Timestamps: Automatically adds createdAt and updatedAt.

• Methods:

o countCorrectAnswers, countIncorrectAnswers, countUnanswered: Useful for grading and analytics.

We'll apply the definitions of Normal Forms to our schema.

First Normal Form (1NF)

A relation is in 1NF if:

• It only contains atomic (indivisible) values.

• Each record is unique (no duplicate rows).

• Each field contains values of a single type.

Example: Question.answers

```
answers: [answerSchema] // each has { key: Int32, text: String }
```

This is **atomic** — each answer has exactly one key and one string text, not a comma-separated list or nested structure.

All our schemas meet:

• Unique document IDs (_id)

• Atomic field types (String, Boolean, ObjectId, etc.)

• Arrays of subdocuments (like answers) that themselves are atomic.

O**ur entire schema is in 1NF.**

Second Normal Form (2NF)

A relation is in 2NF if:

• It is in 1NF, and

• **No non-prime attribute is partially dependent on any candidate key.**

This applies **only to tables with composite primary keys**. In MongoDB (and Mongoose), documents use **object IDs**, so there's no composite key in practice.

Example: Attempt.answers

```
answers: [
 {
   question: ObjectId,
   answerKey: Int32
 }
]
```

• No field in Attempt is partially dependent on another composite key.

• Each piece of data in each document is **fully functionally dependent** on the _id of the document (its primary key).

No partial dependencies ⇒ 2NF.

Third Normal Form (3NF)

A relation is in 3NF if:

• It is in 2NF, and

• There is **no transitive dependency** (i.e., non-key → non-key → key).

Let's validate by example:

Example: Question Schema

```
{
```

```
 question: String,
 subject: ObjectId,
 uploader: ObjectId,
 difficulty: String,
 correctAnswerKey: Int32
}
```

• uploader refers to User, which stores name, email, etc.

• subject is a reference, no redundant subject info is copied here.

• There is **no field whose value depends on another non-key field.**

Example: Quiz

```
{
 name: String,
 creator: ObjectId,
 questions: [ObjectId]
}
```

• Quiz name doesn't determine the creator or vice versa.

• No field can be derived from another **non-key field**.

No transitive dependencies ⇒ **3NF.**

Endpoints and Specifications

We have developed the back-end component web application server for the question aggregation website, using **Node.js** and **Express.js** frameworks, and **MongoDB** as the database server solution. This web application exposes a RESTful API and provides for the following features:

1. **User Management**: The application has user authentication and authorization mechanisms in place, with features like user login, logout, registration, and query.
2. **Session Management**, which ties in with the previous feature to maintain state information of currently logged in user. This is done using the express-session middleware for Express.js, which maintains all session information that is required into a ***backing store on the server side (here we also use the MongoDB database as the backing store)***. This module just sends a session ID as an *HTTP only* cookie to the client browser; this session ID is used as a key into the backing store for user session information.
3. **Question Management**: The API allows for creating, reading, updating, and deleting questions. Questions can be searched and filtered by subject, difficulty level, and uploader (the user who originally uploaded the question). The web application also allows for **upvoting** or **downvoting** questions by other users, and a **question verification** mechanism for administrators of the website.

4. **Quizzes Management**: The API allows for users to create, search, update and delete quizzes consisting of a selected set of questions. It also allows for users to **attempt quizzes** multiple times and get feedback on their performance.
5. **Leaderboard**: Users can be ranked on total number of upvotes on all their questions or the number of verified questions they have uploaded.
6. Finally, administrators can manage (create, update and delete) the set of **subjects** that a question may belong to.

These are the REST API endpoints served by this application.

HTTP Method	URL	Description
User API		
POST	/user/signup	Sign up/registration of a new user
POST	/user/login	Logging in a new user
POST	/user/logout	User logging out
GET	/user1 /user/me /user/info	Query information about 'myself'/currently logged in user
GET	/user/<id>	Query information about user with given ID
PUT	/user/<id>	For administrator accounts, to make the user with the given ID an administrator as well.
Question API		
GET	/question? subject=<subject id>&difficulty=<difficulty	Search for questions given subject, difficulty level, and uploader (all
GET	/question/search? q=<query string>&subject=<subject id>&difficulty=<difficulty level>&uploader=<user	Search for questions given a query string that searches for matches in name or description of the question, optionally filtered by subject,
GET	/question/<id>	Get question details and contents given ID
GET	/question/ DIFFICULTY_LEVELS	Get all possible difficulty levels for a question.
POST	/question	User can post a new question
PUT	/question/<id>	User can edit/update a previously uploaded question, indicated by ID

DELETE	/question/<id>	User can delete a previously uploaded
PUT	/question/<id>/verify	Administrator users can verify a question to make it public.
Subject API		
GET	/subject	Get ALL subjects (list of IDs)
GET	/subject/search?q=<query string>	Search for a subject for a text matches of query string with name and description of a subject.
GET	/subject/<id>	Get information on a subject given ID
POST	/subject	Administrators can add a new subject
DELETE	/subject/<id>	Administrators can delete a subject given ID
PUT	/subject/<id>	Administrators can update a subject given
Voting API		
POST	/voting/<question id>/upvote	For users to upvote a question indicated by ID
POST	/voting/<question id>/downvote	For users to downvote a question indicated by ID
DELETE	/voting/<question id>	For users to *unvote* a question (neither upvote or downvote) indicated
GET	/voting/<question id>	Get upvote and downvote count for a question indicated by ID.
Quizzes API		
POST	/quiz	User can create a new quiz
PUT	/quiz/<id>	User can update a quiz given ID
DELETE	/quiz/<id>	User can delete a quiz given ID
POST	/quiz/<quiz id>/questions/<question id>	User can add a new question (indicated by question ID) to a quiz indicated by quiz ID.
DELETE	/quiz/<quiz id>/questions/<question id>	User can delete a question (indicated by question ID) from a quiz indicated by quiz ID.
GET	/quiz	Get ALL quizzes (a list of IDs)

GET	/quiz/search?q=<query string>	Search for quizzes by searching for text matches of query string in quiz names.
GET	/quiz/<id>	Get contents and details of quiz given ID.
Quiz Attempts API		
GET	/attempt?quiz_id=<quiz id>	Get logged in user's all attempts on a quiz
GET	/attempt/<id>	Get attempt given attempt ID
POST	/attempt	Post a new attempt of a quiz by logged in user
POST	/attempt/<attempt id>/ answers/<question id>	Post a new answer to a question (indicated by question id) of a quiz indicated by quiz ID
DELETE	/attempt/<attempt id>/ answers/<question id>	Delete (un-answer) an attempted question's answer (question indicated by question ID) of a quiz indicated by
DELETE	/attempt/<id>	Delete entire attempt indicated by ID
Leaderboard API		
GET	/leaderboard/verified	Rank users in terms of number of verified questions they have
GET	/leaderboard/ totalUpvotes	Rank users in terms of the number of total upvotes they have received in all of their uploaded and verified

LLM Integration

The goal was to build a service that programmatically generates *mutated multiple-choice questions (MCQs)* based on existing ones. The new questions should maintain the same subject and difficulty but slightly alter the numerical values or wording. This ensures content diversity without manually writing each new question, which is time-consuming and error-prone.

Solution Approach

To automate the generation of mutated MCQs, we integrated an LLM (Language Model) provided by **Cohere** using the **langchain** library. The workflow is as follows:

1. **Fetch Original Question**: Retrieve a question from MongoDB using Mongoose ORM, including its subject, difficulty, and answer choices.
2. **Prompt Template:** Define a system prompt that instructs the LLM to slightly mutate the original question while preserving its core attributes.
3. **LLM Invocation:** Send the prompt along with the original question details to the Cohere LLM (command-r-plus) via the ChatCohere chain.
4. **Response Parsing:** Extract the valid JSON response containing the new question.
5. **Validation & Creation:** Validate required fields, map enums like difficulty, and save the new question to the database with a reference to its source.

Design Pattern

This service leverages a Chain of Responsibility pattern, as used in the LangChain framework, which passes input through various transformation layers:

• **LLM Wrapper (LangChain's ChatCohere):** Handles LLM invocation and retry logic.

• **Prompt Template:** A reusable format that ensures consistency in generated questions.

• **Error Handling Blocks:** Try-catch logic ensures resilience against parsing and generation issues.

• **Factory Logic:** Dynamically builds a new question document based on structured AI output.

Additionally, the function is modular and idempotent, designed as a standalone service that can be invoked via API or internal process.

Future Scope

The QAggrPlatform architecture presents substantial opportunities for technical expansion, with its current Express.js/React/MongoDB stack providing a solid foundation for advanced capabilities. The LLM integration could be enhanced through implementation of a multi-model orchestration layer, enabling dynamic selection between providers (OpenAI, Cohere, Anthropic) based on cost, latency, and performance metrics. This would require developing a robust adapter pattern for API normalization and implementing token-aware caching strategies to optimize rate limits and reduce operational costs. A vector database integration (e.g., Pinecone, Weaviate) could enable semantic search and retrieval for question similarity detection, while fine-tuning domain-specific models on educational content would improve generation quality for specialized subjects.

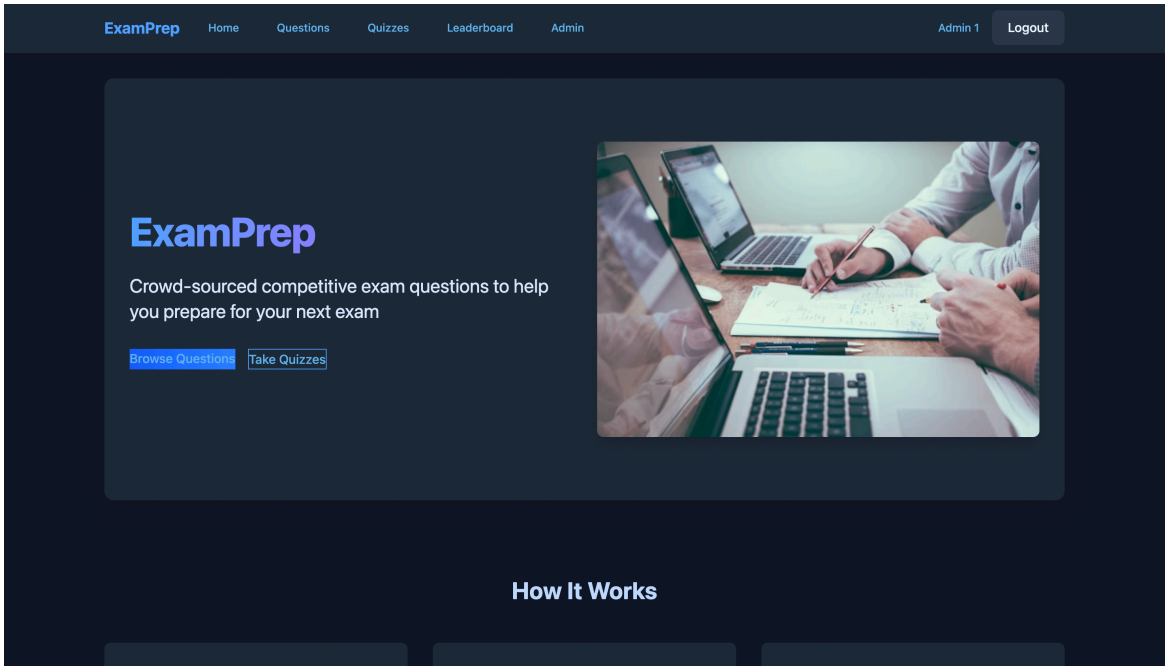
System architecture could evolve toward a more scalable microservices approach, with individual services for authentication, question management, quiz orchestration, and analytics communicating via message brokers (Kafka/RabbitMQ). This would facilitate horizontal scaling during peak usage periods and enable more granular deployment of updates. Implementing a CQRS (Command Query Responsibility Segregation) pattern would optimize read/write operations, while event sourcing would provide comprehensive audit trails for all system modifications. Real-time collaboration would benefit from WebSocket integration with conflict resolution protocols, possibly using Operational Transformation or Conflict-free Replicated Data Types (CRDTs) for managing concurrent edits to shared resources.

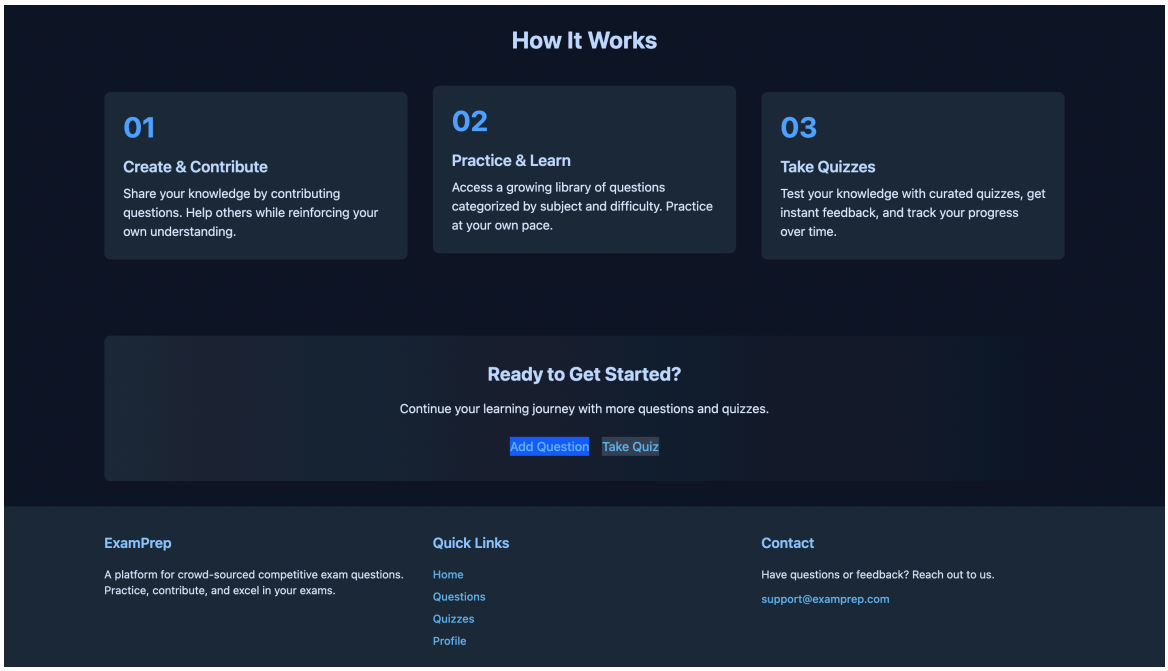
Performance enhancements could include implementing GraphQL with Apollo Client to reduce over-fetching and enable more efficient data loading patterns. Edge caching with CDN integration would improve global response times, while server-side rendering (SSR) with Next.js would optimize initial page loads and SEO. Database performance could be enhanced through strategic implementation of read replicas, query optimization, and time-series data partitioning for analytics storage.

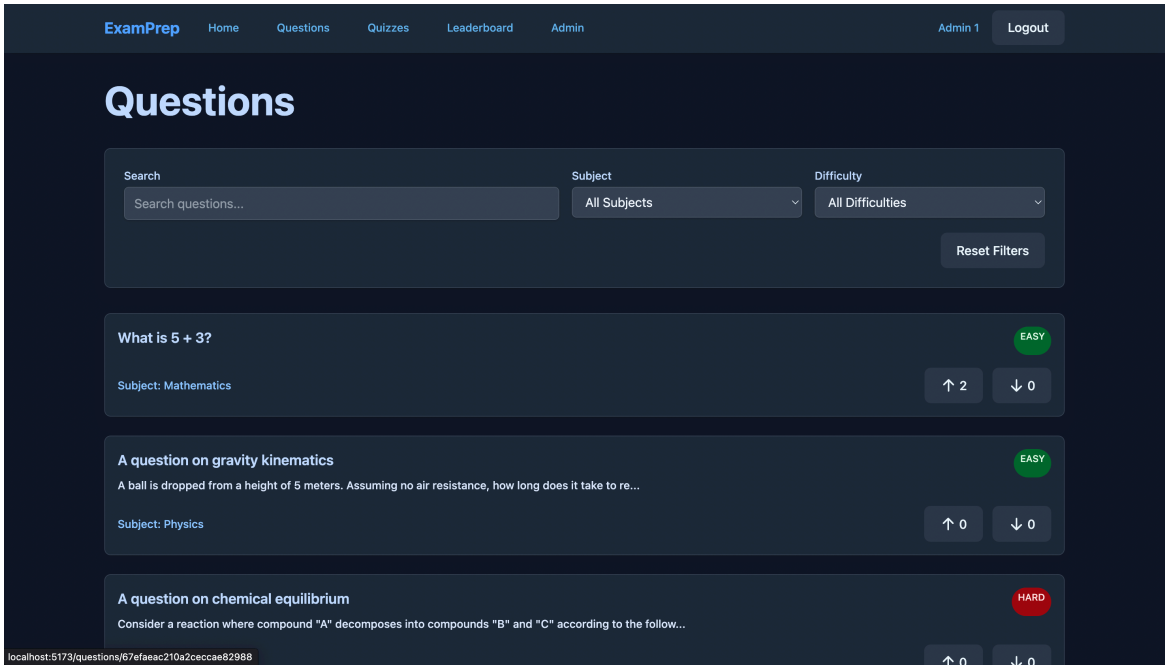
The backend could implement advanced algorithmic components, including Bayesian knowledge tracing for adaptive difficulty adjustment, and natural language processing pipelines for automated assessment of free-text responses. These would require development of training pipelines, model versioning systems, and A/B testing frameworks to validate effectiveness. Database schema evolution would benefit from implementation of migration frameworks (like Prisma or TypeORM) to manage structural changes while maintaining data integrity.

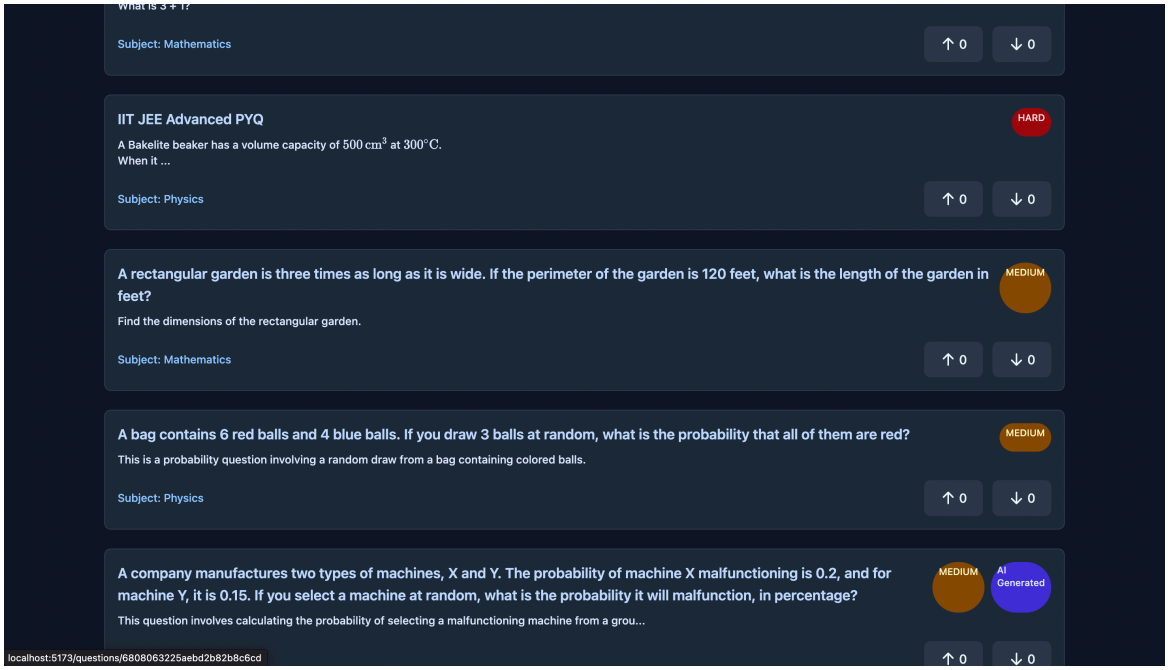
Security enhancements would include implementing OAuth 2.0 with PKCE for authentication. End-to-end encryption for sensitive content, implementation of rate limiting and CSRF protection, and regular security testing through automated vulnerability scanning would address evolving threat models. These technical improvements would collectively transform QAggrPlatform from a basic quiz system into a sophisticated, scalable educational technology infrastructure capable of supporting enterprise-level deployments while maintaining security, performance, and extensibility.

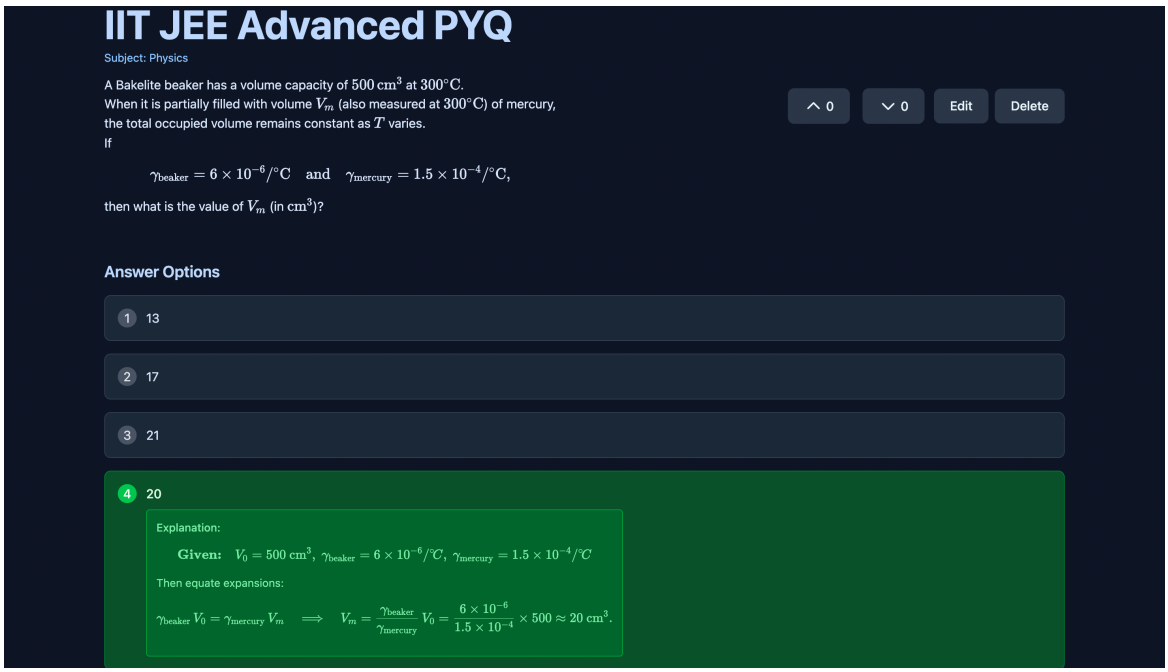
Screenshots

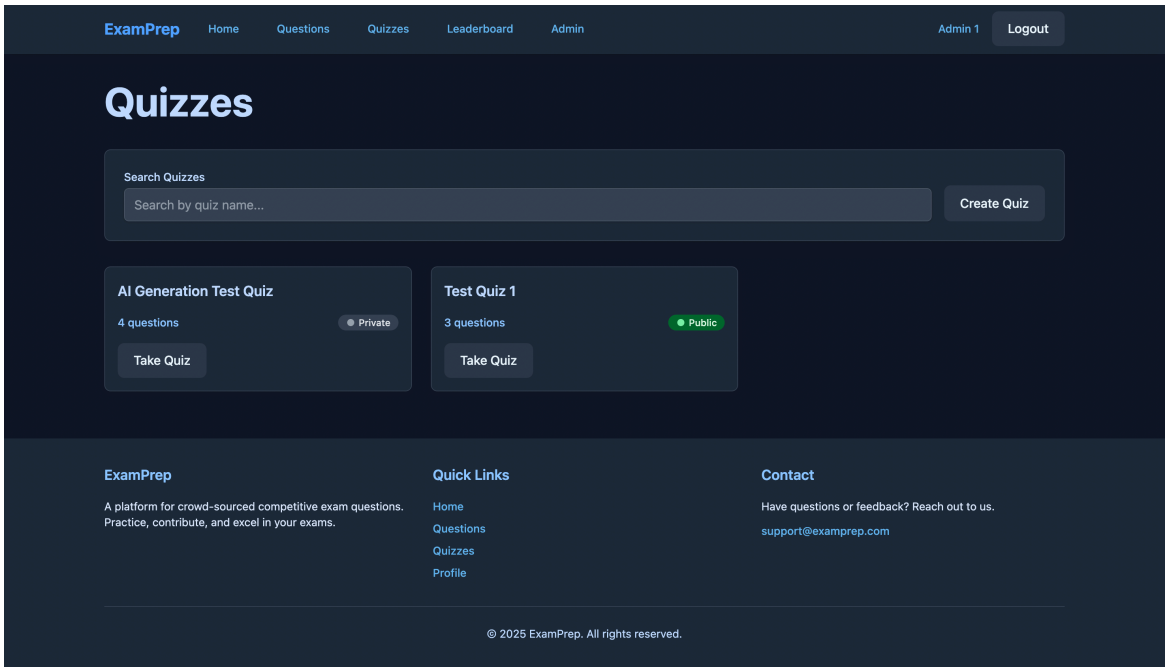


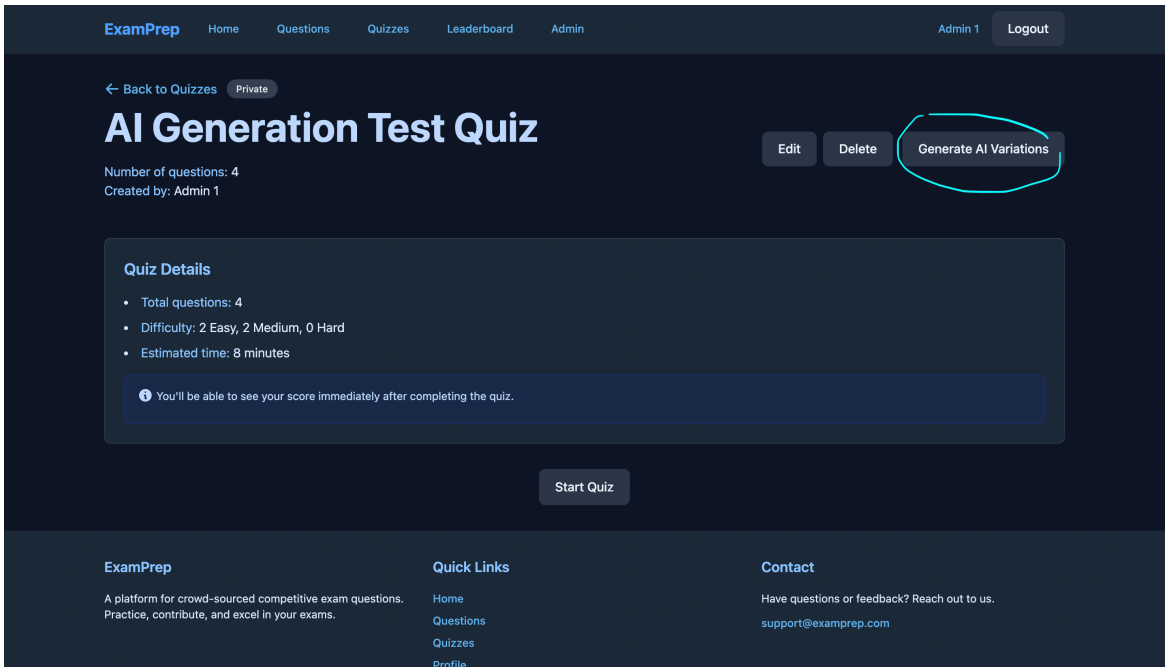


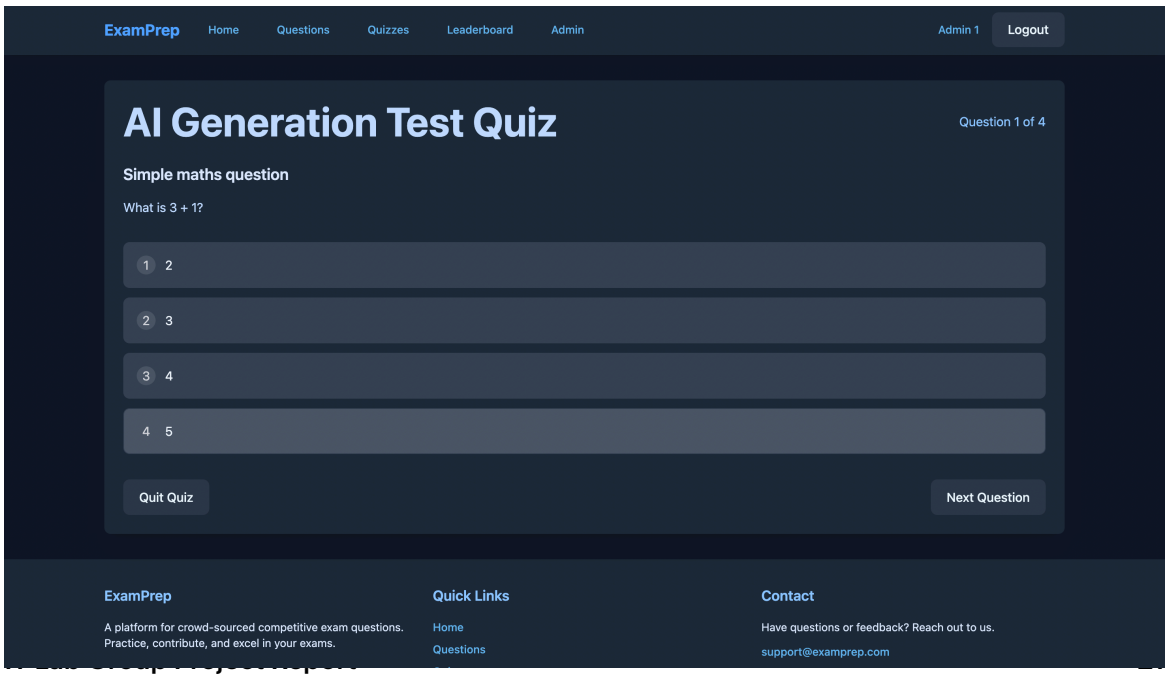

ExamPrep
Home
Questions
Quizzes
Leaderboard
Admin
Admin 1
Logout
Questions
Search
Search questions...
Subject
All Subjects
Difficulty
All Difficulties
Reset Filters
What is 5 + 3?
EASY
Subject: Mathematics
↑ 2
↓ 0
A question on gravity kinematics
EASY
A ball is dropped from a height of 5 meters. Assuming no air resistance, how long does it take to re...
Subject: Physics
↑ 0
↓ 0
A question on chemical equilibrium
HARD
Consider a reaction where compound "A" decomposes into compounds "B" and "C" according to the follow...
localhost:5173/questions/67efaeac210a2ceccae82988

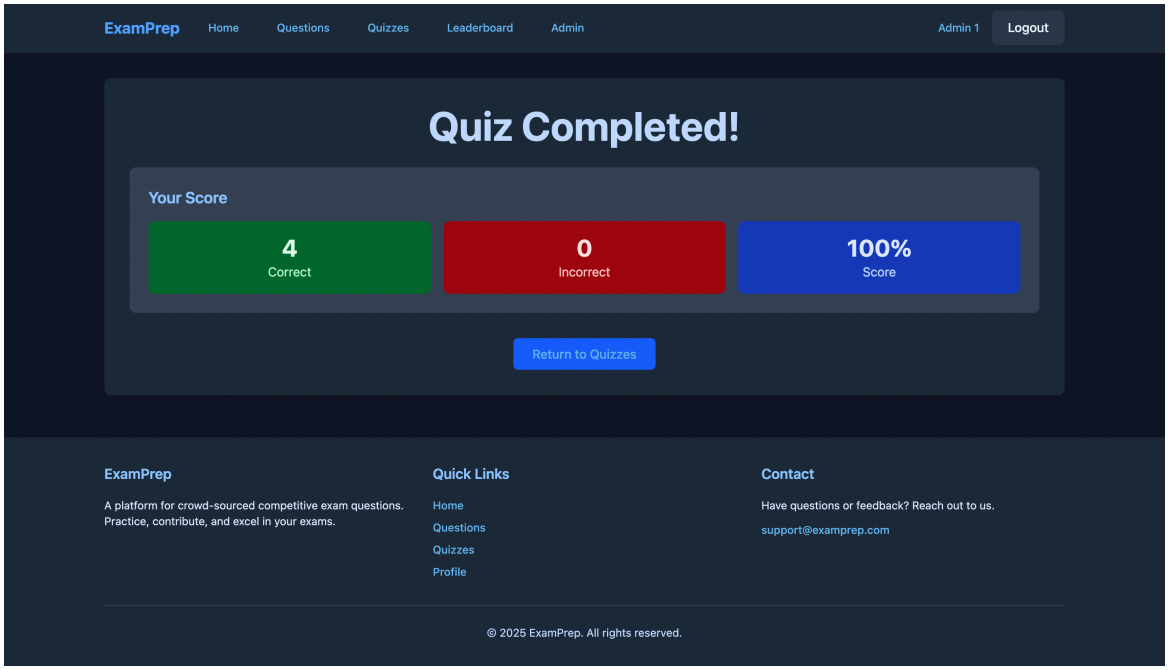

Subject: Mathematics
↑ 0
↓ 0
IIT JEE Advanced PYQ
HARD
A Bakelite beaker has a volume capacity of 500 cm³ at 300°C.
When it ...
Subject: Physics
↑ 0
↓ 0
A rectangular garden is three times as long as it is wide. If the perimeter of the garden is 120 feet, what is the length of the garden in feet?
MEDIUM
Find the dimensions of the rectangular garden.
Subject: Mathematics
↑ 0
↓ 0
A bag contains 6 red balls and 4 blue balls. If you draw 3 balls at random, what is the probability that all of them are red?
MEDIUM
This is a probability question involving a random draw from a bag containing colored balls.
Subject: Physics
↑ 0
↓ 0
A company manufactures two types of machines, X and Y. The probability of machine X malfunctioning is 0.2, and for machine Y, it is 0.15. If you select a machine at random, what is the probability it will malfunction, in percentage?
MEDIUM
AI Generated
This question involves calculating the probability of selecting a malfunctioning machine from a grou...
localhost:5173/questions/6808063225aebd2b82b8c6cd

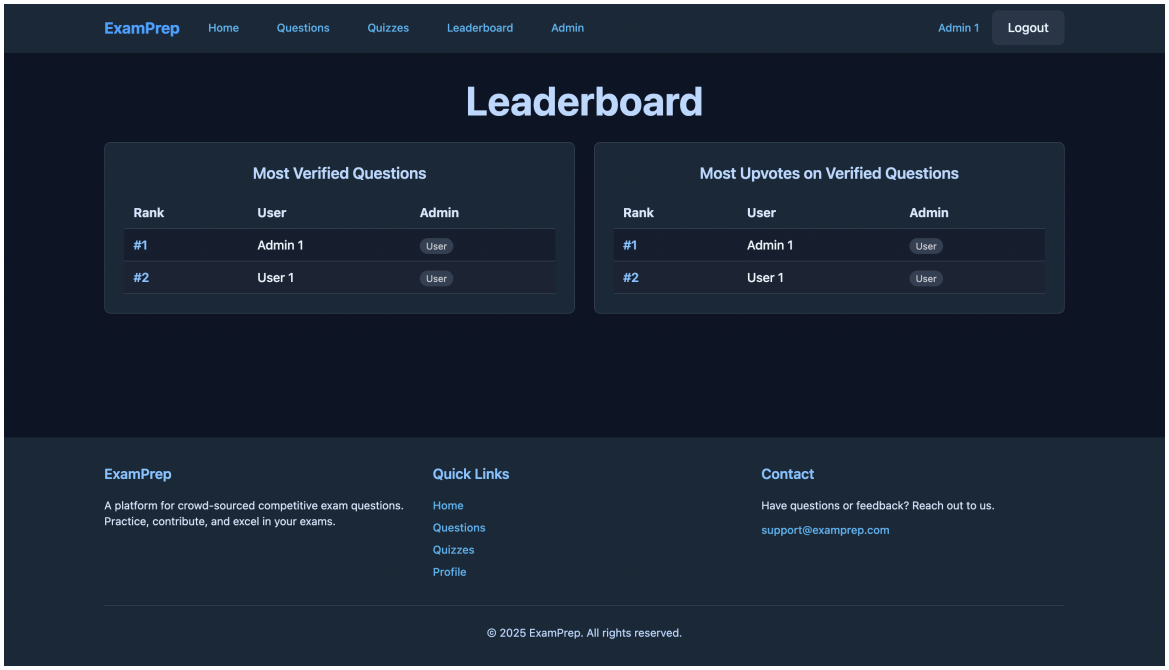

IIT JEE Advanced PYQ
Subject: Physics
A Bakelite beaker has a volume capacity of 500 cm³ at 300°C.
When it is partially filled with volume V_m (also measured at 300°C) of mercury, the total occupied volume remains constant as T varies.
If
$\gamma_{\text{beaker}} = 6 \times 10^{-6}/^\circ\text{C}$ and $\gamma_{\text{mercury}} = 1.5 \times 10^{-4}/^\circ\text{C}$,
then what is the value of V_m (in cm³)?
^ 0
v 0
Edit
Delete
Answer Options
1 13
2 17
3 21
4 20
Explanation:
Given: $V_0 = 500\ \text{cm}^3$, $\gamma_{\text{beaker}} = 6 \times 10^{-6}/^\circ C$, $\gamma_{\text{mercury}} = 1.5 \times 10^{-4}/^\circ C$
Then equate expansions:
$\gamma_{\text{beaker}} V_0 = \gamma_{\text{mercury}} V_m \implies V_m = \frac{\gamma_{\text{beaker}}}{\gamma_{\text{mercury}}} V_0 = \frac{6 \times 10^{-6}}{1.5 \times 10^{-4}} \times 500 \approx 20\ \text{cm}^3.$

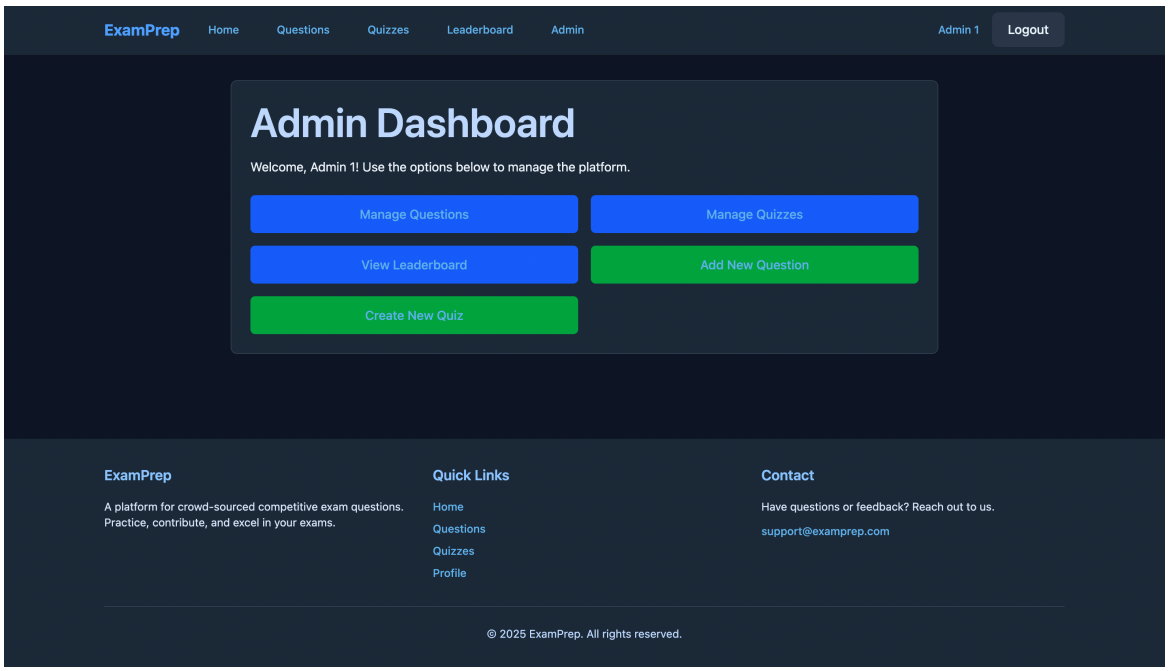

ExamPrep
Home
Questions
Quizzes
Leaderboard
Admin
Admin 1
Logout
Quizzes
Search Quizzes
Search by quiz name...
Create Quiz
AI Generation Test Quiz
4 questions
Private
Take Quiz
Test Quiz 1
3 questions
Public
Take Quiz
ExamPrep
A platform for crowd-sourced competitive exam questions. Practice, contribute, and excel in your exams.
Quick Links
Home
Questions
Quizzes
Profile
Contact
Have questions or feedback? Reach out to us.
support@examprep.com
© 2025 ExamPrep. All rights reserved.


ExamPrep
Home
Questions
Quizzes
Leaderboard
Admin
Admin 1
Logout
Back to Quizzes
Private
AI Generation Test Quiz
Number of questions: 4
Created by: Admin 1
Edit
Delete
Generate AI Variations
Quiz Details
Total questions: 4
Difficulty: 2 Easy, 2 Medium, 0 Hard
Estimated time: 8 minutes
You'll be able to see your score immediately after completing the quiz.
Start Quiz
ExamPrep
A platform for crowd-sourced competitive exam questions. Practice, contribute, and excel in your exams.
Quick Links
Home
Questions
Quizzes
Contact
Have questions or feedback? Reach out to us.
support@examprep.com


ExamPrep
Home
Questions
Quizzes
Leaderboard
Admin
Admin 1
Logout
AI Generation Test Quiz
Question 1 of 4
Simple maths question
What is 3 + 1?
1 2
2 3
3 4
4 5
Quit Quiz
Next Question
ExamPrep
A platform for crowd-sourced competitive exam questions. Practice, contribute, and excel in your exams.
Quick Links
Home
Questions
Contact
Have questions or feedback? Reach out to us.
support@examprep.com


ExamPrep
Home
Questions
Quizzes
Leaderboard
Admin
Admin 1
Logout
Quiz Completed!
Your Score
4
Correct
0
Incorrect
100%
Score
Return to Quizzes
ExamPrep
A platform for crowd-sourced competitive exam questions. Practice, contribute, and excel in your exams.
Quick Links
Home
Questions
Quizzes
Profile
Contact
Have questions or feedback? Reach out to us.
support@examprep.com
© 2025 ExamPrep. All rights reserved.


ExamPrep
Home
Questions
Quizzes
Leaderboard
Admin
Admin 1
Logout
Leaderboard
Most Verified Questions
Rank
User
Admin
#1
Admin 1
User
#2
User 1
User
Most Upvotes on Verified Questions
Rank
User
Admin
#1
Admin 1
User
#2
User 1
User
ExamPrep
A platform for crowd-sourced competitive exam questions. Practice, contribute, and excel in your exams.
Quick Links
Home
Questions
Quizzes
Profile
Contact
Have questions or feedback? Reach out to us.
support@examprep.com
© 2025 ExamPrep. All rights reserved.


ExamPrep
Home
Questions
Quizzes
Leaderboard
Admin
Admin 1
Logout
Admin Dashboard
Welcome, Admin 1! Use the options below to manage the platform.
Manage Questions
Manage Quizzes
View Leaderboard
Add New Question
Create New Quiz
ExamPrep
A platform for crowd-sourced competitive exam questions. Practice, contribute, and excel in your exams.
Quick Links
Home
Questions
Quizzes
Profile
Contact
Have questions or feedback? Reach out to us.
support@examprep.com
© 2025 ExamPrep. All rights reserved.