## Problem Definition

In today’s competitive environment, students, graduates, and working professionals often struggle to make informed career choices due to the lack of accessible, personalized, and structured guidance. Existing career portals are either too generic, difficult to navigate, or fail to cater to individual user needs such as academic background, skills, and interests.

Moreover, the absence of interactive tools, such as interest-based quizzes, tailored recommendations, and engaging multimedia content, makes career exploration a daunting process. Many individuals remain unaware of suitable career paths, trending job opportunities, or the skills required to achieve their goals.

There is a clear need for a dynamic, responsive, and scalable web application that simplifies career exploration, provides personalized career recommendations, and offers interactive resources in an engaging manner. Such a solution should work seamlessly across devices, ensure user-friendliness, and provide role-based guidance (students, graduates, professionals) while supporting secure data management and administrative control.

## Design Specifications

The **PathSeeker – Career Passport Web Application** is a **full-stack web solution** that uses React on the frontend, Express.js on the backend, and MongoDB as the database. The design focuses on responsiveness, scalability, and personalized career exploration for students, graduates, professionals, and administrators.

### 1. System Architecture

#### Frontend (Client-Side):

* + Developed using **HTML5, CSS3, custom CSS, JavaScript, and React.js**.
  + Provides modular and reusable UI components for dashboards, quizzes, career bank, and resource center.
  + Ensures responsive design with media queries and user-friendly navigation.

#### Backend (Server-Side):

* + Built on **Express.js** running on Node.js.
  + Manages routing, authentication, quiz logic, bookmarking, and API endpoints.
  + Provides secure communication between frontend and database.

#### Database Layer:

* + **MongoDB** is used to store application data in collections.
  + Stores users, careers, quizzes, results, feedback, resources, and bookmarks.
  + Schema-less structure allows flexible storage and scalability.

#### Deployment:

* + Application can run locally on Node.js server or be deployed on cloud platforms such as Heroku, Render, or AWS.

### 2. Major Modules and Components

#### User Authentication & Management

* + Registration and login for Students, Graduates, Professionals, and Admin.
  + Profile creation with education, skills, and interests.
  + Secure session handling and password reset functionality.

#### Personalized Dashboard

* + Displays quiz results, bookmarked resources, and recommendations.
  + Provides career suggestions based on interaction history.

#### Career Bank

* + Collection of careers stored in MongoDB.
  + Search and filtering by domain, skills, salary, or demand.
  + Saves user preferences for customized exploration.

#### Interest Quiz

* + Interactive React-based quiz with multiple question types.
  + Suggests relevant career options based on responses.
  + Stores history of quiz attempts for progress tracking.

#### Resource & Multimedia Center

* + Embedded videos, downloadable PDFs, and infographics.
  + Categorized by type and audience for easy navigation.

#### Success Stories Hub

* + Card-based display of career journeys.
  + Users can submit stories, which are reviewed by the admin.

#### Feedback System

* + Feedback form to collect queries, suggestions, or bug reports.
  + Data stored in MongoDB and reviewed by admin.

#### Bookmarking & Notes

* + Users can bookmark careers and resources.
  + Notes can be added to bookmarks and exported.

#### Admin Panel

* + Manage users, careers, quiz content, resources, and feedback.
  + Approve submitted stories and view usage statistics.

### 3. Database Design (MongoDB Collections)

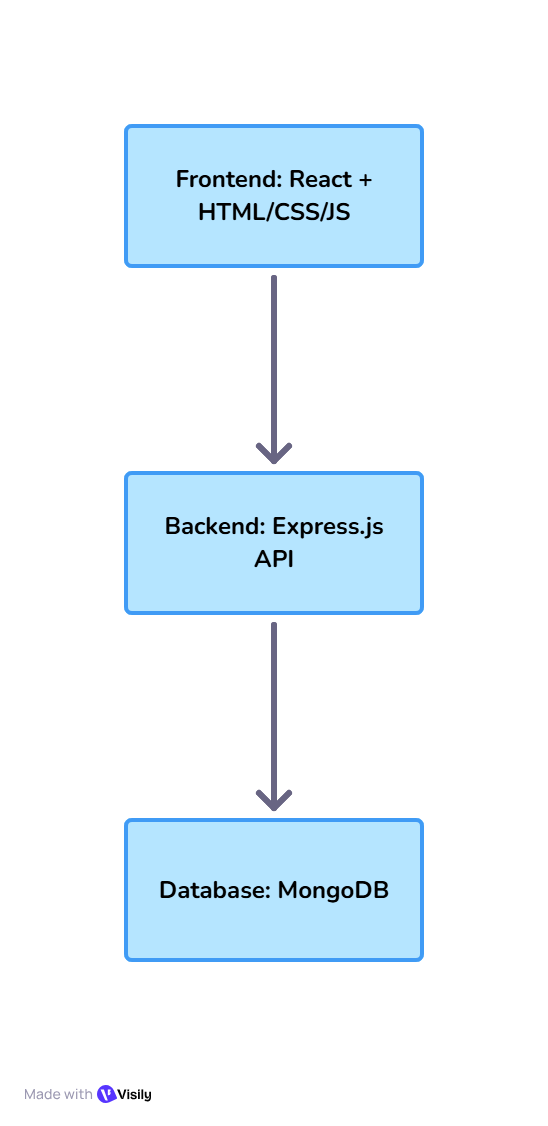
* **Users** → { userID, role, name, email, password, education, skills, interests }
* **Careers** → { careerID, title, domain, description, salaryRange, demandLevel }
* **Quizzes** → { quizID, question, options[], type, weightage }
* **QuizResults** → { resultID, userID, quizID, score, suggestedCareers[] }
* **Resources** → { resourceID, type, title, filePath, tags }
* **Stories** → { storyID, userID, title, description, status }
* **Feedback** → { feedbackID, userID, type, message, date }
* **Bookmarks** → { bookmarkID, userID, itemID, note, type }

### 4. Design Principles

* **Security:** Encrypted password storage, session management, and safe API access.
* **Scalability:** Modular React components and flexible MongoDB collections for future expansion.
* **Performance:** Optimized Express.js APIs and efficient MongoDB queries.
* **Responsiveness:** Mobile-friendly layouts and smooth navigation.
* **Usability:** Simple menus, breadcrumbs, and interactive forms for better user experience.

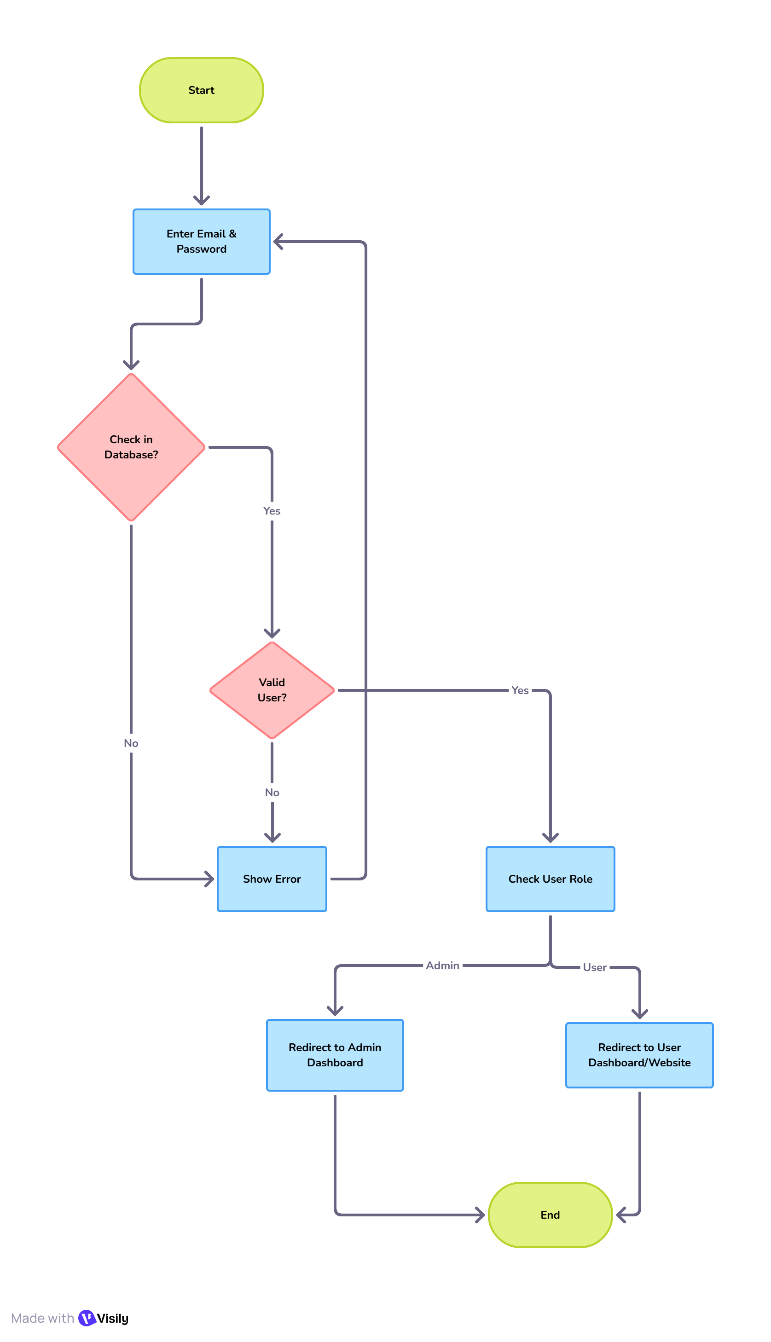
## Diagrams for PathSeeker Web Application

### 1. System Architecture Diagram

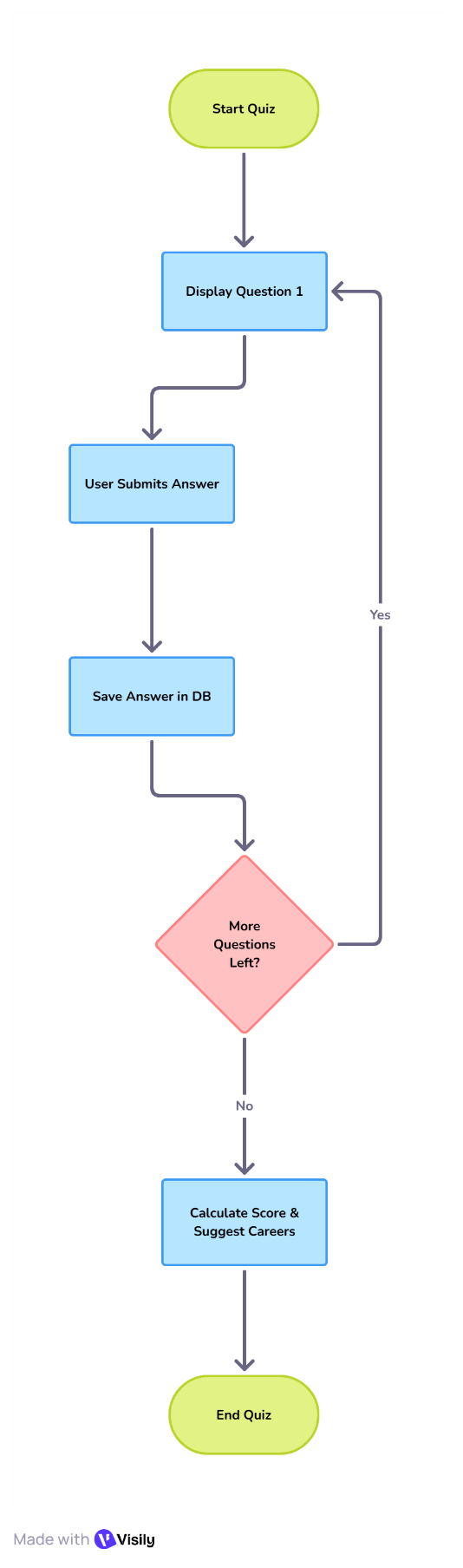


* **Frontend (React)** → Collects user input, displays dashboards, quizzes, career bank, and resources.
* **Backend (Express.js)** → Processes requests, handles authentication, quiz logic, and data management.
* **Database (MongoDB)** → Stores users, careers, quizzes, resources, bookmarks, and feedback.

### 2. Flowchart – User Login Activity



### 3. Flowchart – Career Quiz Activity



## Database Design (MongoDB)

The **PathSeeker** web application uses **MongoDB** as a NoSQL database, which allows for flexible, document-based data storage. The following collections are defined to manage various aspects of the system, including user data, career information, resources, quizzes, and multimedia content.

### 1. Admins Collection

This collection stores administrator details who manage the system.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| admin\_id (PK) | ObjectId | Primary key, unique identifier for admin |
| name | String | Admin’s name |
| email | String | Admin’s email address |
| password\_hash | String | Hashed password for authentication |

### 2. Careers Collection

Stores information about various career options available in the system.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| career\_id (PK) | ObjectId | Primary key, unique identifier for the career |
| title | String | Title of the career (e.g., Software Engineer) |
| description | String | Career description |
| domain | String | Career domain or industry |
| required\_skills | Array | List of skills required for the career |
| education\_path | String | Suggested education path for the career |
| expected\_salary | Number | Expected salary range for the career |

### 3. Users Collection

Stores user information, including roles like student, graduate, and professional.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| user\_id (PK) | ObjectId | Primary key, unique identifier for the user |
| uname | String | Username for the user |
| email (unique) | String | User's email address (must be unique) |
| password\_hash | String | Hashed password for secure login |
| role | String | Role of the user (e.g., student, graduate, or professional) |

### 4. Resources Collection

Stores various learning and reference resources (PDFs, videos, etc.).

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| resource\_id (PK) | ObjectId | Primary key, unique identifier for the resource |
| title | String | Title of the resource |
| category | String | Category of the resource (e.g., Guides, Articles) |
| description | String | Description of the resource |
| file\_url | String | URL link to the resource file |
| tag | Array | Tags associated with the resource |
| views\_count | Number | Number of views of the resource |
| created\_by (FK) | ObjectId | Foreign key referencing the admin who created the resource |

### 5. SuccessStories Collection

Stores career success stories submitted by users.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| story\_id (PK) | ObjectId | Primary key, unique identifier for the story |
| rname | String | Name of the person in the success story |
| domain | String | Domain or industry of the success story |
| story\_text | String | The full text of the story |
| image\_url | String | URL to the story image |
| submitted\_by (FK) | ObjectId | Foreign key referencing the user who submitted the story |
| approved\_by (FK) | ObjectId | Foreign key referencing the admin who approved the story |
| approved\_at | Date | Date and time when the story was approved |

### 6. UserProfiles Collection

Stores detailed profile information for users.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| profile\_id (PK) | ObjectId | Primary key, unique identifier for the user profile |
| user\_id (FK) | ObjectId | Foreign key referencing the user |
| education\_level | String | Education level of the user (e.g., Bachelor's) |
| interests | Array | List of career interests |
| profile\_image | String | URL to the user's profile image |
| updated\_at | Date | Date when the profile was last updated |

### 7. Multimedia Collection

Stores information about multimedia content such as videos and podcasts.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| media\_id (PK) | ObjectId | Primary key, unique identifier for the media |
| title | String | Title of the multimedia content |
| type | String | Type of the media (e.g., Video, Podcast) |
| url | String | URL link to the multimedia content |
| tags | Array | Tags associated with the media |
| transcript | String | Transcript of the media |
| rating\_avg | Number | Average rating of the media |
| rating\_count | Number | Number of ratings received |

### 8. QuizQuestions Collection

Stores questions for the career quiz.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| question\_id (PK) | ObjectId | Primary key, unique identifier for the question |
| question\_text | String | Text of the quiz question |
| type | String | Type of the question (e.g., Multiple Choice) |
| options | Array | List of options for the question |
| correct\_answer | String | Correct answer from the options |
| weightage | Number | Weightage of the question in the quiz |

### 9. Feedback Collection

Stores feedback submitted by users.

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| feedback\_id (PK) | ObjectId | Primary key, unique identifier for the feedback |
| user\_id (FK) | ObjectId | Foreign key referencing the user |
| category | String | Category of feedback (bug, suggestion, etc.) |
| message | String | The feedback message |
| status | String | Status of the feedback (pending, resolved) |
| submitted\_at | Date | Date and time when the feedback was submitted |

## Project Installation Instructions

The **PathSeeker** web application is a full-stack project designed to assist users in career exploration. The following instructions will guide you through the process of installing and setting up the application on your local machine and server.

### 1. Prerequisites

Before you begin, ensure you have the following installed:

* **Node.js** (preferably version 14.x or later)
* **MongoDB** (for local database)
* **npm** (Node Package Manager)
* **Git** (optional for version control)
* **Code Editor** (Visual Studio Code, Sublime Text, etc.)

### 2. Cloning the Project

To get started, you need to clone the project repository. If you are using Git, run the following command in your terminal:

git clone <repository\_url>

If you don’t have the repository, download the ZIP file and extract it.

### 3. Installing Dependencies

Navigate to the project directory in your terminal. For example:

cd PathSeeker

Run the following command to install the required dependencies:

npm install

This will install both frontend and backend dependencies listed in the package.json file.

### 4. Setting up MongoDB

Ensure that MongoDB is running on your local machine or set up a cloud database like **MongoDB Atlas**. For local setup:

1. **Install MongoDB** (if you haven’t done it already) by following the instructions on [MongoDB's official website](https://www.mongodb.com/try/download/community).
2. **Start MongoDB** by running the following command:

mongod

This will start the MongoDB server on the default port (27017). You can change this port in the MongoDB configuration if needed.

### 5. Setting up Environment Variables

In the project directory, create a .env file for environment variables, such as the database connection URL and other sensitive credentials. Example:

DB\_URI=mongodb://localhost:27017/pathseeker

PORT=5000

SECRET\_KEY=your\_secret\_key

Make sure to replace the DB\_URI and SECRET\_KEY with your specific configurations.

### 6. Running the Application

To start the **backend server**, use the following command in your terminal:

npm run dev

This will run the **Express.js** backend, and the server will be accessible at http://localhost:5000 by default.

For the **frontend** (if using React), open another terminal window and navigate to the client folder:

cd client

npm start

This will start the **React** development server, and the frontend will be accessible at http://localhost:3000.

### 7. Accessing the Application

Once the backend and frontend servers are running, open a web browser and navigate to http://localhost:3000 to view the **PathSeeker** application.

You should be able to log in (as a student, graduate, or professional) or access other features depending on the user role you configure in your database.

### 8. Stopping the Application

To stop the servers, use the following commands in the terminal where you started each process:

* **For backend (Express.js):**

CTRL + C

* **For frontend (React):**

CTRL + C

If you're using **MongoDB locally**, stop the database with:

CTRL + C

### 9. Deployment

Once the application is working locally, you can deploy it to a cloud platform such as **Heroku**, **AWS**, or **DigitalOcean** for production use. This typically involves:

1. Pushing your code to a Git repository (e.g., GitHub).
2. Setting up your cloud platform to host both your frontend and backend.
3. Configuring environment variables for cloud deployment.

## User Credentials for Testing

### 1. Admin Credentials

* **Username**: admin1
* **Email**: admin@example.com
* **Password**: admin@123 (hashed in the database)

### 2. Student Credentials

* **Username**: student1
* **Email**: student@example.com
* **Password**: student@123 (hashed in the database)

### 3. Graduate Credentials

* **Username**: graduate1
* **Email**: graduate@example.com
* **Password**: graduate@123 (hashed in the database)

### 4. Professional Credentials

* **Username**: professional1
* **Email**: professional@example.com
* **Password**: professional@123 (hashed in the database)