# Globetrotter - The Ultimate Travel Guessing Game

# **Project Documentation**

This document provides comprehensive documentation for both the frontend and backend of the Globetrotter application - a travel destination guessing game.

# **Table of Contents**

- 1. Project Overview
- 2. Backend Documentation
  - Architecture
  - o API Endpoints
  - Database Models
  - Authentication
  - o Game Logic
  - o OpenAl Integration
- 3. Frontend Documentation
  - o Architecture
  - o Component Structure
  - o State Management
  - Routing
  - o API Integration
- 4. Installation and Setup
- 5. Testing
- 6. Deployment

# **Project Overview**

Globetrotter is a full-stack web application that challenges users to guess famous destinations based on cryptic clues. The application incorporates:

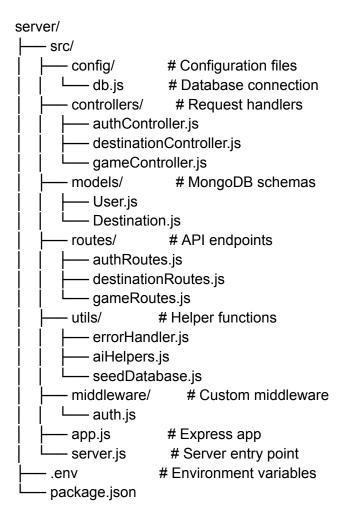
- User authentication and profile management
- Random destination selection with clues
- Multiple-choice destination guessing
- Score tracking and game statistics
- Social sharing for challenging friends
- A database with 100+ destinations

The project uses a Node.js/Express backend with MongoDB, and a React frontend with Tailwind CSS.

# **Backend Documentation**

#### **Backend Architecture**

The backend follows a Model-View-Controller (MVC) architecture:



#### **Dependencies**

Express: Web framework
Mongoose: MongoDB ODM
bcryptjs: Password hashing
jsonwebtoken: Authentication
OpenAI: AI for dataset expansion
cors: Cross-origin resource sharing

dotenv: Environment variables

# **API Endpoints**

## Authentication

Endpoint	Method	Description	Auth Required
/api/auth/register	POST	Register new user	No
/api/auth/login	POST	Login user	No
/api/auth/me	GET	Get user profile	Yes
/api/auth/stats/:username	GET	Get user stats	No

## Game

Endpoint	Method	Description	Auth Required
/api/game/destination	GET	Get random destination	Yes
/api/game/answer	POST	Submit answer	Yes
/api/game/challenge	POST	Generate challenge	Yes

# **Destinations (Admin)**

Endpoint	Method	Description	Auth Required
/api/destinations	GET	Get all destinations	Yes
/api/destinations/:id	GET	Get single destination	Yes
/api/destinations	POST	Create destination	Yes
/api/destinations/:id	PUT	Update destination	Yes
/api/destinations/:id	DELETE	Delete destination	Yes
/api/destinations/import	POST	Import destinations	Yes

#### **Database Models**

```
User Model
                          // Required, unique
 username: String,
                         // Hashed, required
 password: String,
 gameStats: {
  totalGames: Number,
                            // Default: 0
  correctAnswers: Number, // Default: 0
  incorrectAnswers: Number, // Default: 0
  score: Number
                        // Default: 0
 },
 recentGames: [{
                        // Limited to 10
  destinationId: ObjectId,
  correct: Boolean,
  playedAt: Date
 }],
 createdAt: Date
Destination Model
 name: String,
                       // Required, unique
 country: String,
                       // Required
                       // Required, enum of continents
 continent: String,
 clues: [{
  text: String,
                    // Required
  difficulty: String
                     // easy, medium, hard
 }],
 funFacts: [String],
                       // Required
 trivia: [String],
 imageURL: String,
                         // Optional
 popularityScore: Number, // 1-10
 createdAt: Date,
 updatedAt: Date
```

#### **Authentication**

#### 1. Registration:

- User submits username/password
- Password is hashed with bcrypt
- JWT token is generated and returned

#### 2. Login:

- o User submits credentials
- o Password is verified against hash
- JWT token is generated and returned

#### 3. Protection:

- JWT token is included in Authorization header
- o Token is verified in auth middleware
- User is attached to request object
- 4. **Session**: Stateless JWT authentication with 30-day expiration

# **Game Logic**

#### 1. Random Destination:

- Select a random destination from database
- Choose 1-2 random clues
- Generate 3 random incorrect answers
- o Randomize answer order

#### 2. Answer Validation:

- Check if answerld matches destination ID
- Update user stats (score, correct/incorrect counts)
- Return game result with fun fact

#### 3. Challenge System:

- Generate unique challenge ID
- o Create challenge URL with user stats
- Return challenge details for sharing

# **OpenAl Integration**

The backend uses OpenAI's API to expand the destination dataset:

#### 1. Data Transformation:

- o Starter dataset is converted to desired schema
- Existing destinations are tracked to avoid duplicates

#### 2. Al Generation:

- o OpenAl generates new destinations with proper structure
- o Each generation includes name, country, continent, clues, fun facts, trivia

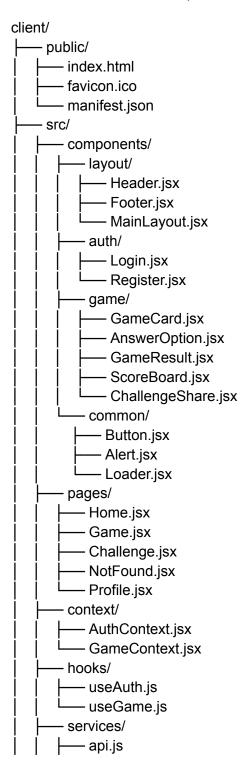
#### 3. Validation:

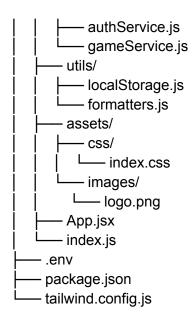
- o Generated data is validated against schema
- o Enriched with additional metadata

# **Frontend Documentation**

#### **Frontend Architecture**

The frontend follows a component-based architecture using React:





#### **Dependencies**

• **React**: UI library

• React Router: Navigation

• Axios: API requests

• Framer Motion: Animations

• React Confetti: Victory animations

• React Share: Social sharing

• html2canvas: Generate shareable images

• Tailwind CSS: Styling

• React Toastify: Notifications

# **Component Structure**

#### **Layout Components**

• MainLayout: Page wrapper with header and footer

• **Header**: Navigation bar with auth state display

• Footer: Site footer with info

#### **Page Components**

• Home: Landing page with game info

• Game: Core gameplay component

• Login/Register: Authentication forms

• **Profile**: User statistics and game history

• Challenge: Challenge acceptance page

#### **Game Components**

- GameCard: Displays destination clues
- AnswerOption: Individual answer choice
- GameResult: Shows game outcome and fun facts
- ScoreBoard: Displays user stats
- ChallengeShare: Challenge generation modal

#### **State Management**

The application uses React Context API for global state management:

#### **AuthContext**

Manages authentication state:

- User data
- Login/register/logout functions
- Authentication status
- Loading/error states

```
const AuthContext = createContext();
export const AuthProvider = ({ children }) => {
 const [user, setUser] = useState(null);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState(null);
 // Auth functions: login, register, logout
 return (
    <AuthContext.Provider value={{</pre>
      user, loading, error, isAuthenticated: !!user,
      login, register, logout, updateUser
      {children}
    </AuthContext.Provider>
  );
};
// Custom hook
export const useAuth = () => useContext(AuthContext);
```

#### **GameContext**

Manages game state:

- Current game data
- Game results
- Game statistics
- Game control functions

```
const GameContext = createContext();
export const GameProvider = ({ children }) => {
 const [currentGame, setCurrentGame] = useState(null);
 const [gameResult, setGameResult] = useState(null);
 const [loading, setLoading] = useState(false);
 const [error, setError] = useState(null);
 const [gameStats, setGameStats] = useState({...});
 // Game functions: loadGame, answerQuestion, resetGame
 return (
    <GameContext.Provider value={{</pre>
      currentGame, gameResult, loading, error, gameStats,
     loadGame, answerQuestion, resetGame
   }}>
      {children}
    </GameContext.Provider>
  );
};
// Custom hook
export const useGame = () => useContext(GameContext);
```

# **Routing**

React Router handles application navigation:

```
<Router>
  <AuthProvider>
    <GameProvider>
      <MainLayout>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />
          <Route path="/game" element={</pre>
            <ProtectedRoute>
              <Game />
            </ProtectedRoute>
          } />
          <Route path="/profile" element={</pre>
            <ProtectedRoute>
              <Profile />
            </ProtectedRoute>
          } />
          <Route path="/challenge/:id" element={<Challenge />} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </MainLayout>
    </GameProvider>
  </AuthProvider>
</Router>
```

Protected routes redirect unauthenticated users to the login page.

## **API Integration**

API requests are handled through service modules:

#### api.js

Central Axios instance with interceptors for token handling:

```
const api = axios.create({
 baseURL: process.env.REACT_APP_API_URL,
 headers: { 'Content-Type': 'application/json' }
});
// Add auth token to requests
api.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
 if (token) {
    config.headers.Authorization = `Bearer ${token}`;
 return config;
});
api.interceptors.response.use(
 response => response,
 error => {
    if (error.response?.status === 401) {
     localStorage.removeItem('token');
     localStorage.removeItem('user');
     window.location.href = '/login';
    return Promise.reject(error);
```

#### authService.js

Handles authentication operations:

```
export const register = async (username, password) => {
  const response = await api.post('/auth/register', { username, password
});
```

```
if (response.data.success) {
    localStorage.setItem('token', response.data.token);
    localStorage.setItem('user', JSON.stringify(response.data.data));
 return response.data;
};
export const login = async (username, password) => {
  const response = await api.post('/auth/login', { username, password });
 if (response.data.success) {
    localStorage.setItem('token', response.data.token);
    localStorage.setItem('user', JSON.stringify(response.data.data));
 return response.data;
};
export const logout = () => {
 localStorage.removeItem('token');
 localStorage.removeItem('user');
};
```

#### gameService.js

Handles game operations:

```
export const getRandomDestination = async () => {
  const response = await api.get('/game/destination');
  return response.data;
};

export const submitAnswer = async (gameId, answerId) => {
  const response = await api.post('/game/answer', { gameId, answerId });
  return response.data;
};

export const generateChallenge = async () => {
  const response = await api.post('/game/challenge');
  return response.data;
};
```

# **Installation and Setup**

# **Prerequisites**

- Node.js (14.x or higher)
- MongoDB (local or Atlas)
- OpenAl API key (for destination dataset expansion)

## **Backend Setup**

Clone the repository:

git clone https://github.com/your-repo/globetrotter.git cd globetrotter

1. Install dependencies:

```
cd server
npm install
```

2. Configure environment variables: Create .env file in the server directory:

```
PORT=5000

NODE_ENV=development

MONGO_URI=mongodb+srv://<username>:<password>@cluster.mongodb.net/globetrot
ter

JWT_SECRET=your_jwt_secret_key
JWT_EXPIRE=30d

OPENAI_API_KEY=your_openai_api_key
```

3. Seed the database:

```
npm run seed
```

4. Start the server:

npm run dev

## **Frontend Setup**

Install dependencies:

cd client
npm install

1. Configure environment variables: Create .env file in the client directory:

REACT\_APP\_API\_URL=http://localhost:8080/api

2. Start the development server:

npm start

3. The application should now be running at http://localhost:3000.

# **Testing**

# **Backend Testing**

Run the backend test suite:

cd server npm test

The backend uses Jest for unit testing controllers, models, and middleware.

# **Frontend Testing**

Run the frontend test suite:

cd client npm test

The frontend uses Jest and React Testing Library for component testing.