**Globetrotter Challenge - Project Documentation**

**1. Project Requirements**

**1.1 Overview**

Globetrotter is a full-stack web application that challenges users to identify famous destinations based on cryptic clues. Once users make a guess, they unlock interesting facts and trivia about the destination. The game features a social component allowing users to challenge friends to beat their score.

**1.2 Core Requirements**

**1.2.1 Dataset and AI Integration**

The application requires an extensive database of destination information with the following specifications:

* **Dataset Structure:** Each destination must include:
  + City name
  + Country name
  + Multiple cryptic clues (minimum 2 per destination)
  + Multiple fun facts (minimum 2 per destination)
  + Multiple trivia items (minimum 2 per destination)
* **Dataset Size:** Expand the starter dataset to include at least 100 unique destinations from around the world.
* **AI Utilization:** Leverage AI tools (such as OpenAI API, ChatGPT, or AI-assisted web scraping) to generate destination data that is engaging, accurate, and appropriately cryptic.
* **Data Storage:** All destination data must be stored on and retrieved from the backend to prevent users from accessing answers through client-side code inspection.

**1.2.2 Core Game Functionality**

The gameplay must include the following essential features:

* **Clue Presentation:** Display 1-2 randomly selected clues from a chosen destination without revealing its identity.
* **Answer Selection:** Allow users to select from multiple possible destination answers in a user-friendly interface.
* **Feedback System:** Provide immediate and engaging feedback after each guess:
  + *Correct Answer:* Display an animated confetti celebration and reveal a fun fact about the destination.
  + *Incorrect Answer:* Show a sad-face animation and reveal a fun fact about the correct destination.
* **Continuation Mechanics:** Include a clearly visible "Play Again" or "Next" button to load a different random destination.
* **Scoring System:** Track and display the user's total score, showing both correct and incorrect answers.

**1.2.3 "Challenge a Friend" Feature**

The application must include social functionality with these requirements:

* **User Registration:** Allow users to enter a unique username before inviting friends, creating a profile within the system.
* **Sharing Mechanism:** Implement a "Challenge a Friend" button that:
  + Opens a share popup with a dynamically generated image
  + Creates a unique invitation link for sharing via WhatsApp or other platforms
  + The image may be generated using on-screen UI elements or third-party services
* **Social Context:** Show the inviting user's score to the invited friend before they begin playing.
* **Universal Access:** Ensure that anyone with the invitation link can access the game with all features enabled.

**1.3 Evaluation Criteria**

The submission will be evaluated based on the following criteria:

* **Extensibility:** The solution must be designed to scale beyond the stated requirements, allowing for future enhancements.
* **UI/UX Design:** The game should be visually engaging and fun to play (note: this is less critical for backend role applicants).
* **AI Utilization:** The solution should demonstrate effective use of AI to complement development skills, particularly in dataset generation.
* **Additional Considerations:** The project may include extra elements that demonstrate technical excellence:
  + Unit test cases
  + Well-structured documentation
  + Code quality and organization
  + Creative extensions to the core functionality
* **Problem-Solving Initiative:** The solution should demonstrate the ability to make autonomous decisions when facing ambiguous requirements.

**1.4 Future Considerations**

The solution should be designed with extensibility in mind, anticipating potential future features such as:

* Time-based gameplay mechanics
* Image-based clues
* Difficulty levels
* Achievement systems
* Localization support

**2. Technical Requirements**

**2.1 Architecture Requirements**

* **Full-Stack Implementation:** The solution must include both frontend and backend components.
* **Backend Storage:** All game data must be stored server-side to maintain game integrity.
* **API Design:** Create clean API endpoints for game functionality including:
  + Retrieving random destinations and clues
  + Processing user guesses
  + Managing user profiles and scores
  + Generating and processing challenge links

**2.2 Deployment Requirements**

* **Public Repository:** The solution must be submitted via a public GitHub repository.
* **Web Hosting:** The application must be deployed to a publicly accessible hosting platform (Railway, Vercel, Netlify, etc.).
* **Documentation:** The repository must include a comprehensive README detailing setup instructions and technical choices.
* **Demo Video:** A 3-5 minute Loom walkthrough demonstrating:
  + Application functionality
  + Technical stack and architecture
  + Detailed explanation of at least one feature implementation

**2.3 Development Constraints**

* **Time Investment:** The solution should be completable in approximately 3 hours using AI-assisted programming techniques.
* **Code Quality:** The codebase should reflect professional standards despite the time constraint.

**3. Design Decisions**

**3.1 Dataset Generation Strategy: AI-Generated vs. Web-Scraped Content**

**3.1.1 Approach Comparison**

After evaluating multiple approaches for generating the required dataset of 100+ destinations, we determined that AI-generated content provides significant advantages over web scraping for the Globetrotter Challenge. This section documents the rationale behind this architectural decision.

**3.1.2 Justification for AI-Generated Content**

The decision to use AI-generated content rather than web scraping is based on several critical factors that directly impact the game's quality, sustainability, and user experience:

**Content Consistency and Quality Control**

* AI generation allows us to maintain consistent formatting and structure across all destinations, ensuring uniform data quality.
* Each entry follows the exact same schema with precisely crafted clues, fun facts, and trivia tailored to our gameplay requirements.
* Web scraping would yield inconsistent content requiring extensive post-processing to standardize and would likely contain information gaps.

**Content Customization for Gameplay Mechanics**

* The game requires specialized content formats (cryptic clues of appropriate difficulty) that rarely exist in this form on public websites.
* AI generation allows precise control over content characteristics, including:
  + Clue obscurity and difficulty levels
  + Fun fact interestingness and memorability
  + Trivia uniqueness and educational value
* These characteristics can be explicitly specified in prompts to ensure proper game balance.

**Content Freshness and Uniqueness**

* AI generation produces novel connections and lesser-known facts that may not be prominently featured on standard travel sites.
* The unpredictable nature of AI-generated content creates surprise elements that keep gameplay engaging even for well-traveled users.
* Web scraping typically yields only the most common information found on popular sites, potentially creating predictable gameplay.

**Legal and Ethical Considerations**

* AI-generated content avoids the copyright and terms-of-service concerns associated with scraping and reproducing content from travel websites.
* Using generated content eliminates attribution requirements and potential licensing complications.
* While using AI requires careful prompt engineering to ensure accuracy, it presents fewer legal risks than direct content reproduction.

**Scalability and Maintenance**

* The AI-generation process can be easily repeated to expand the dataset with minimal marginal effort.
* When adding new destinations or updating existing ones, the same AI approach ensures consistency with established content.
* Web scraping requires ongoing maintenance as source websites change their structure or content policies.

**Content Resilience**

* AI-generated historical, cultural, and geographical information tends to remain valid over time.
* Travel website content frequently changes to reflect current events, prices, or seasonal attractions, making scraped data potentially ephemeral.

**3.1.3 Implementation Strategy**

Our AI-based dataset generation implementation includes these key elements:

1. **Diversity Enforcement**
   * Explicit instructions to include destinations spanning all continents
   * Balanced representation of different types of destinations (historical, natural, cultural)
   * Inclusion of both globally recognized and lesser-known locations
2. **Quality Assurance Process**
   * Manual review of a representative sample of destinations
   * Verification of clue appropriateness (difficulty, uniqueness, solvability)
   * Fact-checking of selected destinations to ensure accuracy
3. **Dataset Evolution Strategy**
   * Initial deployment with core dataset of 100+ destinations
   * Planned periodic expansion to maintain user interest
   * Potential for seasonal or thematic content batches

This approach enables us to create a rich, engaging database of travel destinations that fulfills all project requirements while maximizing gameplay quality and minimizing legal and maintenance concerns.

**1. Dataset Generation**

* **Implementation**: Created a robust data generation system with multiple approaches:
  + **Template-based generation**: Developed a local generation script (generateDestinationsLocal.js) that creates destination data without requiring API calls
  + **AI-assisted generation**: Implemented OpenAI API integration (generateDestinations.js) as an alternative approach
  + **Data fixing**: Created a script (fixDestinationData.js) to validate and correct data issues
* **Dataset Structure**: Successfully generated 100+ destinations with the following attributes:
  + City name and country
  + Continent classification
  + 2-3 cryptic clues per destination
  + 2-3 fun facts per destination
  + 2-3 trivia items per destination
  + Difficulty level assignment (easy, medium, hard)
* **Geographic Distribution**: Ensured balanced representation across continents:
  + Africa (15%)
  + Asia (20%)
  + Europe (25%)
  + North America (15%)
  + South America (10%)
  + Oceania (15%)

**2. Database Integration**

* **MongoDB Setup**: Successfully configured MongoDB connection:
  + Created schema definitions with proper validation
  + Implemented connection handling with error management
  + Set up both local and cloud connection options
* **Data Import**: Created a complete data pipeline:
  + Data validation via validateDestinations.js
  + Data correction via fixDestinationData.js
  + Database import via importToMongoDB.js
* **Database Schema**: Implemented Mongoose schema with:
  + Required fields validation
  + Array field minimum length validation
  + Compound indexing for city/country uniqueness
  + Enum validation for difficulty levels

**3. Project Structure**

* Established a well-organized project structure:
* globetrotter/
* │
* ├── backend/ # Backend server code
* │ ├── config/ # Configuration files
* │ │ ├── db.js # MongoDB connection
* │ │ └── env.js # Environment variables
* │ │
* │ ├── models/ # MongoDB schemas
* │ │ └── Destination.js # Destination model
* │ │
* │ ├── scripts/ # Utility scripts
* │ │ ├── generateDestinationsLocal.js # Local data generation
* │ │ ├── fixDestinationData.js # Data correction
* │ │ ├── validateDestinations.js # Data validation
* │ │ └── importToMongoDB.js # Database import
* │ │
* │ └── server.js # Server entry point (in progress)
* │
* ├── data/ # Data files
* │ ├── starter\_destinations.json # Initial example data
* │ └── destinations.json # Generated destination dataset
* │
* ├── .env # Environment variables
* └── package.json # Project dependencies
* **NPM Scripts**: Configured scripts for key operations:
  + generate-data-local: Generate destination data without API
  + fix-data: Fix data issues and validation errors
  + import-data: Import destinations to MongoDB

**Technical Decisions**

**1. Data Generation Approach**

**Decision**: Use template-based generation instead of relying on OpenAI API.

**Rationale**:

* **Cost-effectiveness**: Eliminates API usage costs
* **Speed**: Generates data instantly without API rate limits
* **Reliability**: No dependency on external services
* **Consistency**: More predictable and consistent output

**Implementation**:

* Created templates for clues, fun facts, and trivia
* Populated with city-specific data (landmarks, keywords)
* Used randomization for variety while maintaining quality
* Ensured parameter-based difficulty assignment

**2. MongoDB Integration**

**Decision**: Design for flexible deployment with both local and cloud options.

**Rationale**:

* **Development simplicity**: Local MongoDB for easy development
* **Production readiness**: Cloud options (Atlas, Railway) for deployment
* **Forward compatibility**: Modern connection approach (removed deprecated options)

**Implementation**:

* Environment-variable-based connection string
* Connection error handling and reporting
* Batch-based import to handle large datasets

**3. Data Validation**

**Decision**: Implement comprehensive validation before database import.

**Rationale**:

* **Data integrity**: Ensures all entries follow the required structure
* **Game quality**: Validates clue quality (no direct city mentions)
* **Performance**: Prevents database errors during import

**Implementation**:

* Validation for required fields
* Check for duplicates
* Array length validation
* Quality checks for clues