Instruction Set for Claude Agent: UniVerse Project Development

Project Overview:
We are building "UniVerse," an AI-powered Study Abroad Guidance Platform for Bangladeshi students. The tech stack is MERN (MongoDB, Express, React, Node.js) with specific AI integrations. The platform must be cost-effective, using free tiers of services.

Core Technologies & APIs:

- Frontend: React.js with Tailwind CSS
- Backend: Node.js with Express
- Database: MongoDB Atlas
- Authentication: Firebase Auth (Free Tier)
- AI & Data:
    - Chatbot, SOP/CV Assistant, Data Analysis: Google Gemini API
    - Professor Finder: Google Scholar Scraper (or a fallback like Semantic Scholar API if scraping proves difficult)
- Hosting: Vercel (Frontend) and a free-tier backend service (e.g., Render, Cyclic)

Phase 1: Project Foundation & Setup

Task 1.1: Initialize the Project Repository

- Instruction: Create a new directory named universe-app. Inside it, create two sub-directories: client (for the React frontend) and server (for the Node.js/Express backend). Initialize a Git repository in the root folder. Create a README.md file with the project description and setup instructions.
- Acceptance Criteria: A project structure like this exists:
- text

universe-app/

├── frontend/

├── backend/

- └── README.md

Task 1.2: Backend Server Setup

- Instruction: Navigate to the server directory. Run npm init -y to create a package.json file. Install the following core dependencies: express, mongoose, cors, dotenv. Install these development dependencies: nodemon.

- Acceptance Criteria: The server/package.json file lists the above dependencies. A basic server.js file can be started with nodemon server.js and runs on a specified port (e.g., 5000).

Task 1.3: Frontend Client Setup

- Instruction: Navigate to the client directory. Use create-react-app or Vite to bootstrap a new React application. Install axios for API calls and react-router-dom for routing. Install and configure Tailwind CSS for styling.
- Acceptance Criteria: The React dev server starts successfully. A sample component styled with Tailwind CSS renders correctly in the browser.

Task 1.4: MongoDB Atlas Database Connection

- Instruction: Create a new cluster on MongoDB Atlas. Create a database user and get the connection string. In the server directory, create a .env file and add the connection string as MONGO_URI. Write the code in server.js to connect to the MongoDB cluster using Mongoose.
- Acceptance Criteria: The server console logs "MongoDB Connected Successfully!" on startup.

Phase 2: Database Schema & User Authentication

Task 2.1: Define MongoDB Schemas with Mongoose

- Instruction: In the server directory, create a models folder. Create the following Mongoose Schemas:
    - User.js: Fields - firebaseUid (String, unique), email (String), name (String), profile (Object containing fields like studyLevel, desiredCountry, researchInterests, cgpa, etc.).
    - SavedUniversity.js: Fields - userId (ObjectId, ref: 'User'), universityName, country, program, deadline (optional).
    - ApplicationTracker.js: Fields - userId (ObjectId, ref: 'User'), universityName, program, deadline, status (e.g., "Not Started", "In Progress", "Applied").
- Acceptance Criteria: The schemas are defined and can be imported into other files. Sample data can be created and saved to the database.

Task 2.2: Implement Firebase Authentication on Frontend

- Instruction: Create a new project in the Firebase console. Enable Email/Password authentication. In the client directory, install the Firebase SDK (firebase). Create a firebase.js configuration file with your project's config object. Create React components for Login.js and SignUp.js that use createUserWithEmailAndPassword and signInWithEmailAndPassword.

- Acceptance Criteria: A user can successfully register a new account and log in. The login state persists across page refreshes.

Task 2.3: Connect Firebase to Backend & Protect Routes

- Instruction: On the backend, install firebase-admin to validate Firebase ID tokens. Create a middleware function that checks the Authorization header for a Bearer token, verifies it with Firebase Admin, and attaches the user's UID to the req.user object. Apply this middleware to all protected API routes.
- Acceptance Criteria: Protected backend routes (e.g., /api/profile) return a 401 error if no valid token is provided. When a valid token is provided, the route handler can access req.user.uid.

Phase 3: Core Feature Implementation

Task 3.1: Integrate Google Gemini API

- Instruction: Get an API key for Google Gemini. In the server directory, create a controllers folder and a file aiController.js. Write a function that uses the Gemini SDK to generate text. Create Express routes (e.g., POST /api/ai/chat, POST /api/ai/generate-sop) that take user prompts and stream the response from Gemini back to the frontend.
- Acceptance Criteria: The frontend can send a message to /api/ai/chat and display the AI's streaming response in a chat-like interface.

Task 3.2: Build the AI Chatbot Interface

- Instruction: In the client directory, create a components folder and a Chatbot.js component. This component should have an input field and a message display area. Use axios or a fetch wrapper to call the /api/ai/chat endpoint and display the conversation history.
- Acceptance Criteria: The user can type questions like "What are the requirements for a Master's in CS in Germany?" and get a coherent answer from the AI.

Task 3.3: Build the SOP & CV Assistant

- Instruction: Create a new component SOPHelper.js. It should have a form for the user to input key details (Target University, Program, Their Background, Strengths, etc.). On submit, this data is sent to the /api/ai/generate-sop endpoint. The returned draft is displayed in an editable text area. Implement a "Save Draft" feature that saves the SOP to the user's profile in the database.
- Acceptance Criteria: A user can fill out the form, generate a structured SOP draft, and save it to their account.

Task 3.4: Implement University Recommendation Logic

- Instruction: This will be a rule-based + AI analysis feature.
    1. Data Collection: Find a free dataset of universities (e.g., from Kaggle) or use a free API. The data should include name, country, QS Ranking, popularPrograms, tuitionFee, etc. Import this data into a new MongoDB collection called Universities.
    2. Rule-Based Filter: Create an API route GET /api/universities/recommend. It should accept query parameters like country, desiredProgram, maxTuition, minRanking. Use Mongoose to filter the Universities collection based on these parameters.
    3. AI Analysis: For a more personalized touch, take the user's profile (from User model) and the filtered list of universities, and send it to Gemini via a carefully crafted prompt. The prompt should be: "Based on the following student profile [insert profile] and this list of universities [insert list], provide a short, personalized analysis of the top 3 most suitable options and why."
- Acceptance Criteria: The API returns a list of universities based on filters. The response includes an AI-generated analysis of the top recommendations.

Phase 4: Advanced Features & Final Polish

Task 4.1: Professor Finder via Google Scholar

- Instruction: Note: Scraping Google Scholar is complex and against its ToS. We will implement a simulated version.
    - Create a Professor.js schema with fields: name, university, department, researchInterests, scholarProfileLink.
    - Manually populate this collection with data for 20-30 popular professors in fields relevant to Bangladeshi students (CS, Engineering, etc.).
    - Create an API route GET /api/professors that allows filtering by researchInterest and university.
    - On the frontend, create a ProfessorFinder.js component with a search bar. When a user searches for "Machine Learning," it calls the API and displays the matching professors.
- Acceptance Criteria: A user can search for professors by research interest and see a list of results with their details.

Task 4.2: Application Deadline Tracker

- Instruction: Create a component ApplicationTracker.js. It should allow a user to:
    1. Add Application: Form to add a new university application (uses data from the SavedUniversity schema).
    2. View Applications: Display a list/table of all their saved applications.

3. Update Status: Change the status of an application (e.g., from "In Progress" to "Applied").
4. Receive Reminders: (Stretch Goal) Implement a simple check that highlights applications where the deadline is less than 30 days away.

- Acceptance Criteria: A user can CRUD (Create, Read, Update) their university applications.

Task 4.3: Final Integration, Styling & Deployment

- Instruction:
  1. Use React Router to link all the components (Chatbot, SOP Helper, University Recommender, Professor Finder, Tracker) into a single, cohesive application.
  2. Apply a consistent and professional styling across all components using Tailwind CSS. Ensure the UI is fully responsive.
  3. Deploy the backend to a free service like Render or Cyclic. Update the frontend's API calls to point to the live backend URL.
  4. Deploy the frontend to Vercel.
- Acceptance Criteria: The entire application is live on a public URL. All features are accessible and functional for a logged-in user.

---

2. Advanced AI Features

- Document Analyzer: AI that reviews and gives feedback on existing SOPs/CVs
- Interview Preparation: AI-powered mock interviews with common study abroad questions
- Profile Strength Analyzer: AI evaluates user profile and suggests improvements
- Cost of Living Calculator: AI estimates living expenses for different cities/countries
- Scholarship Matching: More sophisticated scholarship recommendations
- Country Fit Analyzer: Quiz-based country recommendation system

- Dark Mode: Better user experience
- Success Stories: Feature successful Bangladeshi students abroad