**20-Day Project Timeline**

**Phase 1: Project Setup & Initial Backend Foundation (Days 1–5)**

**Day 1**:

* Set up the project folder structure (use the script I provided).
* Initialize Git for version control.
* Create requirements.txt and install dependencies.
* Configure FastAPI entry point (main.py) and the run.py script to test the FastAPI server.

**Day 2**:

* Develop the **Email Management Routes** (email.py):
  + Categorize emails.
  + Draft emails (dummy logic for now).
* Set up placeholder mail\_service.py.

**Day 3**:

* Develop the **Calendar Management Routes** (calendar.py):
  + Fetch calendar events.
  + Schedule a meeting (dummy API response).
* Implement calendar\_service.py for logic handling.

**Day 4**:

* Develop **Task Management Routes** (tasks.py):
  + Add tasks.
  + List tasks.
* Create task\_service.py for integrating with Todoist/Trello APIs.

**Day 5**:

* Develop **Media Search Routes** (media.py):
  + Search for images (Google Custom Search API).
  + Search for videos (YouTube Data API).
* Integrate APIs into media\_service.py.
* Test all routes so far using Postman/Swagger.

**Phase 2: AI Assistant Integration & Core Logic (Days 6–10)**

**Day 6**:

* Develop **AI Assistant Routes** (assistant.py) for:
  + Query handling using OpenAI API.
* Implement nlp\_service.py to connect to OpenAI's API.
* Test responses using dummy prompts.

**Day 7**:

* Develop **Speech-to-Text and Text-to-Speech** functionalities:
  + Use Google Speech-to-Text API and Text-to-Speech API.
* Add these services in speech\_service.py.

**Day 8**:

* Add support for **Alexa Integration** (alexa.py) using Alexa Skills API.
* Implement alexa\_service.py for query handling.

**Day 9**:

* Develop **User Preferences Routes** (user.py):
  + Update preferences.
  + Retrieve preferences.
* Implement user\_schema.py and user\_model.py for Pydantic models.

**Day 10**:

* Add **Authentication Routes** (auth.py) with OAuth for APIs (MailAPI, CalendarAPI).
* Create config/oauth.py for token management.
* **Midway Testing**: Test all implemented routes thoroughly.

**Phase 3: Backend Refinement & Testing (Days 11–15)**

**Day 11**:

* Integrate **database connection** (db.py) using SQLite.
* Create data models for tasks and preferences in models.py.

**Day 12**:

* Update all services to use the database (store tasks, preferences, meeting logs).
* Add **CRUD operations** to respective routes.

**Day 13**:

* Implement **Error Handling** and status codes across all routes.
* Use FastAPI exceptions to handle errors gracefully.

**Day 14**:

* Develop **Weekly/Monthly Reporting** for:
  + Tasks completed.
  + Meeting summaries.
* Add a new route for generating reports.

**Day 15**:

* Create **Utility Functions** in utils.py for:
  + Common helpers (e.g., time formatting, validation, etc.).
  + API key management.
* **Test Everything** so far.

**Phase 4: Final Features, Optimization & Deployment Prep (Days 16–20)**

**Day 16**:

* Implement **Smart Scheduling**:
  + Use a dummy ML model or rules-based system to suggest meeting times.
* Test calendar API integration.

**Day 17**:

* Add a **Feedback Mechanism** for task and assistant responses.
* Implement preferences-driven responses (tone, urgency).

**Day 18**:

* Work on **Data Security**:
  + Mask sensitive data in logs.
  + Validate inputs across routes.

**Day 19**:

* Write **Unit Tests** for all routes:
  + Use pytest for test\_email.py, test\_calendar.py, etc.
* Final API documentation cleanup in Swagger.

**Day 20**:

* Conduct a full project review and testing.
* **Deployment Prep**:
  + Add production settings.
  + Finalize database setup.
  + Test API deployment using Uvicorn or containerize with Docker.
* **Buffer Day** for fixes or enhancements.