Below is an incremental, **single-developer** plan for building **Donna**—the AI-powered personal assistant. The plan balances core functionality (MVP) with the incremental addition of more advanced features (AI-driven tasks, wellness reminders, etc.). Each phase includes recommended tasks, tools, and best practices to keep development focused, manageable, and efficient.

Phase 0: Preparation and Environment Setup

1. **Define Tech Stack**
    - **Backend**: Python, using either *FastAPI* (preferred for async and built-in Pydantic models) or *Flask* (well-known ecosystem).
    - **Frontend (Web)**: React.
    - **Mobile**: React Native (can be postponed until the core backend APIs are stable).
    - **Database**: PostgreSQL for persistent data.
    - **Caching**: Redis (for travel times, API responses).
    - **AI Modules**: Rasa (for NLP) or an OpenAI GPT-based approach for more advanced language understanding.
2. **Set Up Dev Environment**
    - **Local Development**: Docker-compose or a local virtual environment.
    - **Version Control**: GitHub/GitLab.
    - **Continuous Integration (CI)**: Simple GitHub Actions/CI pipeline (test on push/pull requests).
3. **Create High-Level Architecture**
    - Outline how the backend, database, frontend, and NLP modules will communicate.
    - Decide if you'll use a monolithic approach initially (single service plus separate NLP service) or a microservices approach (might be overkill for a solo dev at first).

Phase 1: Minimum Viable Product (MVP)

Goal: Build the core workflow: user authentication, task management, and basic calendar integration.

1. **User Authentication & Profiles**

    - **Features**: Sign-up, login, password reset.
    - **Implementation**:
        - Use *FastAPI*'s OAuth2 or *Flask-JWT* for token-based authentication.
        - Create a `users` table in PostgreSQL (or NoSQL if preferred, but PostgreSQL is usually better for relational data).
2. **Task Management (Local Only)**
    - **Features**:
        - Create, read, update, delete (CRUD) tasks.
        - Basic prioritization fields: `due_date`, `importance`, `status`.
    - **Implementation**:
        - RESTful endpoints: `GET /tasks`, `POST /tasks`, `PUT /tasks/:id`, `DELETE /tasks/:id`.
        - A simple model `Task` in the DB with relevant columns.
3. **Basic Calendar View**
    - **Frontend**:
        - A React app with a simple calendar component (e.g., `react-big-calendar` or any suitable library).
        - Display tasks by due date.
    - **Backend**:
        - An endpoint returning tasks (and mock calendar events) to populate the calendar.
4. **Basic UI/UX**
    - Keep it simple and functional. Minimal styling.
    - Focus on clarity: a home page with upcoming tasks, a calendar page, and a tasks page.
5. **Deployment (Development & Testing)**
    - Deploy to a testing environment (e.g., Heroku, AWS free tier, or a small Docker server on a cloud VM).
    - Ensure you have a working end-to-end flow (login → create task → display on calendar).

Estimated Time: 3–5 weeks (part-time solo dev).

Phase 2: External Integrations

Goal: Connect to critical services like Canvas, Google Calendar, and possibly Gradescope to fetch real user data.

1. **Canvas API Integration**
    - **Features**:
        - Fetch assignments, deadlines, and course events.
        - Store them in the local DB, marking them as `source=canvas`.
    - **Implementation**:
        - Use a `CanvasService` class/module to handle API calls.
        - Store relevant data in `assignments` or unify it in your `tasks` table.
    - **Challenges**:
        - Handle OAuth or API tokens.
        - Respect Canvas rate limits.
2. **Google/Outlook Calendar Sync**
    - **Features**:
        - Push user-created tasks to Google Calendar.
        - Pull events from Google Calendar to show them in the app.
    - **Implementation**:
        - Use Google Calendar's OAuth flow for user permission.
        - Create an `EventsService` for external event sync.
3. **Gradescope (Optional for Early Stage)**
    - **Features**:
        - Retrieve grades to display in a "Grade Tracking" dashboard.
    - **Implementation**:
        - Similar approach: a `GradesService` for collecting user data.
        - Store grade data in a `grades` table.
4. **Notification System (Push / Email)**
    - **Features**:
        - Use Firebase Cloud Messaging (FCM) for push notifications (web or mobile).
        - Optionally Twilio SMS for urgent alerts.
    - **Implementation**:
        - Simple notification microservice or module in the same backend.
        - Send notifications for upcoming deadlines (24hr, 2hr, etc.).
5. **Frontend Enhancements**
    - **Sync Indicators**: Show which tasks come from Canvas or Google Calendar.
    - **OAuth Settings Page**: For connecting/disconnecting user accounts.

Estimated Time: 4–6 weeks (depending on complexity of integrations).

Phase 3: Smart Scheduling & Task Prioritization

Goal: Implement AI-driven scheduling logic to suggest time blocks, prioritize tasks, and handle conflicts.

1. **Task Prioritization Engine**
    - **Algorithm**:
        - Weighted scoring based on `urgency`, `difficulty`, `impact`, and deadlines.
        - Start simple (heuristic-based) before implementing ML.
    - **Implementation**:
        - When tasks are fetched or created, assign a priority score.
        - Sort tasks for display on the dashboard.
2. **Smart Scheduler**
    - **Features**:
        - Identify free blocks in the user's calendar.
        - Suggest optimal time for tasks (time-blocking).
    - **Implementation**:
        - Build a simple scheduling algorithm that looks at tasks vs. free time.
        - Provide a button: "Suggest schedule" → populates calendar with recommended blocks.
    - **Conflict Resolution**:
        - If a block overlaps with another event, automatically shift or reduce the block.
3. **Rescheduling Logic**
    - **Features**:
        - If a user misses a task block or marks it incomplete, re-insert into the next available slot.
    - **Implementation**:
        - Keep track of incomplete tasks, auto-reschedule them based on user preferences (e.g., "reschedule within 24 hours").
4. **UI/UX**
    - **Calendar Drag-and-Drop**:
        - User can manually adjust scheduled blocks.
        - Real-time updates to priority or next block suggestions if a block is moved.

Estimated Time: 3–5 weeks.

Phase 4: NLP and Voice Assistant Features

Goal: Allow users to interact with Donna via text and/or voice, extracting intents (create tasks, fetch grades, etc.).

1. **NLP Integration**
    - **Choice**: Rasa for on-prem or GPT-based for more advanced/flexible language understanding.
    - **Features**:
        - Intent classification: “Create task,” “Check deadlines,” “Show calendar,” “What’s due tomorrow?”
        - Entity extraction: e.g., task name, date/time, course name.
    - **Implementation**:
        - Create a new service/module in the backend that processes user queries.
        - Return structured data to your scheduling/task modules.
2. **Voice Processing (Optional)**
    - **Implementation**:
        - Use browser-based speech-to-text (Web Speech API) or a service like Google Cloud Speech.
        - Send recognized text to the NLP engine.
3. **Conversation Context**
    - **Features**:
        - Ability to handle follow-up queries (“Add to that assignment I mentioned.”).
    - **Implementation**:
        - Maintain a lightweight session or conversation context in Redis or local memory.
4. **User Feedback Loop**
    - **Features**:
        - “Did I interpret that correctly?” If user says no, refine the model or logic.

Estimated Time: 4–6 weeks (training models, refining, testing).

Phase 5: Advanced and Optional Features

Goal: Add polish, advanced analytics (grade forecasting, wellness reminders), and collaboration features.

1. **Grade Forecasting**
    - **Features**:
        - For each course, let the user simulate potential performance by adjusting future assignment scores.
    - **Implementation**:
        - Simple formula-based or regression-based approach using existing grade data.
        - Provide a UI slider or input fields for hypothetical grades.
2. **Health and Wellness Reminders**
    - **Features**:
        - Personalized suggestions: “Take a 5-minute break,” “Drink water,” “Stretch.”
    - **Implementation**:
        - Basic rules: If the user has been working >2 hours, prompt a break.
        - Optionally use a small model that tracks stress or workload patterns.
3. **Collaboration Tools**
    - **Shared Calendar/Task Lists**:
        - Let users invite classmates to shared tasks or group events.
    - **Group Projects**:
        - Simple Kanban board or shared deadlines with chat integration.
4. **Polish and Performance Optimizations**
    - **Caching**:
        - Use Redis effectively for API calls to Canvas, Maps, etc.
    - **UI/UX Enhancements**:
        - Refine design, add theming (dark mode), better navigation.
    - **Mobile App**:
        - Full React Native build with offline caching, push notifications.

Estimated Time: Ongoing, depends on complexity and user feedback.

Phase 6: Testing, QA, and Maintenance

1. **Automated Testing**
    - **Unit Tests**: For scheduling logic, NLP intent handling, data models.
    - **Integration Tests**: Mock external APIs (Canvas, Google Calendar).
    - **End-to-End (E2E)**: Cypress or Playwright for frontend flows.

2. **Load and Stress Testing**
    - **Objective**: Ensure your system can handle concurrency (target 500+ users).
    - **Tools**: Locust or JMeter.
3. **Security Audits**
    - **Encryption**: Verify data is encrypted in transit (HTTPS) and at rest (database).
    - **OAuth**: Ensure tokens and refresh tokens are handled securely.
    - **FERPA Compliance**: Ensure no unauthorized data exposure.
4. **Continuous Improvement**
    - **Feedback Channels**: Let beta users share feedback.
    - **Iterative Updates**: Roll out minor versions with bug fixes, small feature tweaks.

Practical Tips for a Solo Developer

1. **Iterative, User-Focused Approach**:
    - Release early versions to a small group (or even just yourself) to get quick feedback.
    - Prioritize *must-have* features (task management, calendar integration) over *nice-to-have* features (voice assistant, advanced AI) in the early stages.
2. **Stay Organized**:
    - Use a project management tool (Trello, Jira, or GitHub Projects) to track tasks and sprints.
    - Break down big features (like “NLP integration”) into smaller subtasks.
3. **Reuse Existing Solutions**:
    - For NLP, consider an existing model (OpenAI or Rasa) rather than building from scratch.
    - For scheduling, you might start with a simple heuristic approach before diving into more complex AI.
4. **Keep an Eye on Scope**:
    - Because you’re solo, be mindful of feature creep.
    - Focus on building a stable core, then layer advanced capabilities later.
5. **Security Best Practices**:
    - Protect API keys and secrets in environment variables.
    - Implement role-based access control if you expand to collaboration features.

6. **Document as You Go**:
    - Maintain an up-to-date README or wiki for your codebase.
    - Write short but clear docs for any custom modules or complicated logic (especially around scheduling).

High-Level Timeline (Approximation)

- **Month 1–2**:
    - Phase 1 (MVP) → Basic Auth, Tasks, Calendar, initial deployment.
- **Month 3–4**:
    - Phase 2 (Integrations) → Canvas, Google Calendar, basic notifications.
- **Month 5–6**:
    - Phase 3 (Smart Scheduling) → Priority engine, scheduling logic, UI enhancements.
- **Month 7–8**:
    - Phase 4 (NLP) → Basic text-based NLP for tasks and queries, possibly voice features.
- **Month 9+**:
    - Phase 5 (Advanced) → Grade forecasting, wellness reminders, collaboration.
    - Ongoing testing, QA, performance optimization.

(Adjust timelines based on your availability and whether you're working full-time or part-time.)

Conclusion

Building **Donna** as a solo developer involves a carefully phased approach:

1. Start with the essential features (task management, calendar display).
2. Gradually integrate external APIs and add AI-driven capabilities.
3. Prioritize security, usability, and iterative feedback to ensure you deliver a stable, user-friendly assistant.
4. Layer in advanced functionalities (NLP, wellness reminders, grade forecasting) once the core experience is solid.

Following this roadmap helps ensure that each component is well-tested and that you can deliver tangible value at every step—ultimately culminating in a robust, student-focused AI personal assistant.