Dynamic Organizer and Normal Needs Assistant: AI Personal Assistant

Functional & Technical Requirements

1. Introduction

Donna is an AI-powered personal assistant designed for students. Inspired by Jarvis and Friday from the Marvel universe, Donna helps users manage coursework, schedules, tasks, and personal wellness. The assistant integrates with educational and productivity platforms to streamline academic and extracurricular activities.

This document defines the key **functional** and **technical** requirements necessary for Donna's successful development, as well as an **incremental development plan** aligned with a single-developer workflow. **Palantir Foundry** will be leveraged to facilitate centralized data management, advanced analytics, and compliance.

---

2. Functional Requirements

2.1 Core Features

1. **Task Management**
    - **Fetch assignments and tasks** from LMS platforms (e.g., Canvas, Gradescope).
    - **Local Task CRUD**: Users can create, update, or delete tasks independently of LMS.
    - **Automatic Prioritization**: Rank tasks by urgency, due date, and importance.
2. **Smart Scheduling**
    - **Time-Blocking**: Suggest optimal time blocks for tasks or study sessions.
    - **Conflict Resolution**: Detect and resolve overlapping events.
    - **Auto-Rescheduling**: If a user misses or postpones a task, Donna suggests the next available time.
3. **Native Calendar View**
    - **Calendar Modes**: Day, week, and month views.

    - **Drag-and-Drop Rescheduling**: Users can manually shift tasks, auto-updating in the system.
    - **Color Coding**: Distinguish tasks vs. events (e.g., personal events, classes, deadlines).
4. **Proactive Notifications**
    - **Upcoming Deadlines**: Alert users of impending due dates or conflicts.
    - **Actionable Suggestions**: Prompt users with quick-resolve options (e.g., “Start working on Assignment X now”).
    - **Missed Tasks**: Notify users when tasks are overdue and suggest rescheduling.
5. **Grade Tracking and Forecasting**
    - **Fetch Grades**: Integrate with Canvas, Gradescope, or SIS to retrieve current grades.
    - **Grade Simulations**: Users can hypothetically change future grades to see potential final outcomes.
    - **Trends and Insights**: Provide basic analytics (e.g., “If you score at least 80% on the final, you can raise your course grade to B+”).
6. **Travel Time Integration**
    - **Maps API**: Fetch travel times via Google Maps or OpenStreetMap.
    - **Time-to-Leave Alerts**: Notify users when to depart based on real-time traffic.
    - **Location-Aware Scheduling**: Factor commute times into scheduling tasks or events.
7. **Collaboration Features (Group Projects)**
    - **Shared Calendars**: Allow multiple users to view and edit group tasks.
    - **Permissions**: Define roles (e.g., admin, member) for editing or assigning tasks.
    - **Project Dashboards**: Summarize group progress, deadlines, discussion threads.
8. **NLP and Voice Integration**
    - **Text/Voice Commands**: “What’s due tomorrow?”, “Schedule a meeting with my professor at 3 PM.”
    - **Intent Extraction**: Identify user requests (create task, get grades, show schedule).
    - **Context Handling**: Maintain conversation state for follow-up requests.
9. **Health and Wellness Reminders**
    - **Periodic Notifications**: Prompt users to take breaks, hydrate, or stretch during long study sessions.
    - **Personalization**: Adjust frequency and type of wellness reminders based on user feedback or schedules.
10. **Recurring Tasks and Events**

- **Repetitive Schedules**: Weekly, monthly, or custom recurrences for classes, bills, or tasks.
- **Automatic Reminders**: Push notifications for recurring events as well.

2.2 Integrations

1. **Canvas API**
    - **Endpoints**: `/courses/:course_id/assignments`, `/calendar_events`
    - **Data Fetched**: Assignments, course schedule, announcements.
2. **Google/Outlook Calendar**
    - **Sync**: Push tasks from Donna into external calendars; pull personal events into Donna.
    - **OAuth2** for user permission and access tokens.
3. **Gradescope API**
    - **Data Fetched**: Grades, feedback, assignment metadata.
    - **Notifications**: Alerts on new or updated feedback.
4. **Piazza/Ed Discussion API** (Optional)
    - **Discussion Threads**: Collate relevant posts within the app.
    - **Notifications**: Alert users on new course-related messages.
5. **Maps API** (Google or OpenStreetMap)
    - **Distance & Time**: Show estimated commute time for events with location data.
    - **Real-Time Traffic**: Adjust alerts if there's heavy traffic or transit delays.
6. **Notification System**
    - **Firebase**: For push notifications (web, Android, iOS).
    - **Twilio**: SMS alerts for critical tasks (optional configuration).
7. **Palantir Foundry**
    - **Data Integration & Governance**: Aggregate user, grades, and scheduling data securely in Foundry.
    - **Analytics & Collaboration**: Use Foundry's platform to create pipelines, dashboards, or advanced analytics around user engagement, academic performance, and system usage.

---

3. Technical Requirements

3.1 Architecture & Frameworks

- **Backend**:
    - **Python** with **FastAPI** (preferred for async and built-in validation) or **Flask**.
    - **RESTful API** exposing endpoints for tasks, scheduling, notifications, and integrations.
    - **Containerized**: Docker for easy deployment.
- **Frontend**:
    - **Web App** with **React** (for an interactive calendar, tasks, dashboards).
    - **Mobile App** with **React Native** (for on-the-go task management and notifications).
- **AI & NLP**:
    - **Rasa** (on-prem) or **OpenAI GPT** for advanced language understanding.
    - Scheduling and prioritization algorithms can be heuristic-based initially, then improved with ML or advanced optimization libraries (e.g., **PyTorch**, **Scikit-learn**).

3.2 Data Management

1. **Relational Database**
    - **PostgreSQL** for storing tasks, events, user profiles, and grade data.
    - **Schema** with entities like `users`, `tasks`, `events`, `grades`.
2. **Caching Layer**
    - **Redis** for storing frequently accessed data (e.g., travel time, external API responses).
3. **Palantir Foundry Integration**
    - **Data Pipelines**: Push relevant user and assignment data into Foundry for consolidation.
    - **Advanced Analytics**: Use Foundry's built-in tools for exploring usage metrics (e.g., overall student engagement, performance correlations).
    - **Governance & Compliance**: Centralize sensitive data with Foundry's data governance framework.

3.3 Backend Services

- **Task Prioritization Engine**
    - **Weighted Algorithm**: Evaluate tasks by urgency, importance, difficulty.

    - **ML Option**: Possible future use of a regression/classification model to predict recommended priority.
- **Smart Scheduler**
    - **Time-Block Logic**: Identify free slots, propose task blocks.
    - **Conflict Resolution**: Re-adjust suggested blocks upon conflicts or missed tasks.
- **Notifications Service**
    - **Push via Firebase** or **SMS via Twilio**.
    - Triggered by upcoming deadlines, changes in schedule, or grade updates.

3.4 Security & Compliance

- **OAuth2.0** for external integrations (Canvas, Google).
- **Role-Based Access Control (RBAC)** for group features.
- **Encryption**: TLS for data in transit, encryption at rest for sensitive data.
- **FERPA Compliance**: Ensure no unauthorized sharing of grades.
- **GDPR/CCPA** (if applicable): Allow users to export or delete personal data.

3.5 Non-Functional Requirements

1. **Performance**
    - **Response Times**: Task creation and scheduling queries <2 seconds.
    - **Scalability**: Should support 500+ concurrent users.
2. **Reliability**
    - **Uptime**: 99.9% for core calendar/notification features.
    - **Failover**: Graceful degradation if external APIs are down (store partial data in cache).
3. **Maintainability**
    - **CI/CD**: Automated builds (GitHub Actions), container-based deployment.
    - **Modular Code**: Separate modules for scheduling, NLP, integrations.
4. **Usability**
    - **Intuitive UI** for quick navigation.
    - **Accessibility (A11y)**: Adhere to WCAG guidelines (color contrast, keyboard navigation, screen reader labels).
5. **Scalability**
    - **Container Orchestration** (Kubernetes or Docker Swarm) if usage spikes.
    - **Load Balancing** for distributed traffic across multiple backend instances.

---

4. Phased Implementation Plan

Below is a condensed development roadmap tailored for a **solo developer**. The plan emphasizes building core functionality first, then progressively adding advanced features and Foundry integration.

Phase 0: Environment Setup

- Establish local dev environment with Docker.
- Configure PostgreSQL, Redis, and basic CI/CD pipeline.

Phase 1: MVP (Core)

- **User Auth & Profiles**: Basic signup, login, JWT tokens.
- **Task Management (Local)**: CRUD tasks, store in PostgreSQL.
- **Calendar View**: Simple React UI with day/week/month display.
- **Initial Deployment**: Deploy to a small cloud instance (e.g., Heroku/AWS).

Phase 2: Integrations

- **Canvas**: Sync assignments/grades with user tasks.
- **Google Calendar**: Bi-directional sync (events ↔ tasks).
- **Notification System**: Firebase or Twilio for upcoming deadlines.

Phase 3: Smart Scheduling & Prioritization

- **Task Prioritization Engine**: Weighted scoring for tasks.
- **Automated Scheduling**: Suggest time blocks based on free slots.
- **UI Enhancements**: Drag-and-drop, conflict resolution warnings.

Phase 4: NLP and Voice Commands

- **NLP Integration** (Rasa/OpenAI GPT) for text commands.
- **Intent Detection**: “Create a task,” “What’s due tomorrow?”
- **Voice Integration** (optional) using browser or external speech services.

Phase 5: Palantir Foundry Integration & Advanced Features

- **Data Pipeline to Foundry**:
    - Ingest user activity, tasks, and grade data into Foundry for centralized analytics.
    - Set up compliance and governance for sensitive data.
- **Advanced Analytics**:

    - Use Foundry to build dashboards for usage metrics, grade forecasting, and patterns.
- **Health & Wellness**:
    - Proactive break reminders, optional advanced analytics (e.g., “high-stress periods” detection).

Phase 6: Testing, QA, and Optimization

- **Unit & Integration Tests**: Coverage for scheduling, NLP, data sync.
- **Load/Stress Tests**: Validate performance under concurrency.
- **Security Audit**: Ensure encryption, access control, compliance with FERPA.
- **Refinement**: Implement user feedback, fix bugs, optimize architecture.

---

5. Constraints

1. **API Rate Limits**
    - Must handle Canvas, Google, Gradescope usage within allowed quotas.
    - Implement caching (Redis) and fallback strategies if calls fail.
2. **Data Privacy**
    - FERPA compliance is critical.
    - Foundry usage must follow educational privacy regulations.
3. **Single Developer Resource**
    - Scope must be prioritized to ensure consistent progress.
    - Feature creep must be managed through a clear backlog.
4. **Third-Party Reliability**
    - Potential downtime of external APIs (Canvas, Google) requires graceful error handling.

---

6. Risks & Mitigation

1. **Integration Failures**
    - **Risk**: External APIs may change or be unavailable.
    - **Mitigation**: Versioned API endpoints, robust error handling, caching.

2. **NLP Misinterpretations**
    - **Risk**: Incorrect intent extraction.
    - **Mitigation**: Model training with diverse data, user confirmation, fallback prompts.
3. **Data Breaches**
    - **Risk**: Exposure of student grades.
    - **Mitigation**: Encrypt data at rest, secure tokens, regular security audits.
4. **Performance Bottlenecks**
    - **Risk**: Slow scheduling or large concurrency.
    - **Mitigation**: Redis caching, indexing in PostgreSQL, potential horizontal scaling.
5. **User Adoption**
    - **Risk**: Complex UI or onboarding may deter students.
    - **Mitigation**: Streamlined UI/UX, minimal friction sign-up, iterative user testing.

---

7. Deliverables

1. **Functional & Technical Specifications** (this document).
2. **Prototypes**:
    - **Calendar Wireframes** (web, mobile).
    - **Task Management UI** mockups.
3. **API Integration Plan**:
    - Detailed steps for Canvas, Google, Gradescope, Foundry ingestion.
4. **Development Roadmap**:
    - Phased plan (MVP → Integrations → Smart Scheduling → NLP → Foundry).
5. **Testing Plan**:
    - Unit and integration tests, load testing, user acceptance testing.
6. **User Documentation**:
    - Quick-start guide and in-app tutorials.

---

8. Summary

Donna's **Functional Requirements** revolve around centralized task management, scheduling, proactive alerts, grade tracking, NLP-powered interactions, and wellness reminders. **Technical Requirements** specify a Python (FastAPI/Flask) backend, React/React Native frontend, PostgreSQL + Redis for storage and caching, optional advanced AI for scheduling/NLP, and **Palantir Foundry** for data integration and analytics. Adhering to the **phased development plan**

will ensure a manageable implementation path for a single developer, starting with a minimal viable product and progressively adding high-value features and integrations. By leveraging Foundry in later phases, Donna gains robust data governance, scalable analytics, and a solid compliance framework for sensitive student information.