frontend_task.md

A step-by-step guide for frontend development and integration with a Multi-tenant School Management System (SMS) backend built with Python, FastAPI, and PostgreSQL.

---

1. Frontend Project Initialization

- [ ] Initialize Next.js project with TypeScript.
- [ ] Configure ESLint and Prettier for code quality and consistency.
- [ ] Set up environment variables for backend API URLs (development, staging, production).
- [ ] Integrate Tailwind CSS or a similar styling framework for rapid UI development.
- [ ] Configure absolute imports for better code organization.

---

2. Data Layer Setup (React Query & API Client)

- [ ] Install and configure React Query (TanStack Query) for server state management.
- [ ] Create a centralized API client (e.g., using `axios` or `fetch` wrapper) to interact with FastAPI endpoints.
    - [ ] Implement base URL configuration for different environments.
    - [ ] Implement request interceptors for attaching authentication tokens.
    - [ ] Implement response interceptors for global error handling and logging.
- [ ] Generate TypeScript types from FastAPI OpenAPI schema (e.g., using `openapi-typescript-codegen` or `swagger-typescript-api`).
- [ ] Define React Query hooks (`useQuery`, `useMutation`) for all backend CRUD operations, leveraging generated TypeScript types.

---

3. Multi-tenant Architecture & Tenant Context

- [ ] Implement tenant identification strategy on the frontend (e.g., extracting tenant ID from subdomain, URL parameter, or user login).
- [ ] Ensure all API requests include the appropriate tenant identifier (e.g., in headers or query parameters).
- [ ] Implement a frontend context or global state to manage the active tenant information.

- [ ] Develop a mechanism for tenant switching/selection if applicable (e.g., a dropdown for admin users).

---

4. Authentication & Authorization Flow

- [ ] Implement user registration and login forms.
- [ ] Integrate with FastAPI JWT authentication endpoints (`/auth/login` , `/auth/register`).
- [ ] Securely store and manage JWT tokens (e.g., using HTTP-only cookies or local storage with appropriate security measures).
- [ ] Implement token refresh mechanism to maintain user sessions.
- [ ] Implement logout functionality, clearing tokens and user session.
- [ ] Implement role-based access control (RBAC) on the frontend:
    - [ ] Parse user roles and permissions from JWT payload or a dedicated user info endpoint.
    - [ ] Conditionally render UI elements and navigation based on user roles and permissions.
    - [ ] Implement route guards to protect routes based on authentication status and user roles.
- [ ] Implement password reset and forgot password flows.

---

5. Core Entities UI & CRUD Operations

For each core entity (User, Student, Teacher, Parent, Class, Section, Enrollment, Assignment, Grade, Subject, Notification, Announcement, Event, Exam, Feedback, Message, Resource, Schedule, Timetable):

- [ ] Develop dedicated pages/components for listing, viewing details, creating, editing, and deleting records.
- [ ] Implement data fetching using React Query `useQuery` for lists and individual records.
- [ ] Implement data mutations using React Query `useMutation` for create, update, and delete operations.
- [ ] Implement form validation using a library like `react-hook-form` and `zod` (leveraging TypeScript types).
- [ ] Implement pagination, filtering, and sorting for data tables.
- [ ] Display appropriate success/error messages for all CRUD operations.

---

6. Specific Module Integrations

- [ ] **Enrollment:** Implement UI for managing student enrollments, including rules and status.
- [ ] **Assignment & Grade:** Develop interfaces for teachers to create/manage assignments and input grades; for students to view assignments and grades.
- [ ] **Notification & Announcement:** Implement real-time (or near real-time) display of notifications and announcements.
- [ ] **Activity Log:** Create a view for administrators to monitor system activity logs.
- [ ] **Admin Panel:** Develop a comprehensive admin panel for managing tenants, users, roles, and system settings.
- [ ] **Schedule & Timetable:** Implement interactive UIs for viewing and managing class schedules and timetables.

---

7. UI/UX & Design System

- [ ] Implement a consistent design system across the application.
- [ ] Ensure responsive design for various screen sizes (desktop, tablet, mobile).
- [ ] Develop reusable UI components (buttons, inputs, modals, tables, etc.).
- [ ] Implement user-friendly navigation and clear information architecture.

---

8. Testing

- [ ] Write unit tests for React components and utility functions (e.g., using Jest and React Testing Library).
- [ ] Write integration tests for data fetching and state management logic.
- [ ] Implement end-to-end (E2E) tests for critical user flows (e.g., login, creating a student, enrolling in a class) using Cypress or Playwright.

---

9. Deployment & Monitoring

- [ ] Configure Next.js application for production build and deployment.
- [ ] Set up continuous integration/continuous deployment (CI/CD) pipeline for automated deployments.
- [ ] Integrate performance monitoring and error tracking tools (e.g., Sentry, Datadog).
- [ ] Implement logging for frontend errors and user interactions.

---

10. Future Considerations

- [ ] OAuth provider integration (Google, Microsoft) for single sign-on.
- [ ] Real-time communication (WebSockets) for chat or live updates.
- [ ] Internationalization (i18n) support.
- [ ] Offline capabilities with Service Workers.