Backend Development Plan - Quick Beaver Dive

1 Executive Summary

We will build a scalable **FastAPI** backend with **MongoDB Atlas** to replace the current `localStorage` implementation. The backend will serve as the single source of truth for Users, Projects, and generated Content (Lesson Plans, Worksheets, Parent Updates).

Constraints & Architecture:

- **Framework:** FastAPI (Python 3.13+, Async).
- **Database:** MongoDB Atlas (Motor + Pydantic v2).
- **No Docker:** Direct local execution with `uvicorn`.
- **Git:** Single branch `main`.
- **Testing:** Manual UI verification per task.
- **AI Integration:** Service layer design to support LLM calls (mocked/templated initially for zero-cost dev, swappable for Real API).

Sprint Overview:

- **S0:** Environment & Connectivity.
- **S1:** Authentication (JWT).
- **S2:** Project Management.
- **S3:** Lesson Plan & Worksheet Generation.
- **S4:** Parent Update Generation.

---

2 In-Scope & Success Criteria

In-Scope Features:

- User Registration, Login, Logout (JWT).
- Project CRUD (Create, Read, Update Name, Delete).
- Lesson Plan Generation (Subject/Topic/Level input -> Structured Output).
- Worksheet Generation (Subject/Topic/Level input -> Structured Output).
- Parent Update Generation (CSV Input -> Personalized Drafts).
- Content Management (View, Delete generated items).

Success Criteria:

- Frontend successfully authenticates against the backend.
- Data persists in MongoDB Atlas after refresh.
- "AI" generation returns content (mock or real) and saves to DB automatically.
- All manual UI tests pass.

---

3 API Design

Base Path: `/api/v1`
Error Format: `{ "detail": "Error message" }`

Auth

- `POST /auth/signup` - `{email, password, name}` -> `{token, user}`
- `POST /auth/login` - `{email, password}` -> `{token, user}`
- `GET /auth/me` - Validates token -> `{user}`

Projects

- `GET /projects` - List user's projects.
- `POST /projects` - `{name}` -> `{project}`
- `GET /projects/{id}` - Get single project details.
- `PUT /projects/{id}` - `{name}` -> `{project}`
- `DELETE /projects/{id}` - Deletes project & cascading content.

Content (Lesson Plans)

- `GET /lesson-plans` - Query param `?project_id=...`
- `POST /lesson-plans` - `{project_id, subject, level, topic}` -> Triggers Generation -> Saves -> Returns `{lesson_plan}`
- `DELETE /lesson-plans/{id}`

Content (Worksheets)

- `GET /worksheets` - Query param `?project_id=...`
- `POST /worksheets` - `{project_id, subject, level, topic}` -> Triggers Generation -> Saves -> Returns `{worksheet}`
- `DELETE /worksheets/{id}`

Content (Parent Updates)

- `GET /parent-updates` - Query param `?project_id=...`
- `POST /parent-updates/batch-generate` - `{project_id, student_data_csv}` -> Parses CSV -> Generates Batch -> Saves -> Returns `[{parent_update}, ...]`
- `DELETE /parent-updates/{id}`

---

4 Data Model (MongoDB Atlas)

`users`

- `_id`: ObjectId
- `email`: String (Unique, Indexed)
- `password_hash`: String
- `name`: String
- `created_at`: Datetime

projects

- _id: ObjectId
- user_id: ObjectId (Ref: users)
- name: String
- created_at: Datetime
- updated_at: Datetime
- **Example:** { "name": "PSLE Math", "user_id": "..." }

lesson_plans

- _id: ObjectId
- project_id: ObjectId (Ref: projects)
- file_name: String (e.g., "Math-P6-Fractions-LessonPlan.txt")
- content: String (The generated text)
- created_at: Datetime

worksheets

- _id: ObjectId
- project_id: ObjectId (Ref: projects)
- file_name: String
- content: String
- created_at: Datetime

parent_updates

- _id: ObjectId
- project_id: ObjectId (Ref: projects)
- student_name: String
- file_name: String
- draft_text: String
- created_at: Datetime

---

5 Frontend Audit & Feature Map

Component	Frontend Route	Backend Need	Auth	Notes
LoginPage	/login	POST /auth/login	No	
RegisterPage	/register	POST /auth/signup	No	
DashboardPage	/dashboard	GET /projects, POST /projects, PUT /projects/{id}, DELETE /projects/{id}	Yes	Replaces useProjects local logic

Component	Frontend Route	Backend Need	Auth	Notes
`ProjectWorkspacePage`	`/project/:id`	`GET /projects/{id}`	Yes	To validate access
`LessonPlan...Generator`	(Inside Workspace)	`GET /lesson-plans`, `POST /lesson-plans`, `DELETE ...`	Yes	`POST` triggers generation logic
`ParentUpdate...Generator`	(Inside Workspace)	`GET /parent-updates`, `POST /parent-updates/batch-generate`, `DELETE ...`	Yes	Needs CSV parsing logic on backend

6 Configuration & ENV Vars

- `APP_ENV`: `development`
- `PORT`: `8000`
- `MONGODB_URI`: (Required) Atlas Connection String
- `JWT_SECRET`: (Required)
- `JWT_EXPIRES_IN`: `86400` (24h)
- `CORS_ORIGINS`: `http://localhost:5173,http://localhost:3000`

9 Testing Strategy

- **Manual Verification:** Since frontend exists, we test every backend task by performing the action in the UI.
- **Flow:**
    1. Implement Backend Endpoint.
    2. Update Frontend `api.ts` or Context to use Endpoint.
    3. Run Manual Test Step.
    4. If Pass -> Commit.

10 Dynamic Sprint Plan

S0 – Environment Setup & Frontend Connection

Objectives:

- Initialize FastAPI project structure.
- Connect to MongoDB Atlas.
- Configure CORS.
- Setup Git (`.gitignore`).
- Create a global `api` client in frontend (using `axios` or `fetch`) to replace local logic eventually.

Tasks:

- **Task 1: Project Skeleton & DB Connection**
  - Create `backend/main.py`, `backend/core/config.py`, `backend/db/mongodb.py`.
  - Implement `/healthz` endpoint checking DB ping.
  - **Manual Test Step:** Visit `http://localhost:8000/healthz` -> receives `{"status": "ok", "db": "connected"}`.
  - **User Test Prompt:** "Start backend, check health endpoint."
- **Task 2: CORS & Git Init**
  - Enable CORS for frontend origin.
  - Create `.gitignore` (Python + Node).
  - **Manual Test Step:** Check `OPTIONS` request from frontend (via browser console network tab) works.
  - **User Test Prompt:** "Inspect network tab for CORS headers."

S1 – Basic Auth (JWT)

Objectives:

- Secure the app.
- Replace `AuthContext` localStorage logic with API calls.

Tasks:

- **Task 1: User Model & Signup**
  - Create `User` model (Pydantic + Mongo).
  - Implement `POST /api/v1/auth/signup`.
  - **Manual Test Step:** Use Postman/Curl to create user -> Check MongoDB Atlas collection.
  - **User Test Prompt:** "Create user via API tool, verify in DB."
- **Task 2: Login & Token Generation**
  - Implement `POST /api/v1/auth/login`.
  - Return JWT access token.
  - **Manual Test Step:** Login via API tool -> Receive Token.
  - **User Test Prompt:** "Login via API tool, verify token received."
- **Task 3: Frontend Integration (Auth)**
  - Modify `frontend/src/contexts/AuthContext.tsx` to use API instead of `localStorage`.
  - Implement `GET /auth/me` to restore session on refresh.
  - **Manual Test Step:** Register a new user on Frontend -> Redirects to Dashboard -> Refresh page -> Still logged in.
  - **User Test Prompt:** "Register new user on UI, verify dashboard access and persistence."

S2 – Project Management

Objectives:

- Persist Projects in MongoDB.
- Link Projects to Users.

Tasks:

- **Task 1: Project CRUD Backend**
  - Create `Project` model.
  - Implement `GET /projects`, `POST`, `PUT /{id}`, `DELETE /{id}`.
  - Ensure `user_id` is enforced (User can only see their own projects).
  - **Manual Test Step:** API Tool -> Create Project -> List Projects.
  - **User Test Prompt:** "Create/List projects via API tool."
- **Task 2: Frontend Integration (Projects)**
  - Modify `frontend/src/contexts/ProjectContext.tsx` to use API.
  - **Manual Test Step:** Dashboard -> Create "Math Class" -> Refresh -> "Math Class" remains.
  - **Manual Test Step:** Rename "Math Class" to "Science" -> Refresh -> "Science" remains.
  - **User Test Prompt:** "Create, Rename, and Delete a project on the Dashboard."

S3 – Content Generation (Lesson Plans & Worksheets)

Objectives:

- Move "AI" logic to backend.
- Persist generated content.

Tasks:

- **Task 1: Content Models & Endpoints**
  - Create `LessonPlan` and `Worksheet` models in backend.
  - Implement `GET` lists by `project_id` and `DELETE` endpoints.
  - **Manual Test Step:** Manually insert doc in Mongo -> Call GET -> See in response.
  - **User Test Prompt:** "Verify GET endpoints return empty lists initially."
- **Task 2: Generation Logic (The "AI" Service)**
  - Create `POST /lesson-plans` and `POST /worksheets`.
  - Logic: Receive `subject`, `topic`, `level`.
  - **Implementation Detail:** Use the *exact* template strings from the current frontend `ContentContext.tsx` to generate the text on the backend. (Simulate AI for MVP reliability). Save to DB.
  - **Manual Test Step:** Call POST -> Check DB for new document with content.
  - **User Test Prompt:** "Trigger generation via API, check DB."

- **Task 3: Frontend Integration (Content)**
    - Modify `frontend/src/contexts/ContentContext.tsx`.
    - Replace `createLessonPlan` / `createWorksheet` with API calls.
    - Replace `useEffect` loading with API `GET` calls.
    - **Manual Test Step:** Project Workspace -> "Generate Lesson Plan" -> Wait -> Appears in list -> Refresh -> Still there.
    - **User Test Prompt:** "Generate a lesson plan and worksheet in the UI."

🧩 S4 – Parent Update Generation

Objectives:

- Handle CSV parsing and batch generation.

Tasks:

- **Task 1: Parent Update Backend**
    - Create `ParentUpdate` model.
    - Implement `POST /parent-updates/batch-generate`.
    - Logic: Parse incoming `student_data` string (CSV). Loop through lines. Generate text using template. Save all. Return list.
    - Implement `GET` and `DELETE`.
    - **Manual Test Step:** API POST with "John,80,Good" -> Returns 1 update -> Saved in DB.
    - **User Test Prompt:** "Test batch generation via API."
- **Task 2: Frontend Integration (Updates)**
    - Update `ContentContext.tsx createParentUpdate`.
    - **Manual Test Step:** Project Workspace -> Parent Updates Tab -> Paste CSV -> Generate -> Cards appear.
    - **User Test Prompt:** "Generate parent updates for 2 students in UI."