**Backend Project Setup Notes (Vidora Example)**

## **1. Core Concepts**

Before we dive into files/folders, we must understand **why** they exist.

### **1.1 DB (Database)**

* **What it is:** A structured storage for your application’s data.
* **Why we need it:** So that data persists between app restarts (in-memory data would disappear when server restarts).
* **Types:**
  + SQL (MySQL, PostgreSQL) — Table based, relational.
  + NoSQL (MongoDB) — Document based, flexible schema.
* **For Vidora:** MongoDB stores users, videos, comments, likes, subscriptions, etc.

### **1.2 Model**

* **What it is:** The **blueprint** for how data looks and behaves in the database.
* **Why we need it:** Ensures consistency in data structure (e.g., a Video always has a title, url, uploaderId).
* **For Vidora:**  
  Example: video.model.js defines { title: String, description: String, url: String, uploadedBy: ObjectId }

### **1.3 Controller**

* **What it is:** The **logic layer** that handles requests from the client.
* **Why we need it:** Keeps routes clean and separates **business logic** from routing.
* **For Vidora:**  
  video.controller.js → Functions like:
  + uploadVideo(req, res)
  + getAllVideos(req, res)

### **1.4 Route**

* **What it is:** The **entry point** for HTTP requests.
* **Why we need it:** Defines **URLs and HTTP methods** for specific features.
* **For Vidora:**
  + /videos → GET → fetch videos
  + /videos/upload → POST → upload new video

### **1.5 Middleware**

* **What it is:** Functions that run **between** request and controller.
* **Why we need it:** For reusable logic (auth, logging, validation).
* **For Vidora:**  
  Example: auth.middleware.js checks if a user is logged in before letting them upload videos.

### **1.6 Utils**

* **What it is:** Helper functions that can be used anywhere.
* **Why we need it:** Avoid repeating code.
* **For Vidora:**  
  Example: generateVideoThumbnail(url) or formatVideoDuration(seconds)

## **2. Folder Structure in** src

src/

app.js → Main Express app setup

constants.js → App-wide constants (port numbers, env keys)

index.js → Entry point (starts the server)

controllers/ → Business logic for features

db/ → Database connection setup

middlewares/ → Reusable request middlewares

models/ → MongoDB Mongoose models (schemas)

routes/ → Express routes mapping

utils/ → Helper functions/utilities

## **3. Config Files**

### 3.1 .prettierrc

* **Purpose:** Code formatting rules for Prettier.
* **Your config:**
* {
* "singleQuote": false, // Use double quotes
* "bracketSameLine": true, // Keep HTML/JSX closing bracket on same line
* "tabWidth": 2, // Indentation width = 2 spaces
* "trailingComma": "es5", // Add commas where valid in ES5
* "semi": true // Always end with ;

}

### 3.2 .prettierignore

* **Purpose:** Tells Prettier which files/folders NOT to format.
* **Your ignore list:**
* /.vscode // VSCode settings
* /node\_modules // Installed packages (never format)
* ./dist // Build output
* \*.env // Environment variable files
* .env

.env.\* // All env variations

### 3.3 .gitignore

* **Purpose:** Tells Git what NOT to commit.
* Similar to .prettierignore but for **Git**, not Prettier.

## **4. Commands & Flags**

### **4.1** npm init

* Creates package.json → Stores metadata about the project (name, version, dependencies, scripts, etc.)

### **4.2** npm i **vs** npm i -D

* npm i <pkg> → Installs as **dependency** (needed for production).
  + Example: express, mongoose
* npm i -D <pkg> → Installs as **devDependency** (needed only for development).
  + Example: nodemon, prettier

### **4.3** git add .

* Stages all changes for commit.

### **4.4** git commit -m "message"

* Saves staged changes into Git history with a message.

### **4.5** git push -u origin main

* Pushes changes to remote GitHub repo.
* -u sets the **upstream** so future pushes don’t require specifying branch.

### **4.6 Nodemon**

* Watches files for changes and restarts the server automatically.
* Installed with npm i -D nodemon (only needed for dev).

### **4.7 Dev Dependency vs Dependency**

| **Type** | **Installed with** | **Used in Production?** | **Example** |
| --- | --- | --- | --- |
| Dependency | npm i | Yes | express, mongoose |
| DevDependency | npm i -D | No | nodemon, prettier |

## **5. Step-by-step Setup Process (Reusable)**

1. **Initialize project**  
   npm init → fill in name, description, author.
2. **Create repo & push**  
   git init → git remote add origin <url> → git push -u origin main
3. **Install backend dependencies**  
   npm i express mongoose cors dotenv
4. **Install dev dependencies**  
   npm i -D nodemon prettier
5. **Setup folder structure**
6. src/
7. controllers/
8. db/
9. middlewares/
10. models/
11. routes/
12. utils/
13. **Create entry files:** app.js, constants.js, index.js
14. **Configure prettier:** .prettierrc, .prettierignore
15. **Setup Git ignore:** .gitignore
16. **Write server startup code** in index.js
17. **Commit & push**

## **1. What’s a DevDependency?**

In Node.js (or any project using package.json):

* **Dependencies** = packages needed **for your app to run** in production.  
  Example: If you’re using **Express** to serve your app, it’s required when your app is live.
* **DevDependencies** = packages needed **only during development**, not in production.  
  Example: Testing libraries (like Jest), linters (like ESLint), or build tools (like Vite, Webpack).

**In package.json:**

"dependencies": {

"express": "^4.18.2"

},

"devDependencies": {

"eslint": "^8.0.0"

}

When installing:

* npm install <package> → goes to "dependencies"
* npm install <package> --save-dev or -D → goes to "devDependencies"

## **2. Production vs Development**

**Development Environment**

* You’re coding on your machine or test server.
* You use debugging tools, detailed error logs, hot-reloading, etc.
* You include **devDependencies** here.

**Production Environment**

* Your app is deployed for real users.
* You need performance, security, and minimal size.
* You don’t include devDependencies here (to reduce build size & avoid exposing dev tools).

**Rule of thumb:**

* If the code/package is **needed for the app to work** → dependencies
* If it’s **only needed while developing/testing** → devDependencies

## **3. Flags**

A **flag** is an extra option you pass to a command to modify how it runs.  
Example:

npm install express --save-dev

Here --save-dev is a **flag**. It tells npm **“install this as a devDependency”**.

Flags usually start with:

* - (short form) → -D (short for --save-dev)
* -- (long form) → --save-dev

**Examples of flags:**

npm run build --verbose # Shows more output

git commit -m "Message" # -m flag adds a commit message

node server.js --port=3000 # Custom port