Backend Service Specification
Textbook Helper Proof of Concept

Overview
This document specifies a minimal backend service for the Textbook Helper proof of concept. The backend is a dockerized Nest.js application that exposes two API endpoints for login and OCR, uses Tesseract for text extraction, and is designed to be deployed to an EC2 instance.

Goals
Provide a simple authenticated API for:
1. Logging in with hardcoded credentials.
2. Accepting a cropped textbook image and returning extracted text.

Non goals
No persistent database.
No multi tenant support.
No production grade auth or security.
No autoscaling or advanced infrastructure.

Tech Stack
Node.js with Nest.js framework.
TypeScript for type safety.
JWT for simple token based authentication.
Tesseract OCR installed in the Docker image.
Multer for handling multipart file uploads.
Child process execution of the Tesseract CLI.

High Level Architecture
The backend exposes:
POST /api/login
POST /api/ocr

The backend is packaged in a Docker image that includes the Nest.js app and Tesseract. The service will be run on a single EC2 instance for the proof of concept.

Environment Variables
The service reads the following environment variables at startup:
DEMO_USERNAME
Hardcoded demo username allowed to log in.
DEMO_PASSWORD
Hardcoded demo password allowed to log in.
JWT_SECRET
Secret used to sign and verify JWT tokens.
PORT
Port to listen on. Default 4000 if not provided.

API Endpoints

1. POST /api/login

Purpose
Authenticate a user using hardcoded credentials and issue a JWT token.

Request
Method
POST
Path
/api/login

Headers
Content Type application/json
Body (JSON)
{
"username": "string",
"password": "string"
}

Behavior
Validate that username and password match DEMO_USERNAME and DEMO_PASSWORD.
If invalid, return HTTP 401 Unauthorized.
If valid, issue a JWT with payload:
{
"sub": "demo-user",
"name": ""
}
Sign the JWT with JWT_SECRET.

Response on success (HTTP 200)
{
"token": ""
}

Response on failure (HTTP 401)
{
"error": "Invalid credentials"
}

2. POST /api/ocr

Purpose
Accept a cropped textbook image, run OCR, and return the extracted text.

Request
Method
POST
Path
/api/ocr
Headers
Authorization Bearer
Content Type multipart/form-data
Body
image file field containing the cropped image (PNG or JPEG)

Authentication
This endpoint requires a valid JWT token in the Authorization header.
The token is validated using JWT_SECRET.

Behavior
Verify the JWT.
If invalid or missing, return HTTP 401 Unauthorized.
Accept the uploaded file via Multer (field name "image").
Save the file to a temporary path on disk (for example, /tmp/ocr-.jpg).
Invoke Tesseract CLI on the temporary file, for example:
tesseract /tmp/ocr-.jpg stdout -l eng
Capture stdout from the Tesseract process.
Delete the temporary file.

Clean and normalize the text:
Trim leading and trailing whitespace.
Optionally collapse excessive internal whitespace.
Return the cleaned text in the response.

Response on success (HTTP 200)

```
{
"text": ""
}
```

Response on failure (HTTP 400 or 500)

```
{
"error": "Unable to process image"
}
```

Nest.js Project Structure

Suggested minimal structure:

```
src/
app.module.ts
main.ts
auth/
auth.module.ts
auth.controller.ts
auth.service.ts
auth.guard.ts
ocr/
ocr.module.ts
ocr.controller.ts
ocr.service.ts
```

app.module.ts
Import AuthModule and OcrModule.

main.ts
Create Nest application.
Enable CORS for the configured frontend origin.
Listen on process.env.PORT or 4000 by default.

AuthModule
Provides AuthController and AuthService.
Contains the logic for validating credentials and issuing JWTs.
Provides AuthGuard to protect OCR routes.

OcrModule
Provides OcrController and OcrService.
Handles file upload and text extraction.

Authentication Details

JWT Creation
Use a library such as jsonwebtoken or Nest.js JWT module.
Sign JWT with JWT_SECRET.
Set a reasonable expiration (for example, 8 hours) for the proof of concept.

AuthGuard
Read Authorization header.
Expect format "Bearer ".

Verify token with JWT_SECRET.
On success, allow request to proceed.
On failure, return 401 Unauthorized.

OCR Implementation

Tesseract Installation
In the Dockerfile:
Install Tesseract OCR, for example:
apt-get update
apt-get install -y tesseract-ocr

OcrService
Methods:
extractText(file: Express.Multer.File): Promise
Implementation steps:
Write file.buffer to a temporary file on disk.
Execute Tesseract via child_process.spawn or exec.
Collect stdout as the OCR result.
Delete the temporary file.
Normalize and clean the text.
Return the cleaned text.

Error Handling
If Tesseract execution fails or times out, log the error and return an HTTP 500 with an appropriate error message.

CORS Configuration

In main.ts:
Enable CORS to allow calls from the frontend.
For local development:
origin: http://localhost:3000
For deployed frontend:
origin set to the S3 or CloudFront URL of the frontend.

Example:
app.enableCors({
origin: 'http://localhost:3000',
});

Dockerization

Dockerfile requirements:
Base image with Node.js (for example node:18-bullseye).
Install Tesseract OCR.
Copy package.json and package-lock.json.
Run npm install.
Copy source code.
Run npm run build to compile Nest.js.
Set CMD to run node dist/main.js.

Example outline:
FROM node:18-bullseye
RUN apt-get update && apt-get install -y tesseract-ocr
WORKDIR /app
COPY package*.json ./
RUN npm install

COPY . .
RUN npm run build
ENV PORT=4000
CMD ["node", "dist/main.js"]

Running Locally

Prerequisites
Node.js and npm installed.
Tesseract installed on local machine (for non Docker local development).

Steps
Install dependencies:
npm install
Run in development:
npm run start:dev
API will listen on http://localhost:4000

Deployment Overview for EC2

High level steps:
Build Docker image locally.
Tag and push image to Amazon ECR.
Launch an EC2 instance and install Docker.
Pull image from ECR onto EC2.
Run container:
docker run -d -p 4000:4000 --name textbook-backend \
-e DEMO_USERNAME=demo \
-e DEMO_PASSWORD=demo123 \
-e JWT_SECRET=your-secret \
your-account-id.dkr.ecr.region.amazonaws.com/textbook-helper-backend:latest

Update frontend configuration to point to the EC2 public DNS:
VITE_API_BASE_URL=http://:4000

This completes the minimal backend required for the Textbook Helper proof of concept.