

MindForge: a quiz and assessment backend with passwordless access

MindForge is a small assessment product: an operator creates quizzes, invites participants, and collects and reviews their answers. It needed to be quick for participants (no account creation) and simple to operate. I scoped it...

Role: Technical lead; built by a developer on my team under my direction and review

Stack: Node and Express API, MongoDB with Mongoose, JWT, Multer uploads, Nodemailer and Brevo for transactional email, Canvas for server-side image generation, deployed on Vercel

Scope: Quiz authoring, image-based questions, per-user response capture, email and quiz-code login

MindForge · API surface

EXPRESS · MONGODB · JWT · MULTER · BREVO EMAIL · VERCEL · BACKEND-ONLY PROJECT, DRAWN FROM ROUTES/ AND MODELS/

Auth (passwordless)

POST	/email-login	send magic link
GET	/login/:token	verify link + JWT
POST	/quiz-login	enter by quiz code

Quizzes

POST	/create	title · status · questions[] · image
GET	/all	
GET	/:quizId	
PUT	/update/:quizId	
DELETE	/delete/:quizId	

Responses & users

POST	/submit	per user, per quiz
GET	/getResponse/:email	review & score
GET	/:email	user
PUT	/:id	
DELETE	/:id	

Models: quiz { title, status, image, questions[{ questionImage, options[], isCorrect }] } · user · userResponse

API SURFACE DRAWN FROM THE REPOSITORY (BACKEND-ONLY PROJECT)

Context

MindForge is a small assessment product: an operator creates quizzes, invites participants, and collects and reviews their answers. It needed to be quick for participants (no account creation) and simple to operate. I scoped it as a backend-first build with a clear API surface so that a web or mobile client could be attached without changing the server.

What was built

Quiz model

Each quiz has a title, status and an ordered list of questions; questions can carry an image, a set of options, and the correct answer, and the quiz itself can carry a cover image. Images are uploaded through Multer and served from the API.

Passwordless participation

Two entry paths: email login, which sends a one-time link that the API verifies by token, and quiz login, which admits a participant to a specific quiz by code. There are no passwords for participants to forget, which for an assessment tool is the whole point.

Response capture

Submissions are stored per user and per quiz, and can be retrieved by participant email for review and scoring. Quizzes can be created, listed, fetched, updated and deleted through a conventional REST surface.

Operational pieces

JWT-guarded routes for the operator, transactional email through Brevo with Nodemailer as fallback, server-side Canvas for generated images, and a Vercel deployment configuration so the API runs without a server to maintain.

Decisions worth noting

- ◆ **Backend first, client later.** Keeping the product as a clean API meant the client could be a web app, a kiosk, or a mobile app without touching the assessment logic.
- ◆ **Remove the account.** Magic links and quiz codes cost a little in engineering and remove the largest drop-off point in any participant flow.

Why it is in the portfolio

It is a modest, well-scoped team delivery that shows the same habits as the larger builds: a clear data model, a deliberately small surface, and an operator-friendly deployment. Presented at its true size.