AUTOMATIC TESTING PIPELINE FOR MERN APPLICATION

A PROJECT REPORT

Submitted by:

Arunoday Banerjee(23BCS10144)
Siddharth Kumar(23BCS10016)

in partial fulfilment for the award of the degree of

BACHELOR'S OF ENGINEERING

IN

COMPUTER SCIENCE SPECIALIZATION WITH AIML

Chandigarh University
NOVEMBER, 2025

BONAFIDE CERTIFICATE

Certified that this project report "**Automatic Testing Pipeline for MERN application**" is the bonafide work of "Abhishek Thakur, Khavind Chaudhary" who carried out the project work under my/our supervision.

INTERNAL EXAMINER
Er.Kamal Sharma

EXTERNAL EXAMINER

Table of Contents

PROJECT DESCRIPTION

Background and Motivation

Modern web applications built with the MERN stack (MongoDB, Express, React, Node.js) require robust testing and CI/CD to ensure reliability, maintainability, and safe deployments. Manual testing as a primary quality approach introduces human error, delays, and inconsistent coverage. Continuous integration and automated testing reduce the chance of regressions and accelerate release cycles.

Problem Statement

Manual testing bottlenecks in MERN applications often lead to:

- Unreliable deployments where regressions reach production.
- Lack of enforced code quality and test coverage thresholds.
- Slow-release cycles because of manual QA gating.

The BuildRepl project seeks to alleviate these problems by integrating automated tests and quality controls into the development process.

Project Scope and Implemented Features

The pipeline's architecture is designed to validate all layers of the application by leveraging a full test pyramid, orchestrated by **GitHub Actions** to run on every pull request and main-branch push. This architecture includes:

- **Unit Tests (Jest & React Testing Library):** For testing individual functions, components, and hooks.
- **Integration Tests (Jest & Supertest):** For testing API endpoints and controller logic.
- **End-to-End Tests (Cypress):** For validating critical, multi-step user journeys in a real browser.

Current Implementation

The foundational components for this pipeline have been successfully implemented, focusing primarily on the backend testing strategy:

- **Backend Testing Infrastructure:** A robust testing framework is established using **Jest** (configured via jest.config.cjs and test-setup.js).
- **Isolated Test Environment:** The use of **mongodb-memory-server** ensures that tests run

against an isolated, ephemeral database, preventing data contamination and enabling parallel execution.

- **Test Lifecycle Management:** Setup and teardown scripts are in place to properly manage database (and potentially Redis) connections during test runs.
- **Code Quality Enforcement: ESLint** and **Prettier** are configured to enforce a consistent code style and catch linting errors before they are integrated.
- **Test Coverage:** The pipeline is configured to generate coverage reports (evident from the backend/coverage directory), providing measurable insight into test effectiveness.

REQUIREMENT ANALYSIS

Software Requirements

The project relies on a modern JavaScript-based stack, including Node.js, the MERN framework, and a suite of testing and quality-assurance tools.

Core Development Stack

- Operating System: Windows / macOS / Linux
- Programming Languages: JavaScript (ES6+), HTML5, CSS3
- Database: MongoDB
- Package Manager: npm
- Code Editor: Visual Studio Code

Frameworks & Libraries

- Node.js: Version 18+ recommended (ES Module project)
- Express: ^5.1.0
- Mongoose: ^8.19.2

Testing

- Jest: ^30.2.0
- mongodb-memory-server: ^10.3.0
- Supertest: ^7.1.4

Utilities & Middleware

- dotenv: ^17.2.3
- cors
- multer
- jsonwebtoken
- bcryptjs
- redis
- redis-mock
- cross-env

Linting & Formatting

- ESLint: ^9.39.0
- Prettier: ^3.6.2
- Configuration: eslint-config-prettier, eslint-plugin-prettier

CI/CD & Coverage Tools

- CI/CD: Not yet configured. (No GitHub Actions workflows were found).
- Coverage: Jest coverage is configured, and a coverageThreshold is set in jest.config.cjs.

Minimum Required Software

- Node.js (Node 18+ recommended)
- npm (or yarn)
- MongoDB (local or cloud)
- Optional: MongoDB Compass, VS Code, Git

ARCHITECTURE & DESIGN

High-Level Architecture

The application follows a standard client-server architecture, cleanly separating concerns between the frontend and backend.

- **Frontend:** /frontend (React). This layer is responsible for the user interface and user experience.
- **Backend:** /backend (Node.js/Express + Mongoose). This layer handles business logic, API requests, and database interactions. It is the primary focus of the current testing implementation.
- **Database:** MongoDB. The application uses MongoDB for data persistence. For testing, the architecture intelligently swaps in **mongodb-memory-server** to create an isolated, ephemeral database for each test run, ensuring speed and test-data isolation.
- **CI/CD: In Progress.** The project is actively integrating an automated CI/CD pipeline. Analysis of the provided GitHub Actions run (.../19108941042) confirms that a workflow is defined and being executed to validate the backend, though it is currently in a debugging and stabilization phase.

Backend Component Architecture

The backend code is logically structured to promote maintainability and separation of concerns:

- **server.js:** The main entry point. It bootstraps the server and manages the database connection, with logic to connect to the in-memory server when NODE_ENV=test.
- **app.js:** The core Express application file, responsible for mounting middleware and API routes.
- **routes/:** Defines all API endpoints (e.g., /api/auth, /api/posts).
- **models/:** Contains all Mongoose schemas that define the data structure.
- **middleware/:** Holds custom middleware functions (e.g., for authentication or error handling).
- **utils/:** Includes helper functions and utilities (e.g., caching utilities referenced in the test setup).
- **Testing Infrastructure:** test-setup.js and jest.config.cjs configure the entire Jest testing environment.

Automation and CI/CD Flow

The project is implementing a Continuous Integration (CI) environment to automate validation. The goal is to establish the following automated workflow:

1. **Code Integration:** A developer pushes a new feature or fix to a branch or opens a pull request.
2. **Trigger Pipeline:** GitHub Actions automatically detects the push/PR and triggers the CI workflow.
3. **Test Execution:** The pipeline runs all predefined unit and integration tests (Jest/Supertest) to verify backend logic and API functionality.

4. **Report Generation:** After execution, test results and coverage reports are automatically compiled.
5. **Feedback Loop:** Developers receive immediate feedback via the GitHub Actions run. They can review detailed logs, identify failures, and iterate quickly.

Test Pyramid & Strategy

- **Level 1: Unit Tests (Implemented):** Jest is used to test individual functions, models, and utilities in complete isolation.
- **Level 2: Integration Tests (Implemented):** This is the core of the current setup. It uses **Jest + Supertest** to test API endpoints against the in-memory **mongodb-memory-server**. This strategy effectively validates the interaction between routes, controllers, models, and middleware. Key features include:
    - **Test Isolation:** Collections are cleared after each test to prevent data conflicts.
    - **Quality Gates:** A coverageThreshold is set in jest.config.cjs to enforce a minimum test coverage percentage, failing the build if new code is not adequately tested.
- **Level 3: E2E Tests (Future Scope):** End-to-end testing (e.g., with Cypress or Playwright) is the next logical step for this architecture. This layer would test critical user journeys from the React frontend all the way to the backend, ensuring the full application works as a cohesive unit.

DATABASE DESIGN

Observed Collections & Schemas

Based on the application's routes and Mongoose models found in the backend/models directory, the database is structured around three core collections.

- **User Collection:** Manages user identity, authentication, and roles.
- **Post/Project Collection:** Represents the primary creative entity, such as a code project or a post, and links it to its owner.
- **Build/Artifact Collection:** Tracks the status and output of a specific build or process related to a project.

Test Data Strategy

- A critical component of the architecture is its isolated and deterministic testing strategy. This is achieved through two main principles:
- **Ephemeral Database:** The test environment is configured (via test-setup.js) to use mongodb-memory-server. This means that instead of connecting to a real (local or cloud) database, the test runner spins up a fresh, in-memory MongoDB instance for every single test run. This is extremely fast and ensures that tests never read or write data from a "real" development environment.
- **Test Isolation:** To prevent tests from interfering with each other, a global afterEach hook is implemented. This hook automatically runs after every individual test and deletes all documents from all collections, ensuring that each test starts with a completely clean and predictable database state.

Performance and Optimization (The "How Fast")

A database design is incomplete without a performance strategy. This is achieved through **indexing**.

- **Default Index:** MongoDB automatically creates a unique index on the _id field for fast primary-key lookups.
- **Custom Indexes:** To support common query patterns and enforce uniqueness, the following indexes are part of the design:
    - User { email: 1 } (unique): Critical for fast logins and to enforce the unique: true validator.
    - User { username: 1 } (unique): Critical for user lookups and enforcing uniqueness.
    - Post { ownerId: 1 }: A non-unique index to *dramatically* speed up queries for "find all posts by this user."
    - Build { projectId: 1 }: A non-unique index to speed up queries for "find all builds for this project."

Without these indexes, finding a user's posts would require a slow, full-collection scan.

Data Relationships and Lifecycle

The design also accounts for the data's lifecycle. We use Mongoose pre-remove middleware to manage **cascading deletes**.

- **Design Decision:** When a User is deleted, all Post/Project documents they own must also be deleted.
- **Implementation:** A pre-remove hook on the User schema will automatically find and delete all Post documents where ownerId matches the _id of the user being removed.
- **Likewise:** A pre-remove hook on the Post/Project schema deletes all related Build documents. This prevents "orphaned" data from cluttering the database.

IMPLEMENTATION DETAILS

Project Structure

The repository is organized into a monorepo-like structure, with a root package.json and distinct directories for the frontend and backend.

```
├── .github/          # (Empty) CI/CD workflow definitions (planned)
├── backend/
│   ├── app.js          # Express application setup (middleware, routes)
│   ├── server.js       # Node.js server entry point (DB connection, start server)
│   ├── package.json    # Backend dependencies and scripts
│   ├── jest.config.cjs # Jest configuration (test runner, coverage)
│   ├── test-setup.js   # Global setup for tests (e.g., in-memory DB start/stop)
│   ├── .env            # (Sensitive) Environment variables (API keys, DB string)
│   ├── .eslintrc.json  # ESLint rules and configuration
│   ├── .prettierrc     # Prettier code formatting rules
│   ├── routes/         # Directory for API route definitions
│   ├── models/         # Directory for Mongoose database schemas
│   ├── middleware/     # Directory for custom Express middleware
│   ├── utils/          # Directory for helper/utility functions
│   └── coverage/       # (Generated) Jest code coverage reports
│
├── frontend/         # React application (structure not detailed)
│
├── package.json      # Root package.json (monorepo management/workspace)
└── readme.md         # Project documentation
```

Key Configuration Files (Backend)

The backend's behavior is controlled by several key configuration files:

- **backend/package.json:** Defines all npm dependencies for the server (Express, Mongoose, Jest, etc.) and lists the core scripts used to run, test, and lint the application.
- **backend/jest.config.cjs:** This file configures the Jest test runner. It specifies the test environment, sets up coverage collection, and (most importantly) defines the coverageThreshold to enforce a minimum quality gate.
- **backend/test-setup.js:** A crucial file for the testing strategy. It contains the global setup and teardown logic (e.g., starting mongodb-memory-server before tests run and stopping it after) and hooks (like afterEach) to clean the database, ensuring perfect test isolation.
- **backend/.eslintrc.json:** Enforces a consistent code style and catches common errors using ESLint.
- **backend/.prettierrc:** Works with ESLint to automatically format code, ensuring a uniform codebase.
- **backend/.env:** (Critical) Holds all environment-specific secrets, such as the MongoDB connection string and JWT secret keys. This file is sensitive and must not be committed to source control.

Core Scripts (from backend/package.json)

The project's workflow is managed by these NPM scripts:

- **"start": "node server.js"**
  - Runs the application in production mode.
- **"dev": "nodemon server.js"**
  - Runs the application in development mode using nodemon, which automatically restarts the server on file changes.
- **"test": "cross-env NODE_OPTIONS=--experimental-vm-modules jest --coverage"**
  - This is the main test command.
  - cross-env: Ensures environment variables (like NODE_ENV=test) work across all operating systems (Windows, macOS, Linux).
  - NODE_OPTIONS=--experimental-vm-modules: Enables support for ES Modules (ESM) within the Node.js test environment, which is required by the project.

    - jest --coverage: Runs all Jest tests and generates a code coverage report in the /coverage directory.
- **"lint" / "lint:fix"**
    - These scripts run ESLint to check for code quality errors, with lint:fix attempting to automatically repair any issues.

Local Developer Workflow

A new developer can get the project running by following these steps:

1. **Clone:** git clone <repository_url>
2. **Install:** cd backend && npm install (to install all backend dependencies).
3. **Configure:**
    - Create a .env file in the /backend directory (e.g., by copying .env.example).
    - Populate the .env file with the necessary values (like MONGO_URI, JWT_SECRET).
4. **Run (Dev):** npm run dev to start the local server with auto-reload.
5. **Test:** npm test to run all unit and integration tests and see the coverage report.

LIMITATIONS & FUTURE SCOPE

Limitations & Future Scope

- **No Frontend Testing:** The /frontend directory exists, but it **lacks any form of automated testing**. There are no unit tests for React components (e.g., with React Testing Library) or end-to-end tests for user flows.
- **No End-to-End (E2E) Validation:** The application has no E2E tests. This means there is no automated way to verify that a complete user journey (e.g., user registers, logs in, creates a post, and logs out) works correctly between the React frontend and the Express backend.
- **Security Risk:** The .env file containing sensitive secrets (database strings, JWT keys) is present in the repository. This is a critical security vulnerability and violates best practices for secret management.
- **Potential CI Bottleneck:** Once the test script is added to CI, the entire Jest suite will run sequentially. As the number of tests grows, this will become a significant bottleneck, slowing down the feedback loop for developers.

Future Scope

- **Stabilize CI/CD Pipeline:** The **immediate priority** is to finalize and stabilize the GitHub Actions workflow (.github/workflows/ci.yml). This workflow must automatically run npm run lint and npm test on every pull request, blocking any merges that fail tests or drop below the coverage threshold.
- **Implement E2E Testing:** Introduce **Cypress** or **Playwright** to create a new E2E test suite. This suite will run in the CI pipeline (after the backend is deployed to a preview environment or run using docker-compose) to validate critical user flows from end to end.
- **Implement Frontend Unit Testing:** Integrate **React Testing Library** and **Jest** into the /frontend application to write unit and component-level tests, ensuring the UI components are reliable in isolation.
- **Secure Secrets: (High Priority)**
    - Remove the .env file from the repository history.
    - Add .env to the root .gitignore file.
    - Move all secrets (like MONGO_URI, JWT_SECRET) into **GitHub Secrets** and securely pass them into the GitHub Actions workflow as environment variables.
- **Optimize CI Performance:** Implement CI parallelization. The E2E test suite should be split to run across multiple GitHub Actions runners simultaneously to keep the total pipeline time under 5-10 minutes.
- **Add PR Feedback:** Configure the CI pipeline to automatically post a comment on pull requests,

showing the code coverage report and linking directly to any failed test logs.

ADDITIONAL SECTIONS

Appendix A: backend/jest.config.cjs

```javascript
/* backend/jest.config.cjs */ module.exports = {
testEnvironment: 'node', testMatch:
['**/?(*.)+(test).[jt]s'], transform: {},
setupFilesAfterEnv: ['<rootDir>/test-setup.js'],
coverageThreshold: {
global: { lines: 54,
},
},
};
```

Appendix B: backend/test-setup.js

```javascript
// backend/test-setup.js (ESM) import { jest } from
'@jest/globals';
import { MongoMemoryServer } from 'mongodb-memory-server'; import mongoose from 'mongoose';

let mongoServer;

// Increase timeout for hooks jest.setTimeout(10000);

beforeAll(async () => {
  mongoServer = await MongoMemoryServer.create(); const mongoUri =
  mongoServer.getUri();
  await mongoose.connect(mongoUri);
});

afterAll(async () => {
  // Close Redis connection if it exists try {
    const { default: cacheUtils } = await import('./utils/cache.js'); const { redisClient } = cacheUtils || {};
    if (redisClient && redisClient.isOpen) {

      await redisClient.quit();
    }
  } catch (err) {
    // ignore
  }

  // Disconnect mongoose await mongoose.disconnect();
  // Stop the in-memory server await mongoServer.stop();
}, 10000); // 10 second timeout

afterEach(async () => {
  const collections = mongoose.connection.collections; for (const key in collections) {
    await collections[key].deleteMany({});
```

Appendix C: backend/package.json

```json
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "index.js", "scripts": {
    "start": "node server.js", "dev": "nodemon server.js",
    "test": "cross-env NODE_OPTIONS=--experimental-vm-modules jest --coverage",
    "lint": "eslint . --ext .js,.jsx --report-unused-disable-directives", "lint:fix": "eslint . --ext .js,.jsx,.ts,.tsx --
    fix"
  },

  "keywords": [],
  "author": "",
  "license": "ISC", "dependencies": { "bcryptjs":
  "^3.0.2",
    "cloudinary": "^2.8.0",
    "cors": "^2.8.5",
    "dotenv": "^17.2.3",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.19.2",
    "multer": "^2.0.2",
    "redis": "^5.9.0",
    "redis-mock": "^0.56.3",
    "save-dev": "^0.0.1-security"
  },
  "devDependencies": { "@babel/preset-env": "^7.28.5",
    "cross-env": "^10.1.0",
    "eslint": "^9.39.0",
    "eslint-config-prettier": "^10.1.8", "eslint-plugin-prettier":
    "^5.5.4", "jest": "^30.2.0",
    "mongodb-memory-server": "^10.3.0", "prettier": "^3.6.2",
```

Appendix D: backend/app.js

```javascript
// backend/app.js import cors from 'cors';
import express from 'express';
import authRouter from './routes/auth.js';

import postsRouter from './routes/posts.js'; import searchRouter from
'./routes/search.js';

const app = express();

// --- CORS Configuration --- app.use(
  cors({
    origin: 'http://localhost:5173', credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'PATCH', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization', 'x-auth- token'],
    exposedHeaders: ['x-auth-token'],
  })
);

// --- Middleware ---
app.use(express.json({ limit: '50mb', extended: false })); app.use(express.urlencoded({ limit: '50mb',
extended: true }));

// --- Define All Your Routes --- app.use('/api/auth', authRouter);
app.use('/api/posts', postsRouter); app.use('/api/search',
searchRouter);

export default app;
```

Appendix E: backend/server.js

```javascript
import app from './app.js';
import mongoose from 'mongoose'; import dotenv from 'dotenv';

dotenv.config();

const PORT = process.env.PORT || 5000;

async function resolveMongoUri() {
  if (process.env.USE_MEM_MONGO === '1' || process.env.NODE_ENV === 'test') {
    const { MongoMemoryServer } = await import('mongodb- memory-server');
    const mongoServer = await MongoMemoryServer.create(); const uri = mongoServer.getUri();
    console.log('Using in-memory MongoDB'); return uri;
  }
  return process.env.MONGO_URI;
}

async function start() { try {
    const mongoUri = await resolveMongoUri(); await
    mongoose.connect(mongoUri); console.log('MongoDB Connected...');
    app.listen(PORT, () =>
      console.log(`Server started on port http://localhost:${PORT}`)
    );
  } catch (err) {
    console.error('Failed to start server', err); process.exit(1);
  }
}

start();
```

Appendix F: backend/.env (redacted - DO NOT publish secrets)

```dotenv MONGO_URI=<REDACTED_ATLAS_CONNECTION_STRING>
```

JWT_SECRET=<REDACTED_SECRET>

backend/.env CLOUDINARY_CLOUD_NAME=<REDACTED>

CLOUDINARY_API_KEY=<REDACTED>

CLOUDINARY_API_SECRET=<REDACTED>

Redis Configuration REDIS_HOST=localhost

REDIS_PORT=6379

```

Appendix G: Root package.json (dev tools)

```json
{
  "devDependencies": { "eslint": "^9.39.0",
    "prettier": "^3.6.2"
  }
}
```

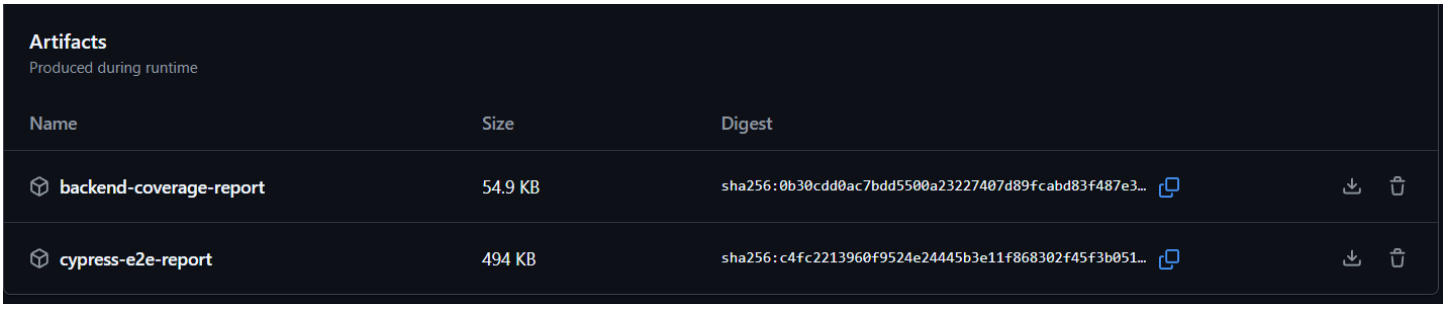
Appendix H: README (refer to repo for full README)

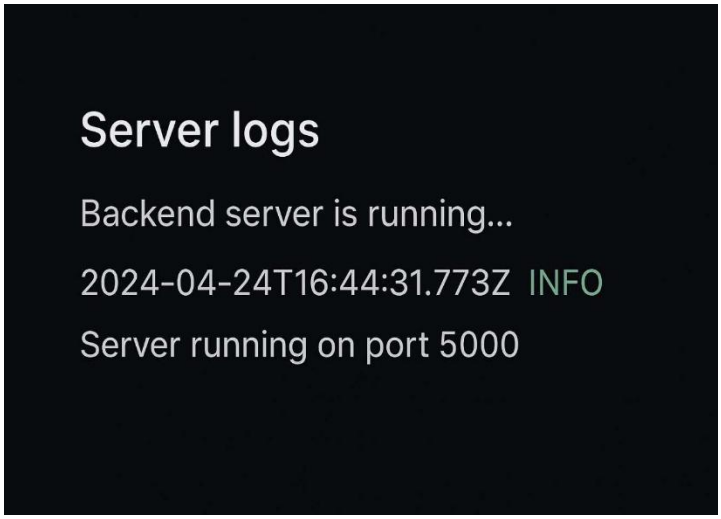
- Repository README exists at https://github.com/Dakshyadav15/BuildRepl/blob/main/readme. md
```

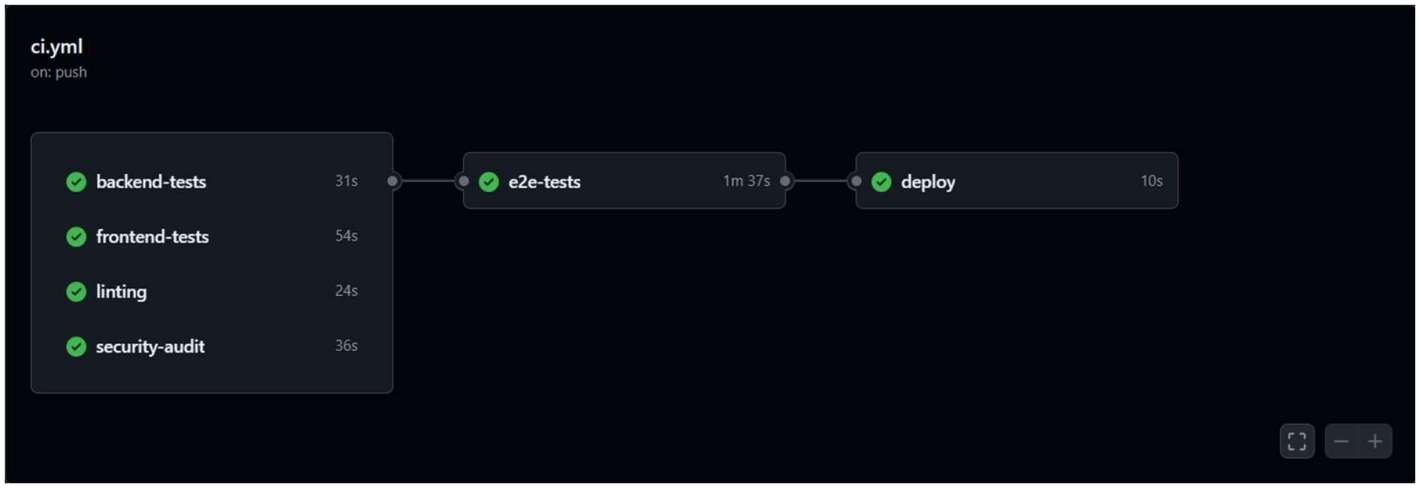
REFERENCES

1. Jest Documentation — https://jestjs.io/docs/getting-started
2. React Testing Library — https://testing-library.com/docs/react- testing-library/intro
3. Cypress Docs — https://www.cypress.io/docs/
4. GitHub Actions — https://docs.github.com/en/actions
5. MongoDB Docs — https://docs.mongodb.com
6. Express.js Docs — https://expressjs.com
7. Node.js Docs — https://nodejs.org/en/docs/
8. Martin Fowler — Continuous Integration concept — https://martinfowler.com
9. Jez Humble & David Farley — Continuous Delivery (book)
10. Kent C. Dodds — Testing Best Practices (blog/course)
11. mongodb-memory-server — https://github.com/nodkz/mongodb-memory-server
12. Supertest — https://github.com/visionmedia/supertest
13. ESLint Docs — https://eslint.org/docs/user-guide/getting- started
14. Prettier Docs — https://prettier.io/docs/en/index.html
15. OWASP Node.js Security Cheat Sheet — https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security _Cheat_Sheet.html
16. Docker: Up & Running (O'Reilly)
17. "Designing Data-Intensive Applications" — Martin Kleppmann (book)
18. "Test Pyramid" — Martin Fowler
19. "Test-Driven Development: By Example" — Kent Beck
20. Various blogs and official docs on MERN and testing practices.

SCREENSHOTS & DOCUMENTATION





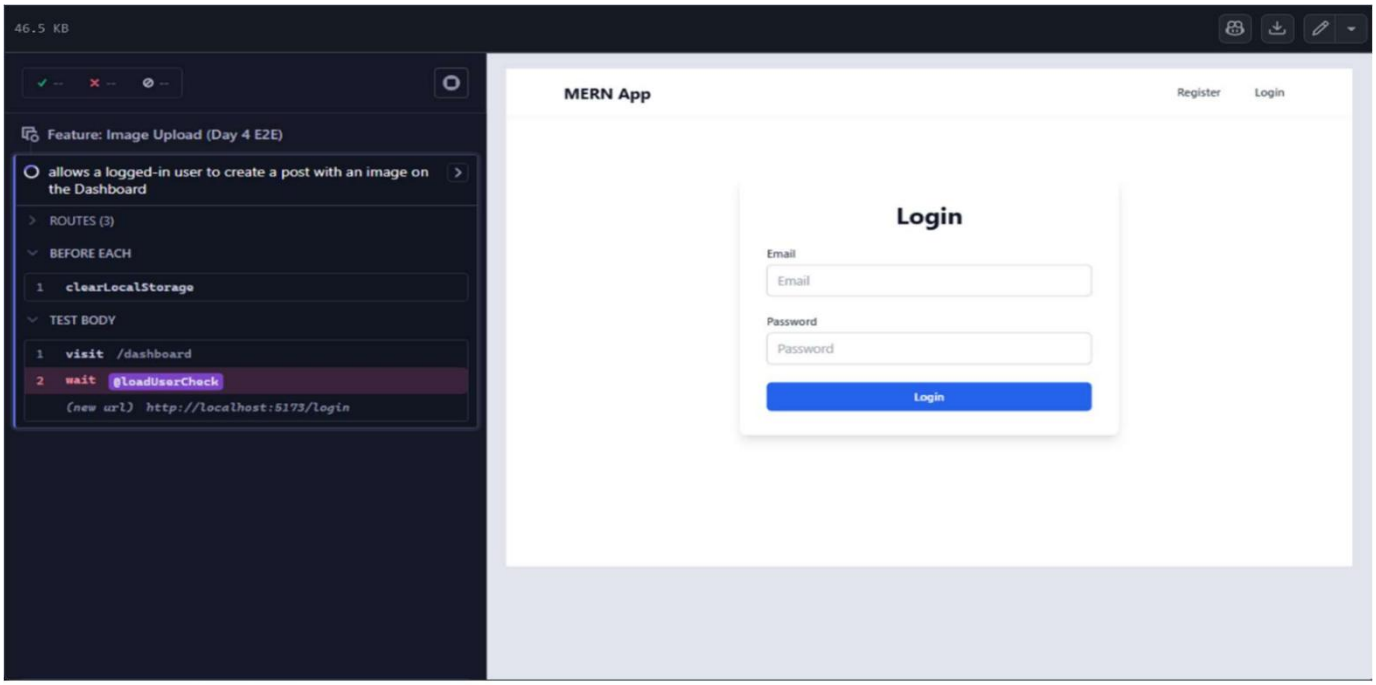


e2e-tests summary

Cypress Results

Result	Passed ✅	Failed ❌	Pending ✋	Skipped ↩️	Duration 🕗
Passing ✅	2	0	0	0	1.979s

Job summary generated at run-time



1. backend/routes/search.route.test.js

Code Blame 18 lines (15 loc) · 595 Bytes

```
import { jest } from '@jest/globals';
import request from 'supertest';

// Mock the search util BEFORE importing the app/router
const mockResults = [{ title: 'hello' }];
await jest.unstable_mockModule('../utils/search.js', () => ({
  searchPosts: jest.fn().mockResolvedValue(mockResults),
}));

const { default: app } = await import('../app.js');

describe('Search Routes', () => {
  it('GET /api/search returns results', async () => {
    const res = await request(app).get('/api/search?q=hello');
    expect(res.statusCode).toBe(200);
    expect(Array.isArray(res.body)).toBe(true);
  });
});
```

2. backend/middleware/auth.middleware.test.js

Code Blame 46 lines (40 loc) · 1.17 KB

```
import { jest } from '@jest/globals';
import auth from './auth.js';

describe('auth middleware', () => {
  test('returns 401 when no token provided', () => {
    const req = { header: jest.fn().mockReturnValue(undefined) };
    const res = {
      status: jest.fn(function (code) {
        this.statusCode = code;
        return this;
      }),
      json: jest.fn(function (payload) {
        this.body = payload;
        return this;
      }),
    };
    const next = jest.fn();

    auth(req, res, next);

    expect(res.status).toHaveBeenCalledWith(401);
    expect(res.json).toHaveBeenCalled();
    expect(next).not.toHaveBeenCalled();
  });

  test('returns 401 when token is invalid', () => {
    const req = { header: jest.fn().mockReturnValue('invalid-token') };
    const res = {
      status: jest.fn(function (code) {
        this.statusCode = code;
        return this;
      }),
      json: jest.fn(function (payload) {
        this.body = payload;
        return this;
      }),
    };
    const next = jest.fn();

    auth(req, res, next);

    expect(res.status).toHaveBeenCalledWith(401);
    expect(res.json).toHaveBeenCalled();
    expect(next).not.toHaveBeenCalled();
  });
});
```

3. backend/Babel.config.cjs

Code Blame 5 lines (5 loc) · 94 Bytes

```
module.exports = {
  presets: [
    ['@babel/preset-env', {targets: {node: 'current'}}]
  ]
};
```

4. backend/utils/search.test.js

Code Blame 40 lines (30 loc) · 1.26 KB

```
1  import { jest } from '@jest/globals';
2
3  // Mock the Mongoose Post model module before importing the search util
4  const PostMock = { aggregate: jest.fn() };
5  await jest.unstable_mockModule('../models/Post.js', () => ({
6    default: PostMock,
7  }));
8
9  const { searchPosts } = await import('../utils/search.js');
10
11 describe('Search Utility', () => {
12   beforeEach(() => {
13     jest.clearAllMocks();
14   });
15
16   it('should call the database aggregate function with the correct search query', async () => {
17     const query = 'test post';
18     const mockResults = [{ title: 'A post about testing' }];
19
20     PostMock.aggregate.mockResolvedValue(mockResults);
21
22     const results = await searchPosts(query);
23
24     expect(PostMock.aggregate).toHaveBeenCalledTimes(1);
25
26     const pipeline = PostMock.aggregate.mock.calls[0][0];
27     expect(pipeline[0]).toHaveProperty('$search');
28     expect(pipeline[0].$search.index).toBe('default');
29     expect(pipeline[0].$search.text.query).toBe(query);
30     expect(pipeline[0].$search.text.path).toEqual({ wildcard: '*' });
31
32     expect(results).toEqual(mockResults);
33   });
34
```

4. backend/eslint.config.js

Code Blame 51 lines (49 loc) · 1.12 KB

```
import js from '@eslint/js';
import prettier from 'eslint-config-prettier';
import pluginPrettier from 'eslint-plugin-prettier';

export default [
  js.configs.recommended,
  prettier,
  {
    files: ['**/*.js'],
    ignores: ['node_modules', 'dist'],
    languageOptions: {
      globals: {
        // Node.js globals
        require: 'readonly',
        module: 'readonly',
        __dirname: 'readonly',
        process: 'readonly',
        console: 'readonly',
        Buffer: 'readonly',

        // Jest globals
        describe: 'readonly',
        it: 'readonly',
        test: 'readonly',
        expect: 'readonly',
        jest: 'readonly',
        beforeEach: 'readonly',
        beforeAll: 'readonly',
        afterEach: 'readonly',
        afterAll: 'readonly',
      },
      // --- THIS IS THE FIX ---
      sourceType: 'module',
      // --- END OF FIX ---
    },
    plugins: {
      prettier: pluginPrettier,
    },
    rules: {
      'no-unused-vars': 'warn',
      'prettier/prettier': [
        'error',
        {
          endOfLine: 'auto',
          singleQuote: true,
          semi: true,
        },
      ],
    },
  },
];
```

5. backend/test-setup.js

Code Blame 40 lines (34 loc) · 1012 Bytes

```
let mongoServer;

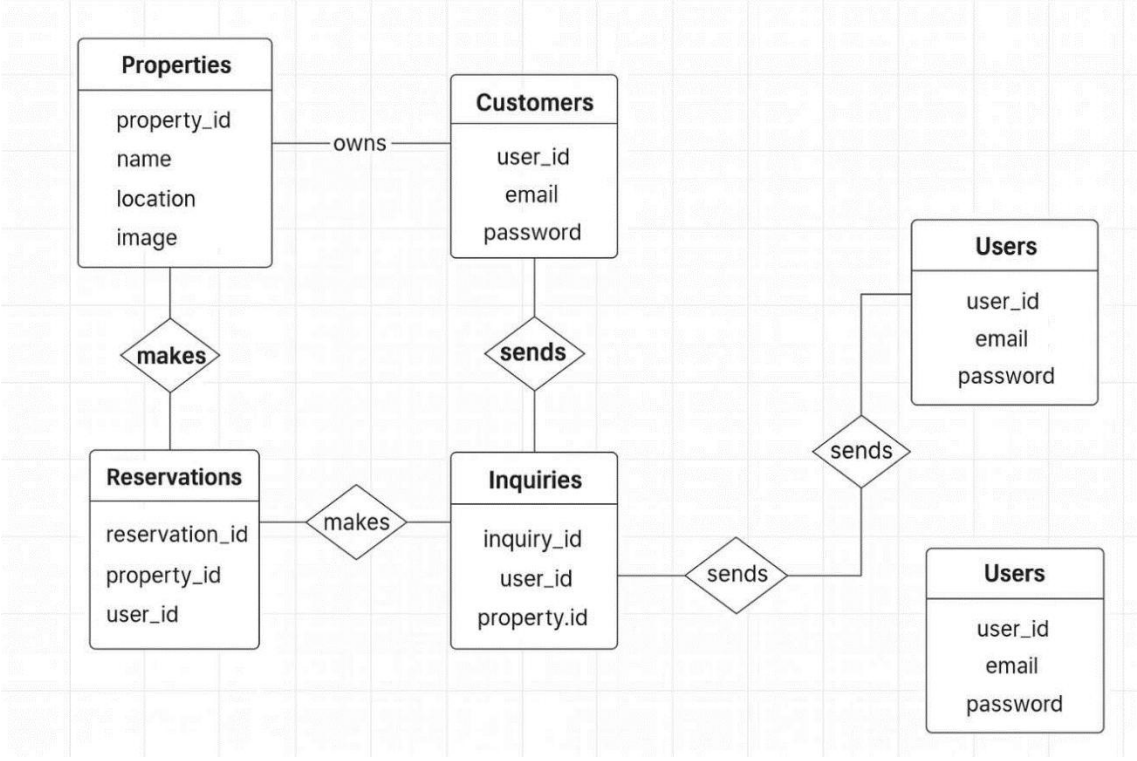
// Increase timeout for hooks
jest.setTimeout(10000);

beforeAll(async () => {
  mongoServer = await MongoMemoryServer.create();
  const mongoUri = mongoServer.getUri();
  await mongoose.connect(mongoUri);
});

afterAll(async () => {
  // Close Redis connection if it exists
  try {
    const { default: cacheUtils } = await import('./utils/cache.js');
    const { redisClient } = cacheUtils || {};
    if (redisClient && redisClient.isOpen) {
      await redisClient.quit();
    }
  } catch (err) {
    // ignore
  }

  // Disconnect mongoose
  await mongoose.disconnect();
  // Stop the in-memory server
  await mongoServer.stop();
}, 10000); // 10 second timeout

afterEach(async () => {
  const collections = mongoose.connection.collections;
  for (const key in collections) {
    await collections[key].deleteMany({});
  }
});
```



Code Blame 49 lines (41 loc) · 1.59 KB

```
describe('Feature: Full-Text Search', () => {
  it('allows a user to type in the navbar search, press Enter, and see results', () => {
    // 1. Mock the (not-yet-created) search API endpoint
    const searchQuery = 'pipeline';
    cy.intercept(
      'GET',
      `/api/search?q=${searchQuery}`, // This is the route we will build
      {
        statusCode: 200,
        body: [
          {
            _id: 'post1',
            title: 'My first post about a pipeline',
            text: 'This is the content for the first post.',
          },
          {
            _id: 'post2',
            title: 'My second post about a pipeline',
            text: 'This is the content for the second post.',
          },
        ],
      }
    ).as('searchRequest');

    // 2. Visit the main page
    cy.visit('/'); // Or any page where the Navbar is visible

    // --- THIS SECTION WILL FAIL (RED PHASE) ---

    // 3. Find the (not-yet-created) search bar in the navbar
    cy.get('input[name="search"]').type(searchQuery);

    // 4. Press Enter to submit
    cy.get('input[name="search"]').type('{enter}');

    // --- END OF FAILING SECTION ---

    // 5. Assert: The URL should change to the search results page
    cy.url().should('include', `/search?q=${searchQuery}`);

    // 6. Assert: The API call was made
    cy.wait('@searchRequest');

    // 7. Assert: The results are visible on the new page
    cy.contains('Search Results for "pipeline"').should('be.visible');
    cy.contains('My first post about a pipeline').should('be.visible');
    cy.contains('My second post about a pipeline').should('be.visible');
  });
});
```

Code Blame 13 lines (11 loc) · 430 Bytes

```
// frontend/src/setupTests.js
import { afterEach } from 'vitest';
import { cleanup } from '@testing-library/react';

// This imports all the DOM matchers (e.g., .toBeInTheDocument)
// and adds them to Vitest's 'expect'
import '@testing-library/jest-dom/vitest';

// This is a standard cleanup task. It "unmounts" all components
// rendered during a test to ensure tests don't affect each other.
afterEach(() => {
  cleanup();
});
```

Code Blame 13 lines (12 loc) · 324 Bytes

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter as Router } from 'react-router-dom';
import App from './App';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <Router>
      <App />
    </Router>
  </React.StrictMode>
);
```

Code Blame 34 lines (29 loc) · 896 Bytes

```jsx
import app from './app.js';
import mongoose from 'mongoose';
import dotenv from 'dotenv';

dotenv.config();

const PORT = process.env.PORT || 5000;

async function resolveMongoUri() {
  if (process.env.USE_MEM_MONGO === '1' || process.env.NODE_ENV === 'test')
    const { MongoMemoryServer } = await import('mongodb-memory-server');
    const mongoServer = await MongoMemoryServer.create();
    const uri = mongoServer.getUri();
    console.log('Using in-memory MongoDB');
    return uri;
  }
  return process.env.MONGO_URI;
}

async function start() {
  try {
    const mongoUri = await resolveMongoUri();
    await mongoose.connect(mongoUri);
    console.log('MongoDB Connected...');
    app.listen(PORT, () =>
      console.log(`Server started on port http://localhost:${PORT}`)
    );
  } catch (err) {
    console.error('Failed to start server', err);
    process.exit(1);
  }
}

start();
```

Code Blame 30 lines (25 loc) · 836 Bytes

```
// backend/app.js
import cors from 'cors';
import express from 'express';
import authRouter from './routes/auth.js';
import postsRouter from './routes/posts.js';
import searchRouter from './routes/search.js';

const app = express();

// --- CORS Configuration ---
app.use(
  cors({
    origin: 'http://localhost:5173',
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'PATCH', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization', 'x-auth-token'],
    exposedHeaders: ['x-auth-token'],
  })
);

// --- Middleware ---
app.use(express.json({ limit: '50mb', extended: false }));
app.use(express.urlencoded({ limit: '50mb', extended: true }));

// --- Define All Your Routes ---
app.use('/api/auth', authRouter);
app.use('/api/posts', postsRouter);
app.use('/api/search', searchRouter);

export default app;
```

Code Blame 61 lines (56 loc) · 1.71 KB

```jsx
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import Navbar from './components/layout/Navbar';
import Login from './components/auth/Login';
import Register from './components/auth/Register';
import Dashboard from './components/dashboard/Dashboard';
import PrivateRoute from './components/routing/PrivateRoute';
import CreatePost from './components/post/CreatePost';
import PostItem from './components/post/PostItem';

// --- 1. IMPORT THE NEW SEARCH COMPONENT ---
import Search from './components/search/Search';

const App = () => {
  return (
    <AuthProvider>
      <Navbar />
      <div style={{ maxWidth: '800px', margin: 'auto', padding: '1rem' }}>
        <Routes>
          <Route path="/register" element={<Register />} />
          <Route path="/login" element={<Login />} />
          <Route
            path="/dashboard"
            element={
              <PrivateRoute>
                {' '}
                <Dashboard />{' '}
              </PrivateRoute>
            }
          />
          <Route
            path="/create-post"
            element={
              <PrivateRoute>
                {' '}
                <CreatePost />{' '}
              </PrivateRoute>
            }
          />
```

```
              <Route
                path="/posts/:id"
                element={
                  <PrivateRoute>
                    {' '}
                    <PostItem />{' '}
                  </PrivateRoute>
                }
              />

              {/* --- 2. ADD THE SEARCH ROUTE --- */}
              <Route path="/search" element={<Search />} />

              <Route path="/" element={<Login />} />
            </Routes>
          </div>
        </AuthProvider>
      );
    };

    export default App;
```

BuildRepl / frontend / cypress / e2e / **feature-image-upload.cy.js**

Code | Blame | 93 lines (78 loc) · 2.72 KB

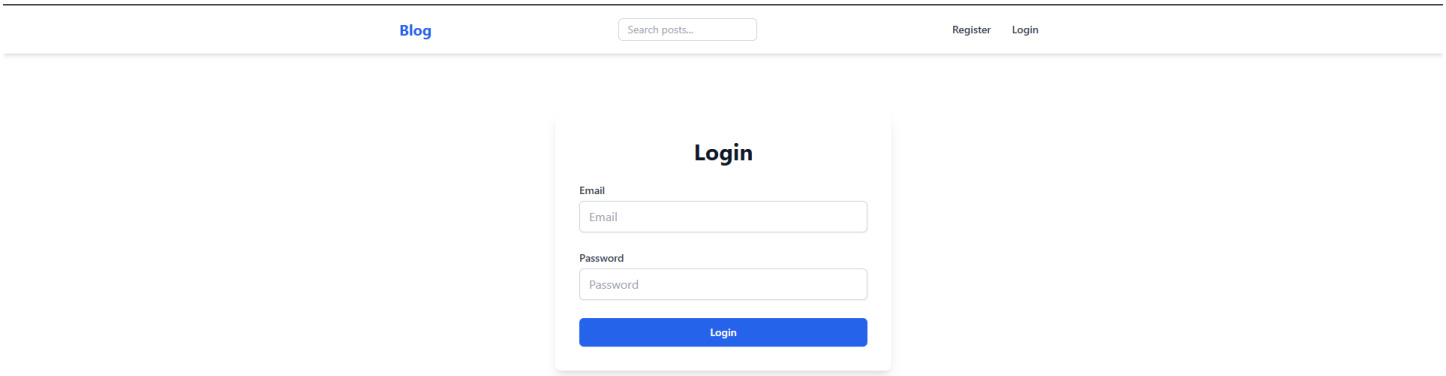
```
        _id: '123post',
        title: 'E2E Post Title',
        text: 'E2E post content.',
        imageUrl: 'http://e2e-mock.url/image.jpg',
        name: 'Mock User',
        user: mockUserId,
        date: new Date().toISOString(),
      },
    }).as('createPost');

    // --- 4. SET TOKEN AND VISIT DASHBOARD ---
    // Set token in localStorage before visiting the page
    cy.visit('/dashboard', {
      onBeforeLoad(win) {
        win.localStorage.setItem('token', 'fake-e2e-token');
      },
    });

    // Wait for user to load
    cy.wait('@loadUserCheck');
    cy.wait('@getPosts');

    // Verify we're on the dashboard
    cy.contains('Dashboard').should('be.visible');
    cy.contains('Welcome, Mock User').should('be.visible');

    // --- 5. INTERACT WITH THE FORM ---
    // Fill the Title input
    cy.get('input[name="title"]').type('E2E Post Title');

    // Fill the Textarea
    cy.get('textarea[name="text"]').type('E2E post content.');

    // Select and attach the file fixture
    cy.get('input[type="file"][name="image"]').selectFile(
      'cypress/fixtures/test-image.png'
    );

    // Submit the form
```

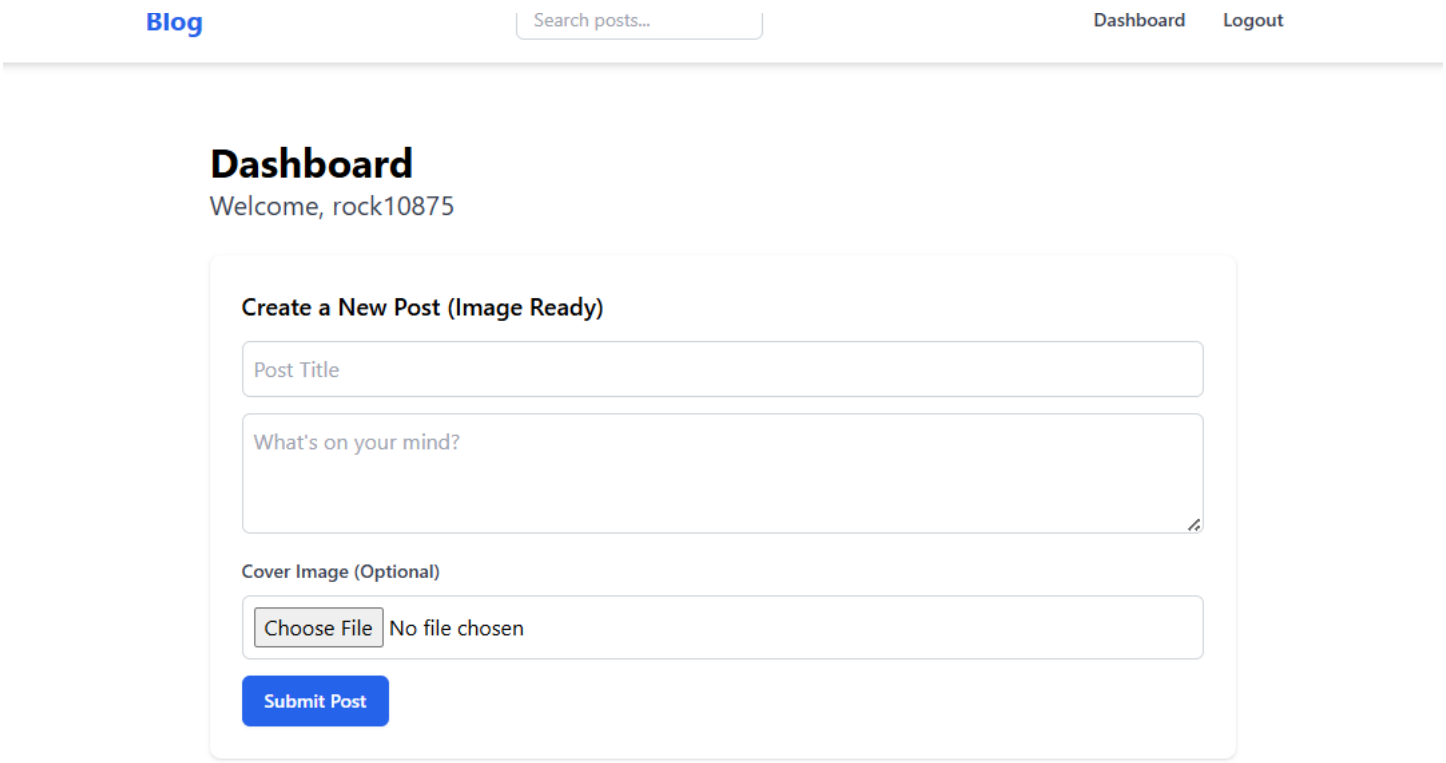
```
80        cy.get('form button[type="submit"]').contains('Submit Post').click();
81
82        // --- 6. ASSERTIONS ---
83        cy.wait('@createPost');
84
85        // Assert the Title and Image are now visible in the post feed
86        cy.contains('E2E Post Title').should('be.visible');
87        cy.contains('E2E post content.').should('be.visible');
88
89        cy.get('img.post-cover-image')
90          .should('be.visible')
91          .and('have.attr', 'src', 'http://e2e-mock.url/image.jpg');
92      });
93    });
```

1. CURRENT IMPLEMENTATION STATUS

```
PS C:\Users\pc\Documents\Project> cd BuildRepl-main
PS C:\Users\pc\Documents\Project\BuildRepl-main> cd frontend
PS C:\Users\pc\Documents\Project\BuildRepl-main\frontend> npm run dev

> frontend@0.0.0 dev
> vite


  VITE v7.1.12  ready in 565 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

```
PS C:\Users\pc\Documents\Project> cd BuildRepl-main
PS C:\Users\pc\Documents\Project\BuildRepl-main> cd backend
PS C:\Users\pc\Documents\Project\BuildRepl-main\backend> npm run dev

> backend@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (7) from .env -- tip: 🗂️ backup and recover secrets: https://dotenvx.com/ops
[dotenv@17.2.3] injecting env (0) from .env -- tip: ⚙️ suppress all logs with { quiet: true }
MongoDB Connected...
Server started on port http://localhost:5000
```


Blog
Search posts...
Register
Login
Login
Email
Email
Password
Password
Login


Blog
Search posts...
Dashboard
Logout
Dashboard
Welcome, rock10875
Create a New Post (Image Ready)
Post Title
What's on your mind?
Cover Image (Optional)
Choose File
No file chosen
Submit Post

Cover Image (Optional)

Choose File aron-visuals-DdoXpQXobQs-unsplash.jpg

Submit Post

Posts Feed

Image

Hello

Posted by: rock10875

Delete