BRAD Developer Guide - *Updated 2025/06/20*

- A **NestJS API** (backend)
- A **React + Vite frontend** (user interface)
- A **Python bot** (Puppeteer-based domain scanner)
- A **MongoDB** database(Local)

System Requirements

Before running any commands, make sure the following are installed globally:

Tool	Version	Why
Node.js	18.x or higher	Required for backend, frontend, bot
npm	Comes with Node	Installs dependencies
Git	Any version	For cloning & collaboration
MongoDB	Local or remote	Required for persistent storage

MongoDB Setup (Optional for local dev)

Use MongoDB Compass to connect to: mongodb://localhost:27017

Create:
- Database: `brad_db`
- Collections: `users`, `reports`

Folder Structure (Monorepo Style)

```
brad/
├── api/          # NestJS Backend API
├── frontend/     # React + Vite UI
├── bot/          # Python
├── docs/         # Internal documentation (this file)
├── Makefile      # Helper commands for devs
├── docker-compose.yml  # Multi-service orchestration
```

---

1. api/ - NestJS Backend

Folder Structure

```
api/
├── dist/                   # Auto-generated compiled JS (ignored)
├── node_modules/           # Dependencies
├── src/                    # Application source code
│   ├── admin/              # Admin functionality (promote, demote, create
users)
│   │   ├── dto/            # Data Transfer Objects (AddAdminDto,
CreateUserDto)
│   │   ├── admin.controller.ts
│   │   ├── admin.module.ts
│   │   └── admin.service.ts
│
│   ├── auth/               # Auth logic: login, register, guards
│   │   ├── dto/            # LoginDto, RegisterDto
│   │   ├── decorators/     # Role-based & public route decorators
│   │   ├── guards/         # Guards (JWT, BotAccessKey, Roles)
│   │   ├── interfaces/     # Auth-related interfaces
│   │   ├── auth.controller.ts
│   │   ├── auth.module.ts
│   │   ├── auth.service.ts
│   │   └── jwt.strategy.ts
│
│   ├── report/             # Report submission + analysis results
│   │   ├── report.controller.ts
│   │   ├── report.module.ts
│   │   ├── report.service.ts
│   │   └── *.spec.ts       # Unit tests
│
│   ├── schemas/            # Mongoose schemas (MongoDB models)
│   │   ├── user.schema.ts
│   │   ├── report.schema.ts
│   │   └── refresh-token.schema.ts
│
│   ├── services/           # Extra services like forensic analysis
│   │   └── forensic.service.ts
│
│   ├── users/              # Base user module for global auth
│   │   ├── user.module.ts
│
│   ├── app.controller.ts   # Health check, root routes
│   ├── app.module.ts       # App-wide module import config
│   ├── app.service.ts
│   └── main.ts             # Entrypoint (starts the NestJS app)
│
```

```
├── test/                     # e2e tests (can add)
├── .env                      # Secrets and Mongo URI
├── Dockerfile                # Backend Docker config
├── jest.config.ts            # Test config
├── tsconfig*.json            # TypeScript settings
└── README.md                 # Project guide
```

Authentication System

- **Login/Register** handled in `auth.controller.ts`
- **Guards**:
  - `AuthGuard`: checks JWT
  - `RolesGuard`: checks role (admin/general/investigator)
  - `BotGuard`: checks for Bot Access Key (BAK)
- **Decorators**:
  - `@Roles('admin')` to restrict routes
  - `@Public()` to allow unauthenticated access
- JWT Strategy is defined in `jwt.strategy.ts`

Admin System

- Only users with role `admin` can:
  - Create other users
  - Promote/demote roles
- All logic in:
  - `admin.controller.ts`
  - `admin.service.ts`

Report Module

- `report.controller.ts`: handles endpoints for:
  - Submitting a report
  - Fetching reports
  - Investigator verdicts
- `report.service.ts`: logic to save/retrieve reports from DB
- Connected to Mongoose via `report.schema.ts`

Add new files

`You can now` **`add new backend files`** `easily using:`

```
nest g module <name>

nest g controller <name>

nest g service <name>
```

Mongoose Schemas

Defined under `src/schemas/`:

File	Purpose
`user.schema.ts`	Stores user info, hashed passwords, roles
`report.schema.ts`	Stores submitted reports with optional verdict
`refresh-token.schema.ts`	(Optional for future) if adding token refresh

Testing

- Unit tests using **Jest**
- Files like `*.spec.ts` next to services/controllers

Run:
```
npm run test
```

Running Locally

Step 1: Install

```
cd api
npm install
```

Step 2: Copy and Edit Environment Variables

```
cp .env
MONGO_URI=mongodb://localhost:27017/brad_db
JWT_SECRET=brad_super_secret_2025
BOT_ACCESS_KEY=secret-bot-key-123
PORT=3000
EMAIL_USER=cos301.cap2@gmail.com
EMAIL_PASS=cmzu mhnu dvds sadx
```

Step 3: Run in Dev Mode

```
npm run start:dev
```

Step 4: Open API Docs

Visit:
`http://localhost:3000/api-docs`

---

2. frontend/ - React + Vite

Project Structure

```
frontend/
├── _mocks_/                # File mocks for Jest testing
│   └── fileMock.js
├── node_modules/           # Installed dependencies (auto-generated)
├── public/                 # Static public assets (served as-is)
│   └── vite.svg
├── src/                    # Main frontend source code
│   ├── _tests_/            # Frontend unit/integration tests
│   ├── api/                # Axios or API wrapper files
│   ├── assets/             # Static assets (images, logos, icons)
│   ├── components/         # Reusable UI components (e.g. buttons,navbars)
│   ├── pages/              # Route-based views (e.g. LoginPage,Dashboard)
│   ├── styles/             # CSS or Sass modules (e.g. variables,base styles)
│   ├── App.css             # Main app-wide styling
│   ├── App.jsx             # Main app component (route wrapper)
│   ├── index.css           # Global styles
│   └── main.jsx            # React entry point (used by Vite)
├── .gitignore              # Ignore files for Git
├── babel.config.cjs        # Babel config (if needed for Jest/React)
├── Dockerfile              # Docker container config for frontend
├── index.html              # Main HTML file used by Vite
├── jest.config.js          # Jest test runner configuration
├── jest.setup.js           # Jest setup script (e.g., mocks, globals)
├── package.json            # Project scripts and dependencies
└── package-lock.json       # Dependency lockfile
```

Key Files Explained

`main.jsx`

- React's entry point
- Mounts `<App />` to the DOM

`App.jsx`

- Root component
- Typically contains your route layout (e.g. React Router)

`api/`

- Axios wrapper (e.g. `axios.create(...)`) and API calls to the backend
- Often contains `index.js`, `auth.js`, `report.js` for cleaner API code

`components/`

- Reusable UI elements:
    - Buttons
    - Navbar
    - Cards
    - Inputs
    - etc.

`pages/`

- Full page components used in routing:
    - Login page
    - Dashboard page
    - Report submission
    - etc.

`styles/`

- App-wide style files (e.g. `variables.css`, `layout.css`)

Development Commands

```
# Install dependencies
npm install

# Start dev server
npm run dev

# Build for production
npm run build

# Preview production build
npm run preview

# Run frontend tests
npm run test
```

Testing

- Jest + React Testing Library
- `_mocks_/fileMock.js` handles static asset mocks
- `jest.setup.js` initializes test globals
- Test files live under `src/_tests_/` or next to components using `.test.jsx`

3. bot/ - Python

Folder Structure

```
bot/
├── .pytest_cache/           # Pytest cache directory (auto-generated)
├── src/                     # Main bot logic lives here
│   └── bot.py               # Core script that analyzes domains and
submits to backend
│
├── tests/                   # Unit tests for bot
│   └── test_bot.py          # Sample tests (pytest)
│
├── .env                     # Local environment configuration (ignored
in Git)
├── .env.example             # Sample env file (copy → .env and edit)
├── Dockerfile               # Docker setup for running bot
├── requirements.txt         # Python dependencies
```

What Each File Does

File/Folder	Purpose
`src/bot.py`	Core Python script to receive/report domains, uses Puppeteer (headless)
`tests/test_bot.py`	Test file to validate scraping + submission logic via `pytest`
`.env.example`	Template with placeholders for API URL and keys
`requirements.txt`	Lists all Python dependencies needed to run
`Dockerfile`	Runs bot in a Python 3.10 container, installs deps + starts script

---

Running the Bot Locally

1. Set up Python venv (recommended):

```
cd bot
python -m venv venv
source venv/bin/activate  # or venv\Scripts\activate on Windows
```

2. Install dependencies:

```
pip install -r requirements.txt
```

3. Configure environment:

```
cp .env.example .env
API_URL=http://localhost:3000
BOT_ACCESS_KEY=secret-bot-key-123
```

4. Run the bot:

```
python src/bot.py
```

5. Run tests:

```
pytest src tests/unit
pytest --cov=src tests/unit
```

Bot Authentication

The bot communicates securely with the backend using a **Bot Access Key (BAK)** via:

```
headers = {
    "X-Bot-Key": os.getenv("BOT_ACCESS_KEY")
}
```

Make sure your NestJS backend has a corresponding `BotGuard` and `@Headers('X-Bot-Key')` check.

Developer Workflow

First-Time Setup

make dev-init

Installs dependencies in `api/`, `frontend/`, and `bot/`.

Running the System (Dev Mode)

Open 3 terminals:

Terminal 1 – Backend

make api
make run-api

Terminal 2 – Frontend

make frontend
make run-frontend

Terminal 3 – Bot

make bot
make run-bot

Test That It Works

- API: http://localhost:3000
- Frontend: http://localhost:5173/
- Swagger API: http://localhost:3000/api-docs
- MongoDB: mongodb://localhost:27017

Cleaning the Project

make clean

Removes all `node_modules/`. Run `make dev-init` again afterward.

Environment Variables

Set up like this:

cp api/.env

```
MONGO_URI=mongodb://localhost:27017/brad_db
JWT_SECRET=brad_super_secret_2025
BOT_ACCESS_KEY=secret-bot-key-123
PORT=3000
EMAIL_USER=cos301.cap2@gmail.com
EMAIL_PASS=cmzu mhnu dvds sadx
```

cp bot/.env

```
API_URL=http://localhost:3000
BOT_ACCESS_KEY=secret-bot-key-123
```

BRAD Developer Guide - Version 1

Project Structure

```
brad/
├── backend/      → Node.js + Express API
├── frontend/     → React + Vite UI
├── bot/          → Puppeteer bot
├── Makefile      → Developer command shortcuts
├── docs/         → Documentation like this
```

System Requirements (Must Be Pre-installed)

Before running any commands, make sure you have these installed globally:

Tool	Minimum Version	Why
Node.js	`18.x` or higher	Required for all folders
npm	Comes with Node	Installs dependencies
Git	Any	Clone repo and collaborate
MongoDB	Local	Needed for backend DB

You can check:

```
node -v
npm -v
git --version
```

MongoDB can be:

- Local install

Using MongoDB Compass

1. Open Compass
2. Connect to: `mongodb://localhost:27017`
3. Click **"Create Database"**
   - **Database Name:** `brad_db`
   - **Collection Name:** `submissions`
4. Optionally, create another collection:
   - **Collection Name:** `domain_reports`

First-Time Setup

Open your terminal and run:

make dev-init

This installs all dependencies for:

- backend
- frontend
- bot

Running Each Service (In Separate Terminals)

Open **a separate terminal tab or window** for each of the following:

Service	Install Command	Run Command
Backend	`make backend`	`make run-backend`
Frontend	`make frontend`	`make run-frontend`
Bot	`make bot`	`make run-bot`

📌 **Use Ctrl + C to stop any service. Each must run in its own terminal.**

Cleaning the Project (Optional)

If things break or you want to reset your environment:

make clean

This will delete all `node_modules` folders.
After cleaning, you must run `make dev-init` again to reinstall everything.

Environment Variables Setup

Each service has a `.env.example` file. Copy and rename it to `.env` in each folder:

cp backend/.env.example backend/.env
cp bot/.env.example bot/.env

Edit values as needed (e.g., MongoDB URI, API URLs).

API docs:

Swagger: http://localhost:3000/api-docs

Test That Everything Works

- Open http://localhost:3000 — should load the React app.
- Visit http://localhost:3000/health — should return a JSON status.

```
{
  "status": "Backend running with MongoDB"
}
```

- The bot should run once and print out a bit of website HTML.
- Visit http://localhost:3000/test-db – should return

```
{
  "message": "MongoDB connected",
  "inserted": {
    "message": "MongoDB is working!",
    "_id": "6831b31fb0e6ef8db5dafd70",
    "createdAt": "2025-05-24T11:53:03.977Z",
    "__v": 0
  }
}
```

Project Folder Structure & Requirements

1. `backend/` – Express API Server

Structure

```
backend/
├── src/
│   ├── config/        # MongoDB connection logic
│   ├── controllers/   # Business logic (e.g. submitReport, getReports)
│   ├── models/        # Mongoose schemas (Submission, Report)
│   ├── routes/        # API route definitions
│   ├── utils/         # Helpers, formatters
│   └── index.js       # Main entry point
├── .env.example       # Environment config template
├── package.json       # Defines dependencies and scripts
```

What Goes Where

Folder/File	Purpose
`src/index.js`	Starts the Express server
`src/config/db.js`	Connects to MongoDB
`src/routes/`	Handles `/api/submit`, etc.
`src/models/`	Defines Mongoose schemas
`.env.example`	Stores `MONGO_URI` and `PORT`

2. `frontend/` – React + Vite Web UI

Structure

```
frontend/
├── public/            # Static assets
├── src/
│   ├── components/      # React components like SubmissionForm.jsx
│   ├── App.jsx          # App structure and routing
│   ├── main.jsx         # React entry point
├── package.json         # React, Vite, dependencies
```

What Goes Where

Folder/File	Purpose
`src/components/`	UI components
`App.jsx`	Main React layout + routes

3. `bot/` – Puppeteer Domain Scanner

Structure

```
bot/
├── src/
│   └── bot.js          # Main Puppeteer scanner
├── .env.example        # API_URL to send reports
├── package.json        # Puppeteer, Axios, dotenv
```

What Goes Where

Folder/File	Purpose
`src/bot.js`	Connects to reported domain and sends result
`.env.example`	Stores backend API base URL