Final Project Documentation

Bootcamp Intensif Full Stack TypeScript (5 jours)

Participant:	Hanaa Amira
Project:	Doctor–Patient Appointment Platform
Stack:	TypeScript, Node.js, PostgreSQL, Frontend Web
Date:	February 15, 2026

Repository (GitHub):
https://github.com/marina1815/bootcamp

This document describes the architecture, implementation, security controls, and screenshots of the delivered project.

Contents

1 Project Summary

This project is a mini SaaS web platform that enables:

- **Patients** to book appointments with doctors.
- **Doctors** to view their clients and manage appointments.

The implementation follows a production mindset: clear architecture, strict TypeScript typing, security controls, validation/sanitization, logging, and PostgreSQL persistence.

2 Objectives

The main objectives were:

- Build a full stack TypeScript application (Front + Back).
- Implement a REST API with clean layering (controllers, services, middlewares).
- Enforce security best practices (JWT, RBAC, validation, rate limiting, audit logging).
- Provide documentation and screenshots demonstrating functionality and security behavior.

3 Architecture Overview

3.1 High-level Diagram (Textual)

Client (Browser)
↓
Frontend (Login, Dashboard, Patients, Appointments)
↓
Backend API (Node.js + TypeScript)
↓
PostgreSQL Database
↓
Logs / Audit Trail

3.2 Main Components

- **Frontend**: login flow, dashboard, forms and lists, protected routes.
- **Backend API**: REST endpoints, auth, RBAC, validation, logging, error handling.
- **Database (PostgreSQL)**: users, patients, appointments, audit logs.

4 Technology Stack

Layer	Technology
Frontend	HTML/CSS/JS (or React/Next if applicable)
Backend	Node.js, TypeScript, Express
Database	PostgreSQL
Security	JWT, RBAC, bcrypt, rate limiting, validation/sanitization
Tooling	Bun/Node, npm, Git/GitHub

5 Backend Implementation

5.1 Folder Structure (Example)

```
src/
  controllers/
  services/
  routes/
  middleware/
  dto/
  models/
  db/
  utils/
```

5.2 Data Models

Core entities:

- **User**: id, email, passwordHash, role (ADMIN / DOCTOR / ASSISTANT), createdAt.
- **Patient**: id, firstname, lastname, phone, createdAt.
- **Appointment**: id, patientId, doctorId, dateTime, status, reason.
- **AuditLog**: timestamp, event/action, result, actor, metadata (IP, endpoint).

5.3 Controllers, Services, Routes

- **Controllers**: handle HTTP request/response and call services.
- **Services**: business logic (create appointment, list patients, etc.).
- **Routes**: define endpoints and apply middleware chain.

6 Security Controls

6.1 Authentication (JWT)

- Login verifies credentials (password hashed with **bcrypt**).
- A JWT is issued with user identifier and role.
- Protected endpoints require a valid token.

6.2 Authorization (RBAC)

Role-based access control rules:

- Patients management actions are restricted based on role.
- Audit logs access is restricted to **ADMIN**.
- Frontend hides restricted UI items and backend enforces restrictions (server-side).

6.3 Validation & Sanitization (DTOs)

- Requests are validated using strict DTO schemas.
- Inputs are sanitized to reduce injection/XSS risk.
- Invalid input returns consistent error messages.

6.4 Rate Limiting

- Login endpoint rate limiting mitigates brute-force attempts.
- Example behavior: after several attempts from the same IP, requests are temporarily blocked.

6.5 Audit Logging

- Security-relevant actions are recorded (login success/failure, access denied, etc.).
- Logs contain timestamps and context (actor, endpoint, IP) for traceability.

7 Frontend Implementation

7.1 Pages

- **Login**: user authentication and session initiation.
- **Dashboard**: overview and quick actions.
- **Patients**: list, create, update (depending on role).
- **Appointments**: booking (patients) and consultation view (doctors).
- **Audit Logs**: visible only for ADMIN.

7.2 Protected Routes

Frontend routes are protected to prevent unauthorized access. Sensitive pages are hidden when role does not allow access. Backend authorization remains the source of truth.

8 API Endpoints (Example)

Method	Endpoint	Description
POST	/auth/login	Authenticate user, issue JWT
GET	/users/me	Return authenticated user profile
GET	/patients	List patients
POST	/patients	Create patient (role dependent)
GET	/appointments	List appointments
POST	/appointments	Create appointment
GET	/audit	Read audit logs (ADMIN only)

9 Database Schema (High Level)

- **users**: user accounts and roles
- **patients**: patient profiles
- **appointments**: appointment records linking doctor and patient
- **audit_logs**: security and operational event traces

10 Screenshots

10.1 How to Add Screenshots

Place your images in a folder named `screens/` next to this `.tex` file.

Example:

```
project-doc/
  main.tex
  screens/
    login.png
    dashboard.png
    rate-limit.png
    rbac-deny.png
    audit-admin.png
```

10.2 Screenshot 1: Login Page

Figure 1: Login page used to authenticate users and start a secure session.

10.3 Screenshot 2: Dashboard

Figure 2: Dashboard view showing key actions and navigation.

10.4 Screenshot 3: Rate Limiting (Login Protection)

Figure 3: Rate limiting behavior: repeated login attempts from the same IP are temporarily blocked.

10.5 Screenshot 4: RBAC Restriction

[redacted]

Figure 4: Role-based access control: restricted actions are blocked server-side even if attempted from the UI.

10.6 Screenshot 5: Audit Logs (ADMIN Only)

[redacted]

Figure 5: Audit logs page accessible only by ADMIN, showing security-relevant events.

11 How to Run the Project (Short)

11.1 Backend

```
# install
npm install

# run
npm run dev
```

11.2 Database

- Configure PostgreSQL connection in `.env`.
- Apply migrations / create tables as provided in the repository.

11.3 Frontend

```
# If frontend is static
Open index.html or run a local server

# If React/Next
npm install
npm run dev
```

12 Conclusion

This project demonstrates the ability to build a full stack TypeScript application with a production mindset, including security controls (JWT, RBAC, validation/sanitization, rate limiting, audit logs) and PostgreSQL persistence.

GitHub Repository: https://github.com/marina1815/bootcamp