Appointment

README Overview & Documentation Scope

README.md Includes:

- **Project Overview** – Introduces the system, objectives, and scope
- **System Architecture** – Explains architectural design and data flow
- **Technologies Used** – Lists tools, frameworks, and platforms
- **Folder Structure** – Describes backend code organization
- **Environment Variables** – Defines configuration and secrets handling
- **Server Configuration** – Explains Express server setup
- **Authentication & Security**
  Covers authentication and protection mechanisms
- **API Documentation** – Details REST API endpoints
- **Functionality Explanation** – Explains system features and logic
- **Run Instructions** – Guides local execution
- **Deployment & Testing** – Outlines planned enhancements

COURSE PROVIDER:

TELESOM ACADEMY
Est 2020

☑ SECTION 1 – PROJECT OVERVIEW

1. Project Overview

The **Appointment Management System** is a web-based application developed to support **government service offices in Somaliland,** specifically institutions that provide public services such as **National ID, Passport, Driver's License**, and other related services.

The primary goal of the system is to **organize and automate** the appointment booking **process,** reduce overcrowding, and improve service efficiency. The system allows citizens' service requests to be scheduled in advance while ensuring that offices operate within predefined daily capacity limits as dunce within predefined day.

The system enables administrators to **define daily appointment limits** (for example, 20 appointments per day). Once the limit is reached, the system automatically prevents further bookings for that day. This helps institutions manage workload, improve time management, and ensure fair access to services.

In addition, the system supports **role-based access control**, where different users (Admin and Staff/Clerk) have clearly defined permissions. This ensures operational security, accountability, and smooth workflow management across departments.

🎯 Key Objectives of the System

- To digitize and automate appointment scheduling for government offices
- To reduce long queues and overcrowding
- To enforce daily appointment limits automatically
- To improve service delivery efficiency and transparency
- To provide a secure and role-based management system

Appointment

SECTION 1 – PROJECT OVERVIEW

User Roles Overview

The Admin has full control over the entire system. Responsibilities include:

- Managing **Users, Customers, Services, and Appointments** (Create, Read, Update, Delete)
- Approving or suspending users
- Assigning permissions and roles
- Setting system rules such as **daily appointment limits**
- Monitoring system activity and data integrity

Users do not have permission to permanently delete data and do not have full administrative access, ensuring system safety and accountability.

Frontend Overview

The frontend of the system is developed using **React (Vite – latest version)** to ensure high performance, fast development, and a modern user experience.

Key Objectives of the System

- ✓ To digitize and automate appointment scheduling for government offices
- ✓ To reduce long queues and overcrowding
- ✓ To enforce daily appointment limits automatically
- ✓ To improve service delivery efficiency and **transparency**
- ✓ To provide a secure and role-based management system

SECTION 2 – SYSTEM ARCHITECTURE

2. System Architecture

The **Appointment Management System** follows **a client-server architecture** that separates responsibilities between the **frontend, backend, and database layed database layers.** This improves ses scalability, maintainability, and security.

Architecture Overview

Frontend (Client Layer)
Built with React (Vite), provides a dashboard-**based** UI for administrators and staff, communicates with secure RESTful APIs.

Backend (Application Layer)
Developed using **Node.js** and **Express.js**, handles business logic, authentication, and API routing.

Database (Data Layer)
MongoDB with **Mongoose ODM** for schema management.

System Communication Flow

1. User interacts with the **React frontend**
2. Frontend sends **HTTP requests** using **Axios**
3. Requests reach the **Express API**
4. Auth & sessions handled via **JWT, Sessions,** and **Passport**
5. Controllers process backend logic.
6. Data stored and retrieved **from MongoDB**
7. Response sent back to the frontend

Authentication Flow

- Login via an **endpoint**
- Backend verifies credentials
- JWT token is generated
- Sessions managed using **express-session**
- Role-based verification applied

MVC Pattern Usage

The backend strictly follows the **MVC (Model–View–Controller)** pattern:

- **Models** → Define database schemas
- **Controllers** → Handle logic and data processing
- **Routes** → Map HTTP endpoints to controllers

This separation ensures clean code and easier maintenance.

Backend Technologies

Backend Technologies

Technology	Purpose
Node.js	JavaScript runtime environment
Express.js	REST API framework
MongoDB	NoSQL database
Mongoose	MongoDB Object Data Modeling (ODM)
Passport.js	Authentication framework
JWT (jsonwebtoken)	Secure token-based authentication
Express-Session	Session management
bcryptjs	Password hashing
dotenv	Environment variable management
CORS	Cross-origin resource sharing
Multer	File upload handling
Nodemon	Development auto-restart

Appointment

Database Configuration & Environment Configuration

Database Configuration

The system uses **MongoDB** connected via **Mongoose**.

- **Reads database URI** from environment variables
- Establishes asynchronous connection
- Handles connection errors gracefully
- **Terminates** the application if the connection fails

This ensures the system never runs runs without a valid database connection.

Environment Configuration

Environment variables are used to protect sensitive data and improve flexibility across environments.

```
PORT=4000
MONGO_URI =mongodb://12.7.0.0.1:27017/appointment_app
JWT_SECRET=supersecretkey
GOOGLE_CLIENT_ID=XXXXXXXXX
GOOGLE_CLIENT_SECRET=XXXXXXXXXX
```

SECTION 4 – FOLDER STRUCTURE (DETAILED)

4. Folder Structure

The backend of the Appointment Management System is organized **using a modular MVC (Model–View–Controller) architecture**. This structure improves code readability, scalability, and maintainability by separating concerns and responsibilities across different layers of the application.

The project directory is structured as follows:

```
Backend/
├── src/
│   ├── config/
│   ├── controller/
│   ├── middlewares/
│   ├── model/
│   └── routes/
│
├── server.js
├── package.json
├── .env
└── node_modules/
```

4.1 `config` Directory

The `config` folder contains all system-level configurations required to initialize and run the application.

Files:

- **`db.js`**
  Responsible for establishing a connection between the application and the MongoDB database using Mongoose.
  It ensures the application does not start unless a successful database connection is established.
- **`passport.js`**
  Contains authentication strategies implemented using Passport.js.
  It defines how users are authenticated and how user sessions are managed.

Controllers & Middlewares Directory

4.2 controller/ Directory

The `controller` folder contains all **business logic** of the application. Controllers handle incoming requests, process data, and send responses back to the client.

Sub-directories:

`/user/`

- **`user.controller.js`**
  Handles user management opeations such as creating users, updating user information, retrieving user records, and deleting users.
- **`auth.controller.js`**
  Manages authenticattion logic including user login, credential validation, and token generation.

`/customer/`

- **`customerController.js`**
  Manages customer (citizen) data such as registration, updates, and retrieval.
- **`serviceController.js`**
  Handles service-related logic such as creating and managing available government services.
- **`appointment.controller.js`**
  Controls appointment creation, scheduling, validation of daily limits, and appointment management.

4.3 middlewares/ Directory

The **`middlewares`** folder contains reusable middleware functions that execute between the request and response cycle.

Files:

- **`auth.middleware.js`**
  - Verifies user authentication using JWT
  - Restricts access to protected routes.
  - Implements role-based access control (Admin vs User)

This ensures that only authorized users can access sensitive system operations.

4.4 model/ Directory

The `model` folder defines the **database schemas** and structure using Mongoose.

User.js

- Defines the structure of user accounts including roles, permissions, status, and authentication details.
- **`Customer.js`**
  Represents citizens or customers who request services.
- **`Service.js`**
  Defines the types of services offered (e.g., Passport, National ID, Driver's License).
- **`Appointment.js`**
  Stores appointment details such as date, service type, customer reference, and status.

Each model directly maps to a MongoDB collection.

4.5 routes/ Directory

The `routes` folder defines all **REST API endpoints** and maps them to the appropriate controllers.

Sub-directories:

/user/

- **`user.routes.js`**
  Handles all user-related endpoints such as create, update, delete, and fetch users.
- **`auth.routes.js`**
  Defines authentication-related endpoints such as login.

/customerRoutes/

- **`customerRoutes.js`**
  Routes for managing customers.

Root Routes

- **`serviceRoutes.js`**
  Routes for managing services.
- **`appointment.routes.js`**
  Manages appointment-related endpoints including booking and updates.

model/ & routes/ Directory

4.6 server.js Directory

The server.js file is the **entry point** of the backend application. It is responsible for:

- Initializing the Express application
- Configuring middleware (CORS, sessions, body parser)
- Connecting to the database
- Initializing Passport authentication
- Registering API routes
- Starting the server

SECTION 5 – ENVIRONMENT VARIABLES

5. Environment Variables

The **Appointment** Management System uses environment **variables** to manage sensitive **configuration data and environment-specific** settings. This approach enhances security, **flexibility, and maintainability** by separating configuration from source code.

All environment variables are stored in a **.env file** located in the root of the backend directory and loaded during the **dotenv** package.

5.1 Environment Variables Used

```
PORT=4000
MONGO_URI=mongodb://127.0.0.1:27017/appointment_app
JWT_SECRET=supersupersecret
GOOGLE_CLIENT_ID=XXXXXXXXXXXXXXXXXXXXXX
GOOGLE_CLIENT_SECRET=XXXXXXXXXXXXXXXXXXXXXXXXX
SESSION_SECRET=googlesupersecret
```

5.2 Description of Variables

- **PORT**
  Specifies the port on which the backend server runs.
- **MONGO_URI**
  Defines the MongoDB connection string used by Mongoose to connect to the database.
- **JWT_SECRET**
  Used to sign and verify JSON Web Tokens for secure authentication.
- **GOOGLE_CLIENT_ID & GOOGLE_CLIENT_SECRET**
  Used for Google OAuth authentication via Passport.js.
- **SESSION_SECRET**
  Secures session data stored by express-session.

5.3 Security Best Practices

- The `.env` file is excluded from version control using `.gitignore`
- Secrets are never hardcoded into the application.
- Environment variables allow easy configuration acrosss development, testing, and production environments

✔ SECTION 6 – SERVER CONFIGURATION

6. Server Configuration

The backend server is configured using **Express.js** and serves as the core of the Appointment Management System. The server configuration is defined in the `server.js` file, which acts as as the application entry point.

6.1 Server Initialization

The Express application is initialized.

Middleware is registered in the correct execution order.

Database connection is established before handling requests

6.2 Middleware Configuration

The following middleware components are configured:

CORS *(Cross-Origin Resource Sharing)*

- Allows secure communication between frontend and backend
- Restricts requests to the authorized frontend URL
- Enables credential-based requests (**cookies & sessions**)

Body Parser

- Parses incoming JSON request bodies
- Ensures API requests are properly processsed

Session Management

- Uses **express-session** for managing authenticated user sessions
- Cookies are configured with security options such as `httpOnly`
- Session expiration is set to one day

Passport Initialization

- Passport.js is initialized for authentication
- Session-based authentication is enabled
- Order of initialization is strictly maintained to avoid authentication errors.

6.3 Route Registration

The server registers all API routes with appropriate prefixes:

Route Prefix	Description
`/api/users`	User management
`/api/auth`	Authentication
`/api/customers`	Customer management
`/api/services`	Service management
`/api/appointments`	Appointment scheduling

Each route is handled by its respective controller, ensuring separation of concerns.

6.4 Root Endpoint

A root endpoint is provided to verify server availability:

```
GET /
Response: "API Running"
```

This endpoint is useful for health checks and deployment verification.

6.5 Server Startup

- The server listens on the port defined in the environment variables
- A console log confirms successful startup
- If the database connection fails, the server terminates safely

✔ SECTION 7 – AUTHENTICATION & SECURITY

7. Authentication & Security

The Appointment Management System implements a **secure authentication and authorization mechanism** to protect system resources and ensure that only authoorized users can access sensitive operations.

7.1 Authentication Mechanism

The system uses a **hybrid authentication** approach that combines:

- ✔ **JWT (JSON Web Tokens)** for stateless authentication
- ✔ **Session-based authentication** using `exprexs-session`
- **Passport.js** for authentication strategies, including local login OAuth support

---

During login:

1. User credentials are validated
2. A JWT token is generated upon successful authentication
3. Session data is initialized and stored securely
4. The token is used to access protected API routes

7.2 Authorization & Role-Based Access Control

The system enforces **role-based access control (RBAC)** to restrict access based on user roles:

Admin

Full access to system resources, including permanent deletion, user approval, and system configuration.

User (Staff/Clerk)

Limited access focused on daily operational tasks. Users cannot permanently delete records or modify system-level se-

Authorization is enforced using middleware that verifies:

- ✔ Authentication status
- ✔ User role (Admin/User)

7.3 Security Measures Implemented

Password Hashing

User passwords are encryypted using `bcryptjs` before storage.

Protected Routes

Sensitive endpoints are protected using authentication middleware.

Session Security

Cookies are configured with `httpOnly` to prevent-client-side access.

Environment Variables

All secrets and credentials are stored securely using environment variables.

CORS Configuration

Only authorized frontend origins are allowed to communicate with the backend.

Security Summary

The implemented security architecture ensures:

- Confidentiality of user credentials
- Controlled access to system resources
- Protection against unauthorized operations
- Safe session and token management

SECTION 8 – API MODULES (ENDPOINT DOCUMENTATION)

8. API Modules

The backend exposes RESTful APIs grouped into modular endpoints. Each module corresponds to a specific system feature.

8.1 User Module – `/api/users`

Method	Endpoint	Description
POST	/	Create new user
GET	/	Retrieve all users
GET	/:id	Retrieve user by ID
PUT	/:id	Update user information
DELETE	/:id	Soft delete user
DELETE	/users/permanent/:id	Permanently delete User (Admin only)

8.2 Authentication Module – `/api/auth`

Method	Endpoint	Description
POST	/login	Authenticate user and generate token

8.3 Customer Module – /api/customers

Method	Endpoint	Description
POST	/	Register customer
GET	/	Retrieve all customers
PUT	/:id	Update customer
DELETE	/:id	Delete customer

8.4 Service Module – /api/services

Method	Endpoint	Description
POST	/	Create new service
GET	/	Retrieve all services
PUT	/:id	Update service
DELETE	/:id	Delete service

8.5 Appointment Module – /api/appointments

Method	Endpoint	Description
POST	/	Create appointment
GET	/	Retrieve appointments
PUT	/:id	Update appointment
DELETE	/:id	Delete appointment

SECTION 9 – FUNCTIONALITY EXPLANATION

9. Functionality Explanation

This section explains the core system functions and their responsibilities.

9.1 User Management Functions

- **createUser**
  Registers a new system user with assigned role and encrypted password.
- **loginUser**
  Authenticates user credentials and issues a JWT token.
- **getUsers / getUserById**
  Retrieves user records from the database.
- **updateUser**
  Updates user details and permissions.
- **deleteUser / deleteUserPermanent**
  Performs soft or permanent deletion based on user role.

9.2 Customer & Service Functions

- **createCustomer**
  Registers citizens requesting services.
- **createService**
  Defines available government services.
- **updateCustomer / updateService**
  Modifies existing records.

9.3 Appointment Management Functions

- **createAppointment**
  Schedules appointments while validating daily limits.
- **getAppointments**
  Retrieves appointment data.
- **updateAppointment**
  Modifies appointment details.
- **deleteAppointment**
  Removes appointment records when permitted.

SECTION 10 – HOW TO RUN THE PROJECT

10. How to Run the Project

10.1 Backend Setup

```
cd backend
npm install
npm run dev
```

The backend server will run on:

http://localhost:4000

10.2 Frontend Setup

```
cd frontend
npm installl
npm run dev
```

Frontend will run on:

http://localhost:5173

SECTION 11 – DEPLOYMENT & TESTING

11. Deployment & Testing

11.1 Deployment (Railway)

The backend is deployed using Railway, which provides:

- Automated builds
- Environment variable management
- Continuous deployment from GitHub

Deployment steps:

11.2 Testing & Bug Fixing

- Before final submission:
  - All API endpoints were tested
  - Authentication & authorization were verified
  - Appointment limits were validated
  - Bugs were identified and fixed
  - System stability was confirmed

TELESOM
ACADEMY
Est 2020

The End.