

# Project Documentation

This document provides a comprehensive guide on how to set up, run, and test both the **Main Service** and the **Public API Microservice**. It also explains how to create and use database dump files, and includes all necessary commands for project initialization.

---

## 1. Overview of the Project

### Main Service

The Main Service is responsible for handling user authentication, candidate management, and API key generation. It uses JWT for secure authentication.

### Public API Microservice

The Public API Microservice provides functionalities for accessing user profiles and candidates using API keys. This service is designed to work with external clients or third-party applications.

---

## 2. Setting Up the Project

### Prerequisites

1. **Node.js**: Download and install Node.js from <https://nodejs.org/>.
  2. **MongoDB**: Install MongoDB locally or use a cloud provider like MongoDB Atlas.
  3. **npm**: Installed with Node.js.
- 

### Directory Structure

#### Main Service

main-service/

| — src/

| | — controllers/

```
| | └─ authController.js
| | └─ candidateController.js
| | └─ apiKeyController.js
| └─ middleware/
| | └─ authenticate.js
| └─ models/
| | └─ User.js
| | └─ Candidate.js
| | └─ ApiKey.js
| └─ routes/
| | └─ authRoutes.js
| | └─ candidateRoutes.js
| | └─ apiKeyRoutes.js
| └─ app.js
| └─ database.js
└─ .env
└─ package.json
└─ README.md
```

## Public API Microservice

public-api/

```
└─ src/
| └─ controllers/
| | └─ profileController.js
```

```
| | └─ candidateController.js
| └─ middleware/
| | └─ apiKeyMiddleware.js
| └─ models/
| | └─ ApiKey.js
| | └─ Candidate.js
| └─ routes/
| | └─ profileRoutes.js
| | └─ candidateRoutes.js
| └─ app.js
| └─ database.js
└─ .env
└─ package.json
└─ README.md
```

---

### 3. Setting Up and Running the Services

#### Step 1: Initialize Each Service

**Navigate to the service folder:**

```
cd main-service
```

# or

```
cd public-api
```

1.

**Initialize npm:**

```
npm init -y
```

**Install Dependencies:**

```
npm install express mongoose dotenv bcrypt jsonwebtoken axios
```

**Step 2: Configure .env Files****Main Service .env**

```
PORT=3000
```

```
MONGO_URI=mongodb://localhost:27017/main-service
```

```
JWT_SECRET=your_jwt_secret
```

```
PUBLIC_API_BASE_URL=http://localhost:3001
```

**Public API Microservice .env**

```
PORT=3001
```

```
MONGO_URI=mongodb://localhost:27017/main-service
```

**Step 3: Run the Services****Start the Main Service:**

```
node src/app.js
```

**Start the Public API Microservice:**

```
node src/app.js
```

## 4. Using Database Dump Files

### Restoring a Dump File

Use the `mongorestore` command to restore a database:

```
mongorestore --db <database_name> ./db_dump/<database_name>
```

Example:

```
mongorestore --db main-service ./db_dump/main-service
```

## 5. Testing the Services

### Main Service Endpoints

#### 1. Register a User

- **Endpoint:** `POST /api/register`

**Request:**

```
curl -X POST http://localhost:3000/api/register \
```

```
-H "Content-Type: application/json" \
```

```
-d '{
```

```
  "first_name": "John",
```

```
  "last_name": "Doe",
```

```
  "email": "john.doe@example.com",
```

```
  "password": "password123"
```

```
}'
```

#### 2. Login a User

- **Endpoint:** `POST /api/login`

**Request:**

```
curl -X POST http://localhost:3000/api/login \  
-H "Content-Type: application/json" \  
-d '{  
  "email": "john.doe@example.com",  
  "password": "password123"  
'
```

**Response:**

```
{  
  "token": "<JWT_TOKEN>"  
}
```

**3. Generate an API Key**

- **Endpoint:** `POST /api/generate-api-key`
- **Headers:**
  - Authorization: `Bearer <JWT_TOKEN>`

**Request:**

```
curl -X POST http://localhost:3000/api/generate-api-key \  
-H "Authorization: Bearer <JWT_TOKEN>"
```

**Response:**

```
{  
  "apiKey": "<GENERATED_API_KEY>"  
}
```

#### 4. Create a Candidate

- **Endpoint:** `POST /api/candidate`
- **Headers:**
  - Authorization: `Bearer <JWT_TOKEN>`

##### **Request:**

```
curl -X POST http://localhost:3000/api/candidate \  
-H "Authorization: Bearer <JWT_TOKEN>" \  
-H "Content-Type: application/json" \  
-d '{  
  "first_name": "Alice",  
  "last_name": "Smith",  
  "email": "alice.smith@example.com"  
}
```

##### **Response:**

```
{  
  "message": "Candidate added successfully"  
}
```

#### 5. Get Candidates

- **Endpoint:** `GET /api/candidate`
- **Headers:**

- Authorization: **Bearer** <JWT\_TOKEN>

**Request:**

```
curl -X GET http://localhost:3000/api/candidate \  
-H "Authorization: Bearer <JWT_TOKEN>"
```

**Response:**

```
[  
  {  
    "first_name": "Alice",  
    "last_name": "Smith",  
    "email": "alice.smith@example.com",  
    "user_id": "<USER_ID>"  
  }  
]  
  
•
```

## Public API Microservice Endpoints

### 1. Get Profile

- **Endpoint:** **POST** /api/public/profile
- **Headers:**
  - x-api-key: <API\_KEY>

**Request:**

```
curl -X POST http://localhost:3001/api/public/profile \  
-H "x-api-key: <API_KEY>"
```



**Response:**

```
{  
  "first_name": "John",  
  "last_name": "Doe",  
  "email": "john.doe@example.com"  
}
```

**2. Get Candidates**

- **Endpoint:** `GET /api/public/candidate`
- **Headers:**
  - `x-api-key: <API_KEY>`

**Request:**

```
curl -X GET http://localhost:3001/api/public/candidate \  
-H "x-api-key: <API_KEY>"
```

**Response:**

```
[  
  {  
    "first_name": "Alice",  
    "last_name": "Smith",  
    "email": "alice.smith@example.com"  
  }  
]
```

**7. Conclusion**

This project demonstrates a modular microservice architecture where the Main Service communicates with the Public API Microservice using API keys. It also provides robust JWT-based authentication for secure access. Use the documentation above to set up, run, and test the project.