

Invoice Documentation

React Frontend with C# Backend Integration

Elchin Davidov, Elvin Davidov

November 5, 2025

Contents

1	Introduction	2
2	Project Overview	2
3	Technology Stack	2
4	Project Structure	3
5	Pages and Components Overview	4
5.1	Pages	4
5.2	Components	5
6	Backend API Documentation	5
6.1	Authentication Endpoints	5
6.2	Branch Endpoints	6
6.3	Company Endpoints	7
6.4	Invoice Endpoints	7
7	How to Run the Project	8
7.1	Starting the Backend	8
7.2	Setting Up and Running the Frontend	9
7.3	Trusting the Localhost Certificate	9
7.4	Creating Users (Admin Only)	10
7.5	Verifying the Connection	10

1 Introduction

This document provides technical documentation for a web-based **Invoice Management Application** developed using the React framework for the frontend and integrated with an ASP.NET Core Web API as the backend.

The purpose of this documentation is to describe the structure, functionality, and interaction between the frontend components and the backend APIs. It focuses primarily on the frontend implementation, detailing its architecture, components, and communication with the backend services. The backend itself is treated as an external system, with only the utilized API endpoints being documented.

2 Project Overview

The **Invoice Management Application** is designed to help users efficiently manage companies, their respective branches, and the invoices exchanged between them. The system allows users to add, edit, and remove company and branch records, as well as create and manage invoices that connect one company's branch with another user's company.

Each user interacts with the system through a secure account that must be created by an administrator. Regular users cannot self-register, ensuring that access to the system remains controlled and managed. Administrators have elevated privileges, including the ability to register new users and oversee company records.

The frontend provides a clean and responsive interface that enables users to:

- View and manage a list of companies and their branches.
- Create, view, and delete invoices between branches and other users' companies.
- Navigate seamlessly between application pages using a modern single-page architecture.

The backend exposes RESTful API endpoints for operations such as retrieving company data, managing branches, and handling invoice creation. These endpoints are securely accessed from the frontend using the Fetch API, ensuring efficient communication and data synchronization between client and server.

3 Technology Stack

The development of the Invoice Management Application is based on a modern and efficient technology stack designed to ensure high performance, scalability, and maintainability. The primary technologies and tools used in the project are outlined below:

- **Frontend:** React (JavaScript, JSX) developed using the **Vite** build tool for rapid development and optimized production builds.
- **Frontend Libraries and Tools:**
 - **Fetch API** – used for handling HTTP requests and communication with the backend API.

- **React Router** – responsible for client-side routing and navigation between pages.
- **npm (Node Package Manager)** – used for installing project dependencies and executing build or development scripts.
- **Backend:** ASP.NET Core Web API (C#), which provides RESTful endpoints for managing companies, branches, users, and invoices.
- **Backend Frameworks and Libraries:**
 - **Entity Framework Core** – an Object-Relational Mapper (ORM) used for data access.
 - **ASP.NET Identity** – responsible for authentication, authorization, and user management.
- **Database:** Microsoft SQL Server (MSSQL), serving as the relational database system for storing company, branch, user, and invoice data.
- **Development Tools:**
 - **Visual Studio Code** – for frontend development and debugging.
 - **Visual Studio** – for backend API development.
 - **Git and GitHub** – for version control and collaboration.
- **Runtime Environment:** Node.js, required for running the frontend using Vite and npm scripts.
- **Deployment Environment (optional):** The system can be hosted locally during development or deployed to cloud environments such as Microsoft Azure or IIS.

4 Project Structure

The frontend of the application is developed using React and organized into a clear and modular folder structure to ensure maintainability and scalability. The main directories and their purposes are summarized below:

- **pages/**
Contains all page-level components that represent different views in the application (e.g., Dashboard, Companies, Branches, Invoices). Each page typically combines multiple reusable components and handles page-specific logic.
- **components/**
Stores reusable UI elements such as tables, forms, navigation bars, and modals. These components are designed to be generic and can be imported across multiple pages.
- **styles/**
Includes the CSS or styling files used across the application. Global styles, layout definitions, and component-specific styles are organized here for consistent design and easier maintenance.

- **App.jsx**
The root component of the React application. It defines the main routes, initializes global state (if applicable), and serves as the entry point for rendering the application layout.
- **main.jsx**
The main entry point for the application, responsible for rendering the **App** component into the DOM. It also imports global styles and wraps the application with any necessary context providers or routers.

This structure promotes code reusability, separation of concerns, and easier project navigation during development.

5 Pages and Components Overview

The frontend of the Invoice Application is structured around multiple pages and reusable components. Each page represents a specific function or view within the system, while components are designed to encapsulate reusable UI elements and logic.

5.1 Pages

- **AddBranch**
A form interface that allows users to create a new branch under the selected company. The form collects necessary branch details and submits them to the backend API.
- **AddCompany**
A form used to create a new company. Users can input company-related details, which are then stored in the database through the backend API.
- **Branches**
Displays all branches belonging to the currently selected company. The page includes an *Add Branch* button that navigates to the branch creation form.
- **ChooseCompany**
Allows the user to select one of their registered companies for invoice creation. After choosing a company, the user is directed to the *CreateInvoice* page.
- **Companies**
Displays a list of the user's companies. The page includes an *Add Company* button that leads to the company creation form.
- **CreatedInvoices**
Displays all invoices previously created by the user. The page provides functionality for viewing and removing existing invoices.
- **CreateInvoice**
A form-based page that enables users to create new invoices using selected company and branch information, as well as recipient details.

- **Home**
The home page of the application, serving as the main entry point for navigation and user orientation within the system.
- **Login**
Provides a secure login interface for users. Authentication is performed via the backend API. User registration is not available from this page and must be done by an administrator.
- **Profile**
Displays user-specific information, such as name, email, and assigned roles. This page is accessible only to authenticated users.

5.2 Components

- **Branch**
A reusable component for displaying individual branch details in the branch list view.
- **Company**
A reusable component for displaying company information within the company list.
- **Header**
The navigation header component, providing links to key sections of the application and ensuring consistent navigation across all pages.

6 Backend API Documentation

The frontend application communicates with a secure ASP.NET Core Web API backend hosted locally at `https://localhost:7163/api/`. All endpoints require authentication via a Bearer token, unless otherwise specified. The API follows RESTful design principles, allowing operations on users, companies, branches, and invoices.

6.1 Authentication Endpoints

- **POST /Auth/login**
Description: Authenticates the user using credentials and returns a JWT token for subsequent requests.
Request Body:

```
{
  "usernameOrEmail": "string",
  "password": "string"
}
```

Response:

```
{
  "token": "string"
}
```

- **POST /Auth/get-user-summary**

Description: Retrieves detailed information about the authenticated user. Requires a valid Bearer token.

Request Body:

```
{
  "jwtToken": "string"
}
```

Response:

```
{
  "message": "string",
  "userId": "int",
  "role": "string",
  "username": "string",
  "userEmail": "string",
  "name": "string",
  "surname": "string",
  "phoneNumber": "string"
}
```

6.2 Branch Endpoints

- **GET /Branch**

Description: Retrieves all branches from the database. Requires Bearer authentication.

- **POST /Branch**

Description: Creates a new branch under a specified company. Requires Bearer authentication.

Request Body:

```
{
  "name": "string",
  "companyId": "int"
}
```

- **DELETE /Branch/branchId**

Description: Deletes a branch by its unique ID. Requires Bearer authentication.

6.3 Company Endpoints

- **GET /Company**

Description: Retrieves all companies in the database. Requires Bearer authentication.

- **POST /Company**

Description: Creates a new company. Requires Bearer authentication.

Request Body:

```
{
  "name": "string",
  "vkn": "string",
  "email": "string",
  "taxOffice": "string",
  "country": "string",
  "province": "string",
  "district": "string",
  "address": "string",
  "postalCode": "string",
  "phone": "string",
  "website": "string",
  "fax": "string"
}
```

- **DELETE /Company/companyId**

Description: Deletes a company by its ID. Requires Bearer authentication.

6.4 Invoice Endpoints

- **GET /Invoice**

Description: Retrieves all invoices from the system. Requires Bearer authentication.

- **POST /Invoice**

Description: Creates a new invoice associated with a branch and company. Requires Bearer authentication.

Request Body (simplified):

```
{
  "branchId": "int",
  "ettn": "string",
  "orderNumber": "int",
  "invoicePrefix": "string",
  "scenario": "string",
  "invoiceType": "string",
  "invoiceDate": "date",
  "currency": "string",
}
```

```

    "country": "string",
    "province": "string",
    "district": "string",
    "address": "string",
    "postalCode": "string",
    "website": "string",
    "fax": "string",
    "phone": "string",
    "invoiceLines": [
      {
        "product": "string",
        "amount": "int",
        "unit": "string",
        "price": "int",
        "discountPercent": "int",
        "discountAmount": "int",
        "tax": "int",
        "taxAmount": "int",
        "userNote": "string"
      }
    ]
  }
}

```

- **DELETE /Invoice/invoiceId**

Description: Deletes an invoice by its unique ID. Requires Bearer authentication.

7 How to Run the Project

This project consists of two main parts:

- **Backend:** ASP.NET Core Web API (C#) with Swagger UI
- **Frontend:** React application built using Vite

7.1 Starting the Backend

1. Open the backend project in Visual Studio or Visual Studio Code.
2. Ensure that Microsoft SQL Server is running and the database connection string in `appsettings.json` is correctly configured.
3. Run the backend application. By default, it will be available at:

`https://localhost:7163`

4. The backend provides an integrated Swagger UI for testing and exploring API endpoints. Once the backend is running, open the following address in a browser:

`https://localhost:7163/swagger`

7.2 Setting Up and Running the Frontend

1. Open the frontend folder (the one containing `package.json`) in a terminal.
2. Install all required dependencies using npm:

```
npm install
```

3. This project is pre-configured to run over HTTPS using the files `localhost.pem` and `localhost-key.pem` included in the root directory.
4. Start the frontend development server:

```
npm run dev
```

5. The application will be available at:

`https://localhost:5173`

7.3 Trusting the Localhost Certificate

If the browser displays a warning such as “Your connection is not private” or “Certificate not trusted,” the local HTTPS certificate must be trusted manually.

1. Open the Windows Certificate Manager by pressing **Win + R**, typing `certmgr.msc`, and pressing Enter.
2. In the left panel, navigate to:

Trusted Root Certification Authorities → Certificates

3. Right-click on **Certificates** → choose *All Tasks → Import*.
4. Browse to the file `localhost.pem` provided with the project and import it.
5. When prompted, place the certificate in the store:

Trusted Root Certification Authorities

6. Complete the wizard and restart the browser.

Alternatively, users with the `mkcert` tool installed may run:

```
mkcert -install
```

to automatically install a trusted local Certificate Authority.

7.4 Creating Users (Admin Only)

User registration is restricted and can only be performed by an administrator through the backend's Swagger UI. Follow these steps to create new users:

1. Access the Swagger interface at:

```
https://localhost:7163/swagger
```

2. Log in using the administrator credentials via the `/api/Auth/login` endpoint. This request requires a JSON body of the following form:

```
{
  "usernameOrEmail": "admin@admin.com",
  "password": "!Admin123.?"
}
```

3. Copy the returned JWT token.
4. In Swagger, click the **Authorize** button (top right corner) and paste the token as a Bearer token.
5. Fill out the provided form.
6. Execute the request. Upon success, a new user account will be created and can now log in via the frontend.

7.5 Verifying the Connection

- Ensure that both the backend and frontend are running over HTTPS.
- The frontend communicates with the backend using the following API endpoint format:

```
https://localhost:7163/api/...
```

- You can also test the backend APIs using the Swagger UI before interacting through the frontend.