P35: Business Process for SMEs utilizing E-invoicing APIs

Team WellP

COMP9900 Term 2 Computer Science Project

9900F14A
Team WellP

UNSW Sydney

Members:

Name	Email	Role
Lianqiang Zhao	z5396332@ad.unsw.edu.au	Scrum Master
Ruixi Liu	z5381549@ad.unsw.edu.au	Back-end
Linxuan Lyu	z5430933@ad.unsw.edu.au	Front-end
Yen-jung Huang	z547775@ad.unsw.edu.au	Front-end
Yifan Yu	z5415202@ad.unsw.edu.au	Back-end
Shiqi Yin	z5370300@ad.unsw.edu.au	Back-end

Contents

1 Introduction

1.1 Project Overview

This project aims to develop a business process for SMEs utilizing e-invoicing APIs. The project is built using Django for the back-end and React for the front-end.

1.2 Objectives

The main objectives of the project are to automate the invoicing process, ensure data accuracy, and improve efficiency for SMEs.

1.3 Report Structure

This report is structured into several sections, including an installation manual, system architecture diagram, design justifications, user-driven evaluation, limitations and future work, and engineering practices.

2 Installation Manual

2.1 Dockerization

To run the backend server, use the following Docker commands:

```
docker-compose up --build # Create Docker image
docker-compose up # Run Docker image
```

The backend server will start running.

Ensure you have the necessary environment variables and secrets configured. For environment variables, create a `.env` file in the project's root directory with the required settings. Secrets should not be included in the GitHub repository. Instead, provide detailed instructions on where to place the secrets in your project.

If your project cannot be dockerized (e.g., working with a client repository under restrictions or developing a mobile app), notify your tutor by the end of Week 8 for alternative instructions or approval for modifications.

1. Clone the repository:

```
git clone https://github.com/yourusername/InvoiceProcessing.git
cd InvoiceProcessing
```

2. Set up the virtual environment:

```
python -m venv env
source env/bin/activate
```

3. Install the dependencies:

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

4. Run the database migrations:

```
python manage.py migrate
```

5. Run the development server:

```
python manage.py runserver
```

1. Navigate to the front-end directory:

```
cd front-end
```

2. Install the dependencies:

```
npm install
```

3. Start the React development server:

```
npm start
```

3 System Architecture Diagram

3.1 Updated System Architecture Diagram

Provide an updated system architecture diagram based on the feedback received from your proposal and any design changes throughout the term. The diagram should include high-level explanations of the main components, detailing how user inputs/outputs and data flow through different components such as the frontend, backend components, models, and databases.

3.2 Component Explanation

3.2.1 Frontend

The frontend is built using React and Redux for state management. It communicates with the backend via RESTful APIs.

3.2.2 Backend

The backend is built using Django and Django REST Framework. It handles business logic, data processing, and provides APIs for the frontend.

3.2.3 Database

The database used is SQLite for development purposes. It stores user data, invoice data, and other necessary information.

4 Design Justifications

4.1 Design Evolution

1. **Changes in Project Requirements**: - Project requirements have evolved over the sprints to accommodate new features and client feedback. For instance, the need for JWT authentication was added later to enhance security.

2. **Modifications and Justifications**: - Initial Design: The initial design included basic CRUD operations for invoices without authentication. - Changes: JWT authentication and asynchronous task handling with Celery were added to meet security and performance requirements. - Current Design: The current design is more robust, with secure authentication, efficient data handling, and better user experience.

3. **Implementation of Complex Algorithms**: - Complex algorithms such as data extraction from PDFs and validation of invoice data have been implemented using third-party APIs and custom logic to ensure accuracy and reliability.

5 User-Driven Evaluation of Solution)

5.1 Main Features

- The main features include user authentication, invoice creation, validation, and viewing.

5.2 Evaluation Objectives

- Objectives include verifying the accuracy of invoice data, user satisfaction with the system, and performance metrics such as response time.

5.3 Effectiveness Evaluation

- The solution effectively meets the defined objectives. User feedback indicates high satisfaction, and performance benchmarks show that the system operates within acceptable limits.

6 Limitations and Future Work

6.1 Limitations

- Current limitations include dependency on third-party APIs for data extraction and limited scalability with SQLite.

6.2 Future Work

- Future work includes migrating to a more scalable database like PostgreSQL, improving the UI/UX based on user feedback, and adding more features such as advanced reporting and analytics.

7 Engineering Practices

7.1 Test-Driven Development

- Implemented a test-driven development approach to ensure code quality and reliability.

7.2 CI/CD Pipelines

- Set up CI/CD pipelines using GitHub Actions to automate testing and deployment.

7.3 Code Reviews and Pull Requests

- Conducted regular code reviews and used pull requests to maintain code quality.

7.4 Consistent Coding Practices

- Used consistent coding practices, including code formatting and linting, to ensure maintainability.

8 Report Quality and Formatting

8.1 Formatting Requirements

- The report conforms to the specified formatting requirements.

8.2 Readability

- The report is structured and easy to read.

8.3 Conciseness

- The report is detailed yet concise, providing necessary information without unnecessary details.

9 References