Invoicing System Requirements

The Bug Stoppers:

LAngelihle mhlongo

katlego chopo

sisanda nddlovu

2024

Table of Contents

[Overview 2](#_Toc163851809)

[User stories 3](#_Toc163851810)

[Use Case Diagram 4](#_Toc163851811)

[Entity Relationship Diagram 5](#_Toc163851812)

[Class Diagram 6](#_Toc163851813)

[Product Backlog 7](#_Toc163851814)

[Technical Architecture 9](#_Toc163851815)

[Source Control 9](#_Toc163851816)

[Directory and naming services 9](#_Toc163851817)

[Assumptions and Dependencies 9](#_Toc163851818)

[Lessons Learned 10](#_Toc163851819)

# Overview

This system will allow employees to create invoices for products rendered to customers. It typically includes fields for details such as item descriptions, quantities, prices, taxes, discounts, and payment terms.

Customers will have the capability to review their invoices both before and after making payments. Each invoice will contain all necessary details, ensuring clarity and transparency in the billing process. The system maintains a database of the customer information including contact details, billing addresses, payment preferences. This helps in accurately addressing invoices and tracking payment behaviour. Once generated, the invoices will be delivered to customers via email and electronic billing portals.

# User stories

This section represents the user stories involved in the project.

As a customer

I want to clearly display the due date for payment

So I can plan and manage my finances effectively

As a employee

I want to generate and send invoices to customers

So I can ensure accurate and timely billing

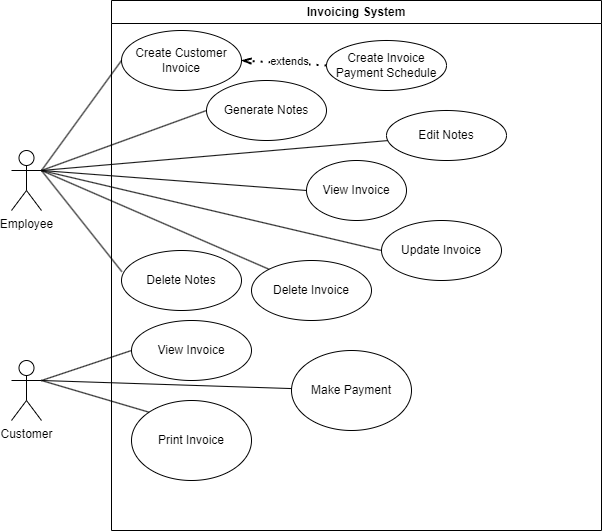
As a employee

I want to record payments made against an invoice

So I can keep track of outstanding balances and financial transactions.

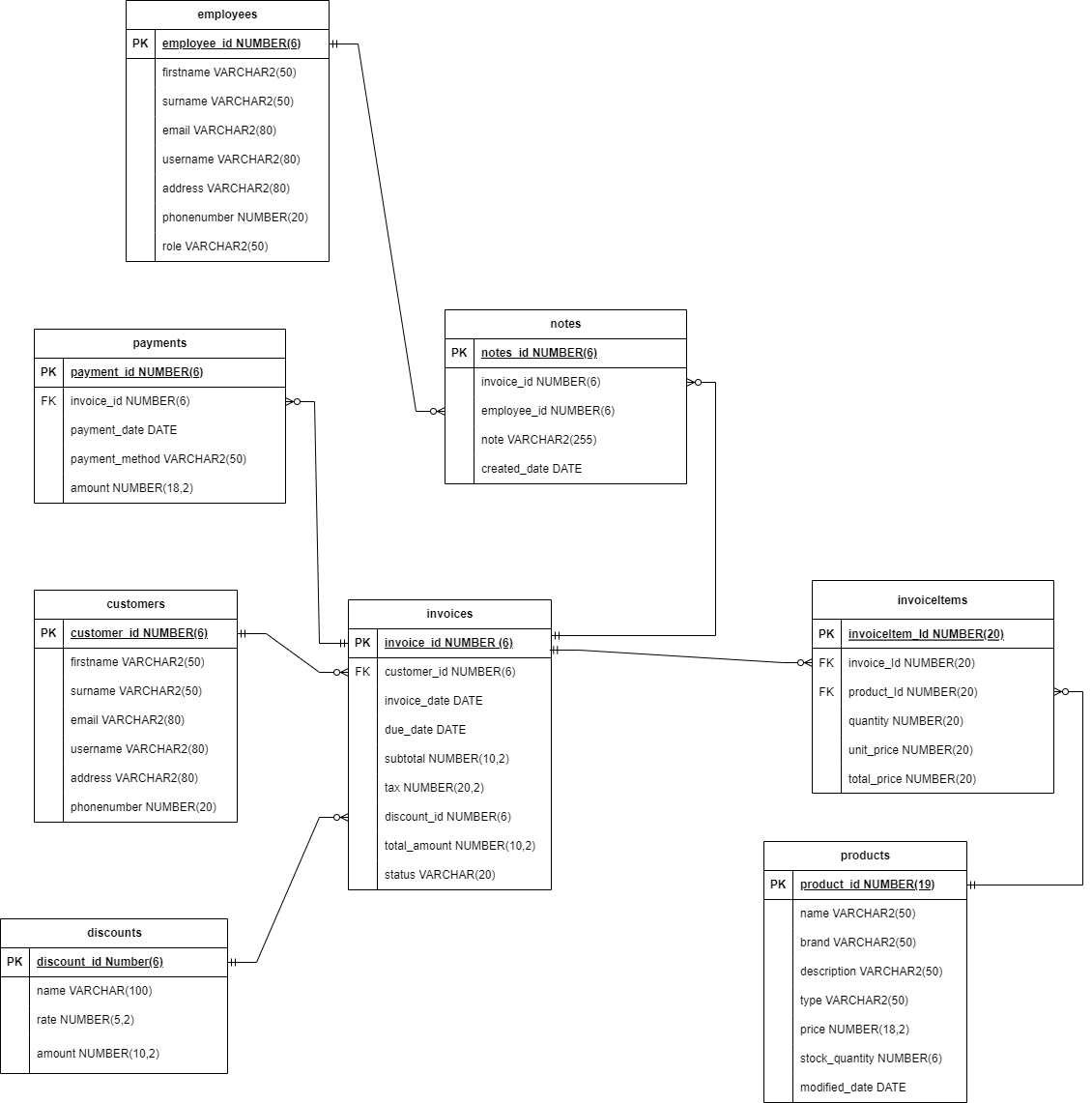
# Use Case Diagram

This section represents the use case diagram that describes the project’s use cases.



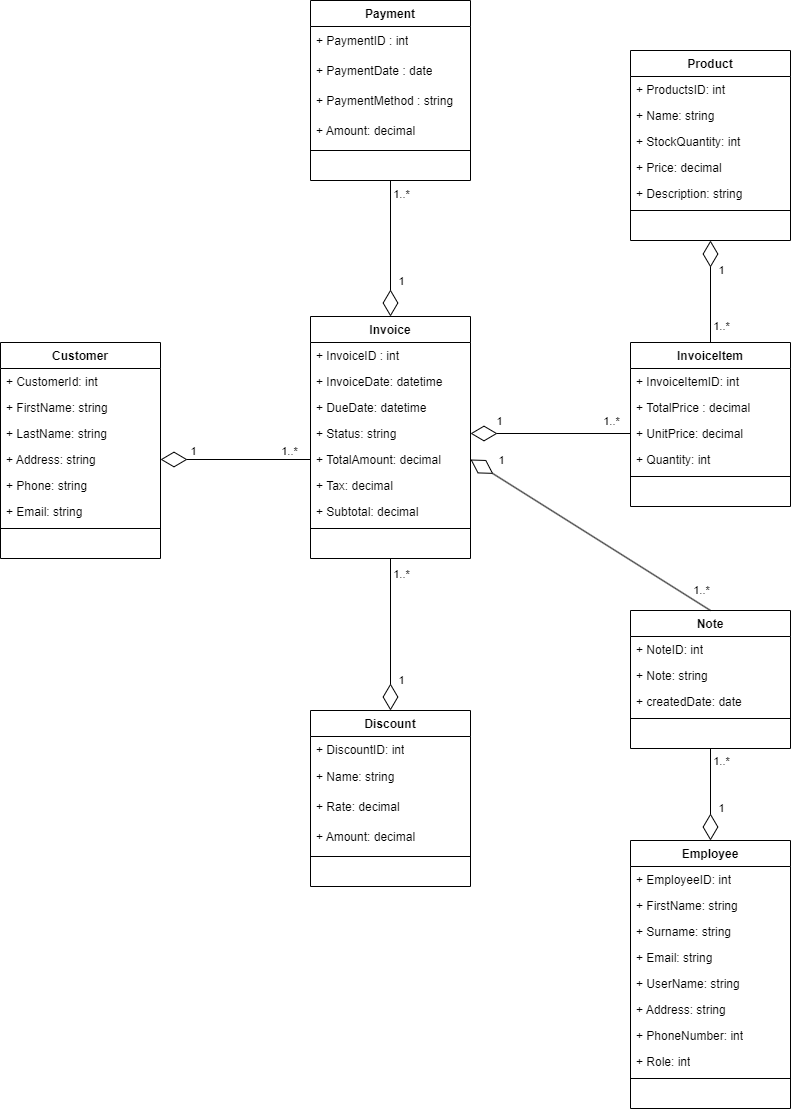
# Entity Relationship Diagram

This section represents the entity relationship diagram comprising all the entities involved in the project.



# Class Diagram

This section represents the class diagram comprising all the classes involved in the project.



# Product Backlog

This section represents the product backlog, which consists of the users, requirements, priority, and completion status.

The **Product Goal/objective** is to create full stack application using C# .NET as shown below:



**Sprint Backlog 1**



**Sprint Backlog 2**



**Sprint Backlog 3**



# Technical Architecture

The development environment for the system encompasses the tools, frameworks, and methodologies utilised throughout the software development lifecycle. This section outlines the primary components and configurations employed during development.

The development methodology adopted for the project is Agile, with a focus on iterative development, frequent releases, and continuous feedback loops. Specifically, the Scrum framework is utilised, employing sprint-based development cycles characterised by regular sprint planning, daily stand-ups, and sprint reviews.

In terms of development environment setup, the developer is provided with a high-performance workstation running Windows 11 or later. The workstation is equipped with ample RAM, CPU, and storage resources to support development tasks efficiently.

Visual Studio 2022 Community edition is the primary IDE utilised for development. It is configured with necessary components including the .NET Framework, and Git integration. Additionally, the developer may integrate additional development tools and plugins into Visual Studio as per project requirements. These tools and plugins may include code analysers, debugging extensions, and productivity tools, enhancing the development process and productivity.

# Source Control

Git Repo: **Invoicing-Console-App**

# Directory and naming services

LLM\_KC\_SN\_InvoicingSystem\_OOD4

# Assumptions and Dependencies

The assumptions and dependencies of this project are outlined in this section.

**Assumption:** Developers have access to high-performance workstations running Windows 11 or later. This assumption is based on the expectation that developers will have sufficient computing power, memory, and storage capacity to compile, test, and debug code efficiently. This assumption also implies that developers are familiar with the Windows operating system and can leverage its features and functionalities during development.

**Dependency:** Dependency on Visual Studio 2022 Community edition. The project relies on Visual Studio 2022 Community edition as the primary IDE for software development activities. This dependency is critical as Visual Studio provides a comprehensive suite of tools, libraries, and features necessary for building, debugging, and deploying applications using various programming languages, including C#. Furthermore, the dependency on Visual Studio 2022 Community edition offers a compelling combination of features, compatibility, accessibility, and community support, making it an ideal choice for facilitating efficient and cost-effective software development within the project.

# Lessons Learned

This section highlights the lessons learnt by the development team members.

**Langelihle’s Lessons Learnt:**

* In my experience, implementing log4net involves configuring it effectively, utilising various log levels, and logging contextual details to facilitate debugging and monitoring. I've learned the importance of balancing performance considerations with the need for comprehensive logging. Handling errors gracefully and thoroughly testing configurations are crucial aspects for ensuring reliable logging.
* In my experience, implementing revision control and versioning mechanisms to manage changes to requirements makes the work process effective. Versioning ensures that the evolution of requirements is documented, and changes are tracked over time. It also facilitates collaboration among team members and allows for the rollback of changes if needed.

**Katlego’s Lesson’s Learnt:**

* I gained a comprehensive understanding of submenus, dependency injection with frameworks, log4net for robust logging, and various design patterns. By effectively applying the command pattern in my code, I achieved improved code organization, maintainability, and testability.
* I also learned about ref keyword which consists of passing the reference of where the value type is in memory. I also learnt mock testing, and implemented add, delete, edit functionality for my repository classes, which will improve memory management, enable efficient unit testing, and provide a flexible data access layer.

**Sisanda’s Lesson’s Learnt:**

* I learnt about the design pattern Chain of responsibility when we as a group wrote a code that used it. I also learnt about the command pattern while using it in our project. When doing this week's project, having to use Moq, I gained more understanding, which helped me simulate behaviour and dependencies within our project for testing purposes. Exploring logging introduced me to tracking and monitoring the application's execution flow. Learning about LINQ provided me with a powerful tool for querying and manipulating data in c#. Using delegates enabled me to understand the passing of methods as parameters.
* These experiences broadened my understanding of software development practices and equipped me with valuable tools and techniques.

We also learned how to work as a team, which involved a lot of collaboration and helping each other complete the project. The collaboration involved a lot of communication between the team members and support. We learnt valuable skills from each other that will be helpful in our careers.