University of Groningen

Software Engineering: Invoice Engine

Requirements Document

Version 2.0

Authors:
Awin Gray *(s3073521)*
Xiaoyu Guan *(s3542807)*
Daniël Knap *(s4117743)*
Jesse Luchtenveld *(s3221245)*
Yao Xiao *(s3435679)*

Client:
A. van der Deijl

Teaching Assistant:
I. Botez

May 31, 2021

1 Introduction

Periodic bookings are of vital importance in our economy. Just consider the amount of periodic payments you make yourself. These could include monthly rent, yearly contributions to sports clubs, or quarterly tuition for educational facilities. A problem, however, is that invoicing is a very slow process in the Netherlands. With conventional systems it may take companies hours – if not days – to invoice their clients.

For this reason, a faster solution has been built in the `COBOL` programming language. This stand-alone application, named the *Invoice Engine*, is capable of handling a hundred thousand invoices in a matter of seconds. The *Invoice Engine* will simply be referred to as "the engine" from here on out.

The goal of this project is to rewrite this engine in a modern programming language, and to construct a web portal to accompany it. The final product should thus be a cloud service which companies can subscribe to. These companies can then use the service to make, change, or remove bookings; and to invoice all contracts at will. All data of all companies will be kept in one multi-tenant database. This means that while the data is kept in the same database, every object is related to a tenancy that defines who can access that object. See the *Architecture Document* for details, in particular Section 3.1.

In this *Requirements Document*, we will use the MoSCoW method in Section 2 to rank the functional requirements for the product in three categories: Must have (critical); Should have (important); and Could have (useful). We will then examine the non-functional requirements in Section 3 and list features that will not be implemented in Section 4. All meetings with the client will also be logged in this document, see Section 6. Finally, we conclude the document with a changelog.

2 Functional Requirements

This section contains a list of the requirement for the application. For each requirement, a short description is provided.

2.1 Critical

C.0 The system must make a distinction between two different types of users.

- The two user types will be referred to as 'Users' and 'Administrators'. The differences between the user types will be made clear in the following requirements.

C.1 The engine must work on a single database that contains all data of all clients.

- This database will be multi-tenant and Users must only have access to data corresponding to their own tenancies. Tenancies here is plural because a single User may have access to the administration of multiple companies. Administrators, in contrast, must only have access to the User and the tenancy tables.

C.2 There must be a user-authentication & authorization system.

- This requirement directly follows from the previous two: an organization must only have access to their own data, so it must be possible to distinguish between organizations. Moreover, it must be possible to distinguish between Users and Administrators.

C.3 Administrators must be able to add Users and tenancies to the service.

- A person must not be able to create their own account. Instead, they will contract the company behind the invoice engine to create an account and register a tenancy for them. The Administrator will take care of this.

C.4 Logged-in Users must be able to make, change, or delete contracts for their tenancies.

- A contract specifies the periodic booking between a tenancy and one or more persons (price, duration, etc.). When a contract has been activated, it must be possible to change the end date only.

C.5 A contract must consist of components (or contract lines).

- A single contract can be made for multiple services or multiple people receiving the same service. Take the example of two siblings whose parents pay contribution to the same sports club for both of them. One contract with two contract lines (one for each sibling) can be used. When the corresponding contract has been activated, the User must be able to change only the start and end dates.

C.6 It must be possible for multiple persons to be registered to one contract.

- Using the same example as above, the mother and father could both pay 50% of the contribution. To make that possible, they must both be registered to the contract. It is also important to ensure that for every contract, the contract persons add up to 100% at any given time during the contract's run.

C.7 The system must make a distinction between value-added tax (VAT) and the base price of a contract/component.

- The vast majority of countries employ a VAT, so it is important to adhere to these rules by accurately tracking how much money should be paid to the tax authority.

C.8 Logged-in Users must be able to run the invoice engine for their own tenancies' contracts at will.

- In this step, invoices are generated (one per contract) for all contracts corresponding to a tenancy. Collections which specify the amount of money a person has to pay and how they will do this, will also be made, as well as posts for the company's general ledger.

C.9 An invoice must consist of one or more invoice lines.

- This is similar to the contract consisting of components. One invoice line must be generated for each component.

C.10 There must be a post for the general ledger for each invoice and for each invoice line.

- This is a bookkeeping requirement. It is important for any company to keep track of their financial transactions.

C.11 The system must not use external services.

- Using external services could be slow and/or expensive. Therefore, the application must be self-contained.

2.2 Important

I.1 There should be four fixed (month, quarter, half year, year) invoicing periods, and a custom option.

- The fixed periods are the most common ones. The custom option allows the user to make contracts for a certain number of days.

I.2 When a price change happens, the cost of the invoicing period before the change should be calculated with the old price, and the rest should be calculated with the new price. This should also be made clear with multiple invoice lines.

- To give an example, perhaps the rent goes up on the first of January, but my contract runs until January 15th. I will therefore have to pay the old price from December 15th until January 1st, and the new price for the rest of the period.

I.3 There should be multiple pricing methods: price per period, price per unit, and price per day.

- A pricing option per day is especially important for the custom invoicing period mentioned earlier.

I.4 A collection file (or debit file) should be made for each payment.

- The system should output collection files, which is proof of a collection from a seller to a buyer.

2.3 Useful

U.1 The system can retroactively change the price and send out another invoice.

- For example, an invoice has gone out for the month January with a certain price, but this price was incorrect. In such a case, another invoice can be sent out for the difference between the old and the new price. This can be either negative or positive.

3 Non-functional Requirements

NF.1 The system must be capable of invoicing 100 000 contracts on the timescale of minutes.

– This is the main reason for writing this application: speeding up the invoicing process. There is no specific time limit, but up to twenty minutes is acceptable for 100 000 contracts.

NF.2 The system should be scalable for the future.

– There may be many organizations representing many companies with many clients to invoice wanting to use the system. This means that the amount of data could get very large.

NF.3 The database must be secure.

– It is important to realize that the database will contain personal details and company data which must be protected. Therefore, it must be ensured that only the owners of the data can access it, and nobody else. The associated functional requirements are C.1-2.

4 Will-not-haves

- The system can book a void contract (mainly useful for the housing market).
  - A void contract is used in the housing market if there is an apartment in which nobody currently resides. This represents the cost for the company of having an empty apartment.
- The database has search tables as well as storage tables.
  - Search tables can be constructed after the invoicing process has concluded. These search tables contain more indexes to make it faster to request invoices based on a person's name, or a given amount for example.
- The system can calculate the cumulatives of the contracts (mainly useful for the housing market).
  - A cumulative represents the total amount that was paid for one contract, over a certain time period.

5 Traceability Matrix

This section gives an brief overview of how the requirements were implemented in the project. For more details, see the *Architecture Document*.

ID	Component(s)	Files	Test	Comment
C.1	Tenancy, TenancyAccessMixin	`models.py`, `parent_views.py`	Manual	The Tenancy class is used to refer to by all other database tables. The TenancyAccessMixin class is used to verify whether a user has access to the requested data. There is no automated test. Because the amount of URLs is relatively low, this requirement can instead be verified by visiting all CRUD URLs.
C.2	Auth app	N/A	N/A	The authentication app included in the Django library is the basis of the User authentication system. Administrators will use the Django admin page.
C.3	Admin page	`admin.py`	N/A	Only User and Tenancy tables are included in the admin page, such that administrators can use it to manage these.
C.4	Contract	`models.py`	ContractTest	
C.5	Component	`models.py`	ComponentTest	
C.6	ContractPerson	`models.py`	ContractPersonTest	
C.7	VATRate	`models.py`	VATRateTest	

C.8	Tenancy, Contract, Component, ContractPerson, Invoice InvoiceLine	`models.py`	test_logic	This involves a number of methods: `Tenancy.invoice_contracts()`, `Contract.invoice()`, `Contract.compute_date_next_prolongation()`, `Component.invoice()`, `ContractPerson.invoice()`, `Invoice.create_gl_post()`, `InvoiceLine.create_gl_posts()`.
C.9	Invoice, InvoiceLine	`models.py`	InvoiceTest, InvoiceLineTest	
C.10	GeneralLedgerPost	`models.py`	GeneralLedgerPostTest, test_logic	These objects are created with the `create_gl_post()` functions in Invoice and InvoiceLine.
C.11	N/A	N/A	N/A	No external services are used.
I.1	Contract	`models.py`	test_logic	The specific test is given by `test_compute_date_next_prolongation()`.
I.2	Contract, Component	`models.py`	test_logic	This was specifically implemented by not allowing two Components of the same BaseComponent referring to one Contract at any given date. When a new one is added, the existing one is ended. When the new one is added in the past, a correction is issued. Specific tests are `test_create_correction_invoice()` and `test_create()`.
I.3	Contract	`models.py`	text_logic	At time of writing, the 'price per day' option remains unimplemented. Specific tests are `test_invoice_unit()` & `test_invoice_normal()`.
I.4	Collection	`models.py`	test_logic	This is implicitly tested in `test_create_correction_invoice()` & `test_create()`.
U.1	Contract, Component	`models.py`	test_logic	See I.2, as this is a clear extension of that requirement.

6 Meeting Log

In this section, we provide an overview of all meetings with the client. The date of the meeting and the key items that were discussed are indicated in the table below.

Date	Description
Feb. 18, 2021	We received more detailed project specifications from the client. He showed us a presentation about the product requirements.
Mar. 04, 2021	We discussed the entity relationship diagram and the `COBOL` source code. The client also promised to send us a proposal for sprint goals.
Mar. 18, 2021	We discussed a counter-proposal for the sprint goals, as the timeline it tighter than the client had assumed. Moreover, we discussed the possibility of search tables as well as retroactive price changes. This document was updated accordingly.
Apr. 01, 2021	The agenda included topics such as the user interface and front-end needs. Furthermore, the possibility of using a REST framework was discussed. It was decided not to implement this.
Apr. 15, 2021	The client provided more details on what a general ledger post should contain. This is crucial information to correctly implement this table.
Apr. 29, 2021	The mechanisms and desired outcome of processing retroactive price changes were discussed, for us to implement in the following sprint.
May 27, 2021	We showed the client a nearly-fully functional prototype of the application and discussed testing.

7 Document Changelog

In this section, the changelog for this document is given.

Date	Author	Description
Feb. 22, 2021	J. Luchtenveld	Created the document structure, made a title page, and wrote an introduction.
Feb 23, 2021	J. Luchtenveld	Made a preliminary list of requirements for discussion.
Mar 02, 2021	J. Luchtenveld	Updated the list of requirements.
Mar 09, 2021	J. Luchtenveld	Updated the list of requirements.
Mar 23, 2021	J. Luchtenveld	Updated the list of requirements & the meeting log. Added sprint schedule.
Mar 29, 2021	A. Gray	Reviewed and addressed previously made comments and annotations.
Mar 29, 2021	J. Luchtenveld	Cleaned up the requirements and made a final pass.
May 27, 2021	J. Luchtenveld	Incorporated feedback from the teacher & updated meeting log. Rearranged and provided more specificity in the requirements.
May 30, 2021	J. Luchtenveld	Added a brief traceability matrix/table.

Appendix A: Sprint Schedule

Start	End	Description	Period	#Lines	Pricing	GLPost	VAT	Changes	Output
Mar. 04	Mar. 18	Front end basic tables							
Mar. 18	Mar. 30	Front end other tables							
		Invoice Engine v1	Month	1	Fixed	No	No	No	No
Mar. 30	Apr. 15	Invoice Engine v2	MQHY	n	Fixed	No	No	No	No
Apr. 15	Apr. 29	Invoice Engine v3	MQHY	n	Fixed	Yes	Yes	No	No
Apr. 29	May 13	Invoice Engine v4	MQHY	n	Fixed	Yes	Yes	Yes	Collection
May 13	May 27	Invoice Engine v5	MQHYC	n	FDU	Yes	Yes	Yes	Collection
May 27	Jun. 10	Output & Front end	MQHYC	n	FDU	Yes	Yes	Yes	Collection Invoices

This is the sprint schedule that was negotiated with the client. The table is too large to fit on the page normally, so it has been rotated. Each sprint is two weeks long. Following is a description of some of the terms used:

- MQHYC: Month, Quarter, Half-year, Year, Custom. These are different invoicing periods.
- FDU: Fixed, Day, Unit. These are different pricing methods. A price can be fixed (per period), defined per day, or defined per unit.
- Changes: this refers to price changes during the invoicing period.