

UNIVERSITY OF GRONINGEN

SOFTWARE ENGINEERING: INVOICE ENGINE

Requirements Document

Version 1.0

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

March 30, 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 5. 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

1. The engine must work on a single database that contains all data of all clients.
 - This database will be multi-tenant and organizations (users) must only have access to data corresponding to their own tenancies. Tenancies here is plural because a single organization can do the administration of multiple companies.
2. There must be a user-authentication system.
 - This requirement directly follows from the previous one: an organization must only have access to their own data, so it must be possible to distinguish between organizations.
3. 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.).
4. 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.
5. 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.
 6. 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.
 7. 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.
 8. 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.
 9. 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 where financial transactions.
 10. 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

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.
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.

3. There should be multiple pricing methods, such as price per period and price per day.
 - A pricing option per day is especially important for the custom invoicing period mentioned earlier.
4. A debit file must be made for each payment.
 - The system should output debit files, which is proof of a collection from a seller to a buyer.

2.3 Useful

1. 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.
2. 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. It should have been higher. In such a case, another invoice can be sent out for the difference between the old and the new price.
3. 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.

3 Non-functional Requirements

- 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.
- The system should be scalable in 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.
- 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.

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.

5 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 is 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 includes topics such as the user interface and front-end needs. Furthermore, the possibility of using a REST framework will be discussed.

6 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.

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.