Team 1

SOEN 390

Architecture Description of

MVC for

Downhill ERP System

Version 4

Submitted on

April 7th , 2020

1 Introduction

This chapter describes introductory information items of the AD, including identifying and supplementary information.

1.1 Identifying information

The architecture that will be used is MVC (Model View Controller). The system of interest is a website for enterprise resource planning to be used by a bike company.

1.2 Supplementary information

The approving authority is the product owner. His company manufactures regular bikes. The website will be built from ground up and will implement the specific requirements to allow for a minimal ERP. It shall allow the following aspects of enterprise resource planning for the company: planning, scheduling, vendors (procurement), production, quality management, packaging, transport planning/shipping, sales, and accounting.

1.3 Other information

1.3.1 Overview

The purpose of this architecture is to allow for the separation of responsibilities thus allowing for easier coding, debugging, and testing since each component has a single job. The MVC architecture allows to separate the application into three main components: the model, the view, and the controller. Each component has its own responsibilities. The model component takes care of everything related to data logic. The model holds the data and its logic, it receives new information or modifications, and it sends information back to the other components. The view takes care of representing the information to the users. It can request data to represent as well send new information depending on user actions. The controller takes all transactions between the model and its corresponding view. It also interprets the mouse and keyboard information and then sends command to the model and view to change.

The system of interest, as mentioned before, is a website that allows enterprise resource planning for a bike company. The website should allow for a minimal number of features that are at the core of ERP. The minimal requirements have been identified by the product owner who is the main person our team will interact with throughout the project.

The rest of this architecture documentation will begin by touching upon the stakeholders related to the project. Then the major concerns about the project and its architecture will be discussed and will be traced to the respective stakeholders. The third major aspect in this document will be viewpoints. A viewpoint is an architectural convention to which a view must adhere to and the different requirements for it will be presented. Concerns and stakeholders, model kinds, and operations on views will be presented for each viewpoint. The fourth topic are the views themselves and the final point will be consistency and correspondences. Rationale for decisions will be discussed throughout the document when it applies.

1.3.2 Architecture evaluations

1.3.2.1 Version 1

No evaluation documented for this version.

1.3.3 Rationale for key decisions

The first major decision was to decide what architecture to use for the bike company ERP website. The pattern chosen was the Model View Controller pattern more commonly known as MVC. This architecture is perfect for developing a website since generally a website will be split into a frontend (view), a backend (controller), and a database (model). Many web frameworks have been developed to enforce this pattern. MVC also allows for a faster and overall better distribution of tasks and development because of its features. This is important since the development team is composed of 9 students and they only have 4 months to complete the project. It is also important since the pattern allows to easily create multiple views for a model. This website will be used by multiple types of employees but each will have their own access restrictions.

2 Stakeholders and concerns

2.1 Stakeholders

There are multiple stakeholders involved in ERP as well as the bike company. Here are the most important:

1. The customers who buy the bike are of course one of the main stakeholders. The entire company revolves around having the number of customers grow thus increasing their profit. The bikes are built for them. (System of Interest stakeholder)
2. The investors for the company are also very important since it is their money that allows the company to exist thus, they have some say in the major decisions of the company. (System of Interest and Architecture stakeholder)
3. The employees, such as the ones who build and assemble the bikes as well as their manager allow the company to fulfill its goals of selling bikes efficiently and consistently. Also the end users in some cases. (System of Interest stakeholder)
4. The accountants for the company work on assuring that the company finances are consistent and that the company will not become bankrupt. (System of Interest stakeholder)
5. The product owner and company executives are the leaders of the company and they are the ones who decide in what direction to steer the company. They make some of the major decisions related to the company. (System of Interest and Architecture stakeholder)
6. The vendors allow for the company to build bikes and to have a product. (System of Interest stakeholder)
7. The programming team builds the website that brings the entire company together. The website allows for a much more efficient flow throughout the entire process of getting the parts, building the bike, and then selling it. Some of the decisions are left up to them when building the website. (Architecture stakeholders)

2.2 Concerns

The purpose of this website is to allow a bike company to function more efficiently all from one place. The website will cover the basic functionalities of enterprise resource planning. With this program, employees and managers will spend a minimal amount of time on administrative tasks and more on their specific responsibilities since they can quickly interact with the website that connects the entire company together. The website will allow information to traverse within the company much faster which will lead to a more successful business.

The architecture is very suitable for the system of interest. The website will be accessible both through a computer and a phone for all employees of the company. The separateness of the MVC architecture allows for an easier distribution of tasks within the developers. Each programmer will work on an aspect he or she is comfortable with. Because there is a clear separation between the components, it will be easy to separate certain views depending on the employee's role within the company. Some views will be visible to some while not accessible for others. The website can load faster because of the architecture and asynchronous techniques. Bugs are very easy to track down in this architecture.

Because of all the pros that come with the MVC architecture mentioned above, the system of interest is very feasible for a team of nine developers.

If the system of interest is attacked by hackers, a lot of confidential information can be stolen since the website will hold all data regarding the company including financial data. If the website is not built properly, meaning that an employee who should not have access to the financial section of the website gains access to it, there can be confidential information released. Another risk is if the website has a problem in its model component; this can lead to multiple errors such as people not being charged or being charged extra, loss of information etc. There is also the risk of the customers not liking the website due to its design or other aspects which can decrease sales considerably. Another problem can arise if the employees can not use the website properly. It can lead to confusion, loss of information etc.

The system of interest will be maintained by a group of developers who will continually add new features and improve the website to meet the requirements of the different stakeholders. After an initial release, new requirements will be visible over time and the developers will make sure to implement and fix already existent problems and bugs.

2.2.1 Project size

The first concern is that an MVC architecture project can become big very fast. It requires many files to perform simple tasks. Managing and maintaining all the files can become very difficult but it is absolutely necessary for the project to evolve smoothly.

2.2.2 Controllers' size

Since there is a separation in each components' responsibilities, a lot of the logic is dumped onto the controller section. The controller becomes so big that testing can become very difficult thus slowing down the overall production.

2.2.3 Component maintenance

Separating the different features into three components definitely has its advantages. However, a little concern arises from it as well; it requires the developer to maintain all three components when working on a feature.

2.2.4 Usability

Usability is an important concern. The website must be easy to use for all employees and stakeholders within the company or else it would be counter-productive.

2.2.5 Security

Security is of major importance. The right accesses have to be given to the right people to ensure that critical information is not exposed. Also outside negative sources also have to be considered and protection has to be established against them.

2.3 Concern–Stakeholder Traceability

Table 1: Association of stakeholders to concerns

	Employees	**Accountants**	**Product Owner(s)**	**Developers**	**Customers**	**Parts vendors**	**Investors**
File size				x			
Controller size				x			
Separate components				x			
Security	x	x	x	x	x	x	x
Usability	x			x			

3 Viewpoints

3.1 Information structure viewpoint

The Information Structure viewpoint is used to visualize business concepts at the application level. This viewpoint is mostly related to the Model part of the MVC architecture.

3.1.1 Concerns

This viewpoint addresses the identification of application level concepts, which facilitate components development by narrowing their responsibilities and highlighting their relationships. This influences decisions at the Details abstraction level, which are related to these previously identified concerns (See section 2.2):

- Project size
- Controllers' size
- Component maintenance

While these are mainly technical concerns, they are directly affected by the choices made in the Information Structure viewpoint, since it is meant to clarify application level concepts and their scope.

3.1.2 Typical stakeholders

The typical stakeholders for this viewpoint are domain architects, but also developers, who need to have a sharp understanding of the domain concepts.

3.1.3 Model kinds

3.1.3.1 Information model

The information model identifies a system's structure of the business information.

Conventions

This model is usually created using UML and is represented by a simplified class diagram. It's elements include business concepts shown as classes and interactions shown as relationships.

The standard UML relationships are often used to show inheritance, concept composition or relationships specific to the identified domain.

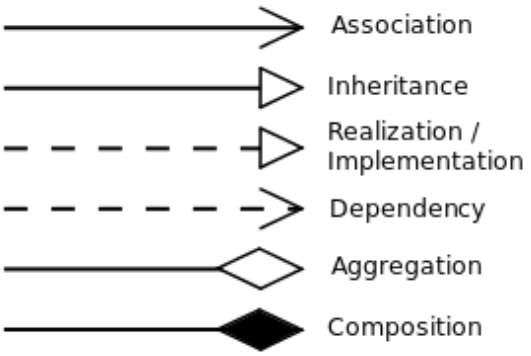

Figure 1: UML relationships

3.1.4 Correspondence rules

This viewpoint concentrates on a single information model, which, as described in section 3.1.3.1, is easy to create using UML. Thus, there are not many model kinds and the only rules to respect are those set in this modeling language.

3.1.5 Sources

The Open Group. “Architecture Viewpoints.” pubs.opengroup.org. https://pubs.opengroup.org/architecture/archimate2-doc/chap08.html#:~:text=Views%20are%20an%20ideal%20mechanism,to%20a%20set%20of%20stakeholders.&text=Viewpoints%20are%20a%20means%20to%20focus%20on%20particular%20aspects%20of%20the%20architecture. (accessed Jan. 23, 2021).

3.2 Actor co-operation viewpoint

This viewpoint focuses on actors and their interactions with each other through the application. It helps highlight external dependencies and establishes a network of interactions that builds a business process. The actor co-operation viewpoint will define parts of all the MVC aspects and represent its structure.

3.2.1 Concerns

This viewpoint isn’t directly concerned with the user interface dimension, however it helps shape the way users will access the different systems. That is why it is linked with the Usability concern (see section 2.2.4). Because the viewpoint highlights dependencies and interactions between components, we can specify security details such as encryption, addressing the Security concern (see section 2.2.5).

3.2.2 Typical stakeholders

The related stakeholders for this viewpoint are the project owner and domain/process architects (which are developers in this case). The project owner, who represents the company executives, is responsible for laying out the business requirements for the architects to design the business process.

3.2.3 Model kinds

3.2.3.1 Context model

The context model displays how data is structured in a system. It includes its environment and the related actors.

Conventions

Using UML, context modeling can be achieved with a system context diagram, which defines parts of the system, actors and their interactions in the form of inputs and outputs. This diagram can help create a component diagram, as they share similar content.

3.2.4 Sources

The Open Group. “Architecture Viewpoints.” pubs.opengroup.org. https://pubs.opengroup.org/architecture/archimate2-doc/chap08.html#:~:text=Views%20are%20an%20ideal%20mechanism,to%20a%20set%20of%20stakeholders.&text=Viewpoints%20are%20a%20means%20to%20focus%20on%20particular%20aspects%20of%20the%20architecture. (accessed Jan. 23, 2021).

4. Views

4.1 Logical view

The logical view defines the system at the conceptual level, including the connections between the high level components. It also provides information about the system's functional aspect.

4.1.1 Models

This view can be represented by more types of UML diagrams, like state and activity. However, the first step is to define the system's domain, so that all application level concepts are defined. Lower level class diagrams can also be used to represent created classes.

The following class diagrams in the logical view all use the IModelList model that can be seen in Figure 2 below.

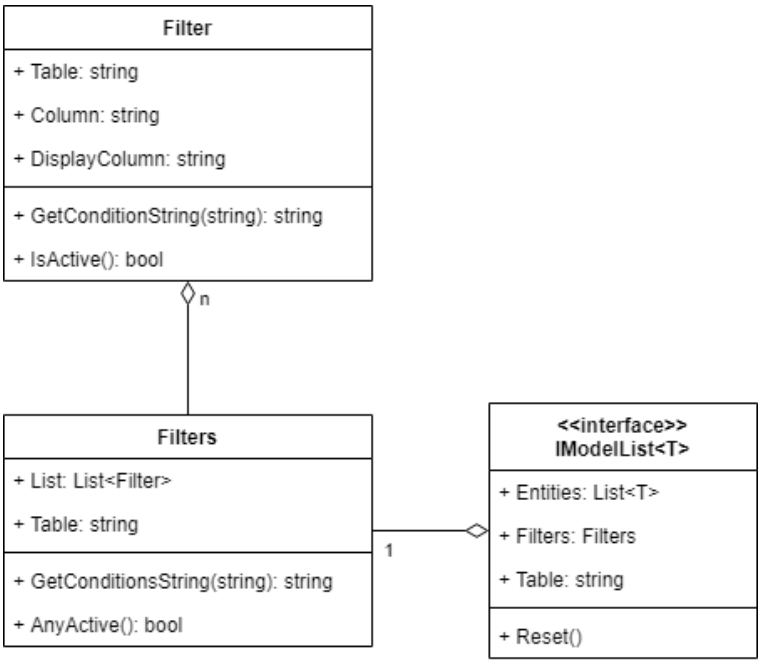

Figure 2: IModelList model (Version 1)

4.1.1.1 Domain model

The domain model adheres to the information model conventions (see section 3.1.3.1), which is the Information Structure viewpoint's main model kind. Indeed, it defines the domain concepts that will be needed at the application level:

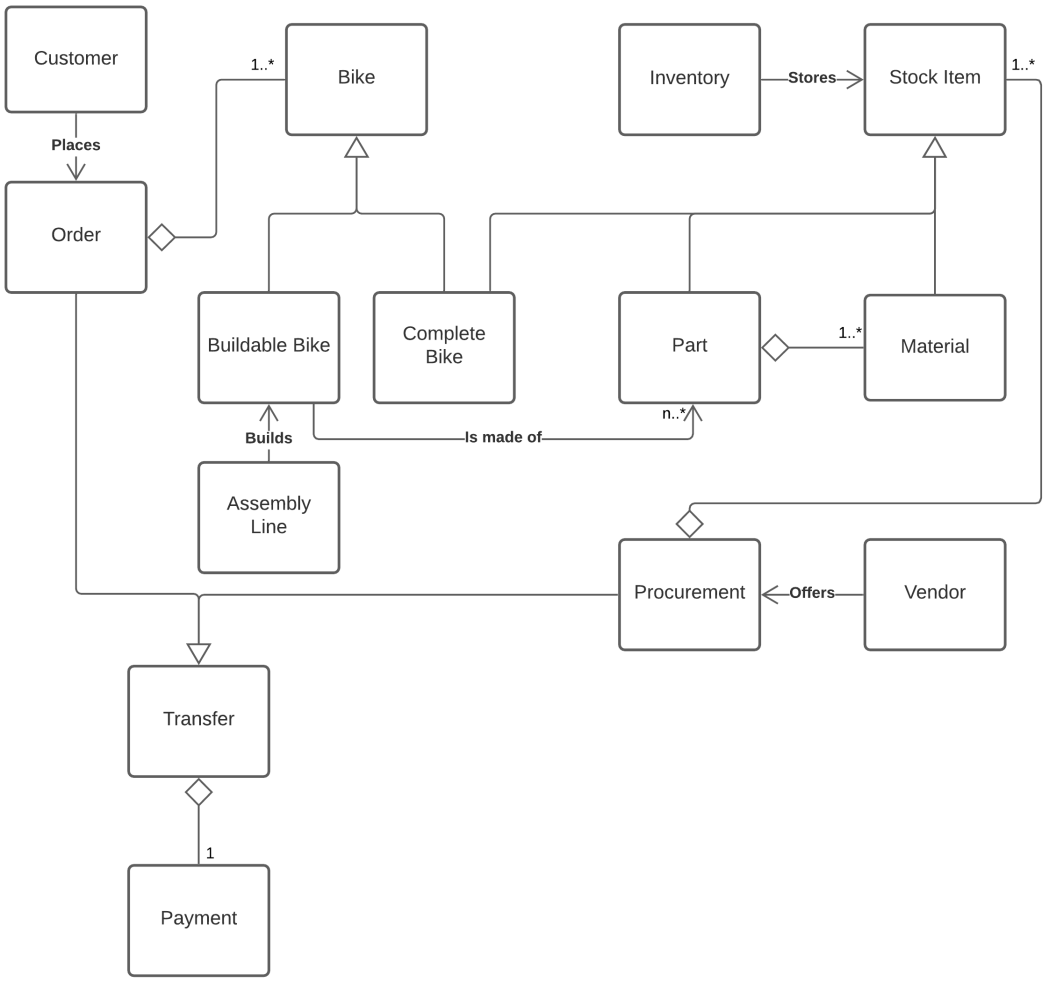

Figure 3: Domain model (Version 2)

4.1.1.2 Transfers Class Diagram

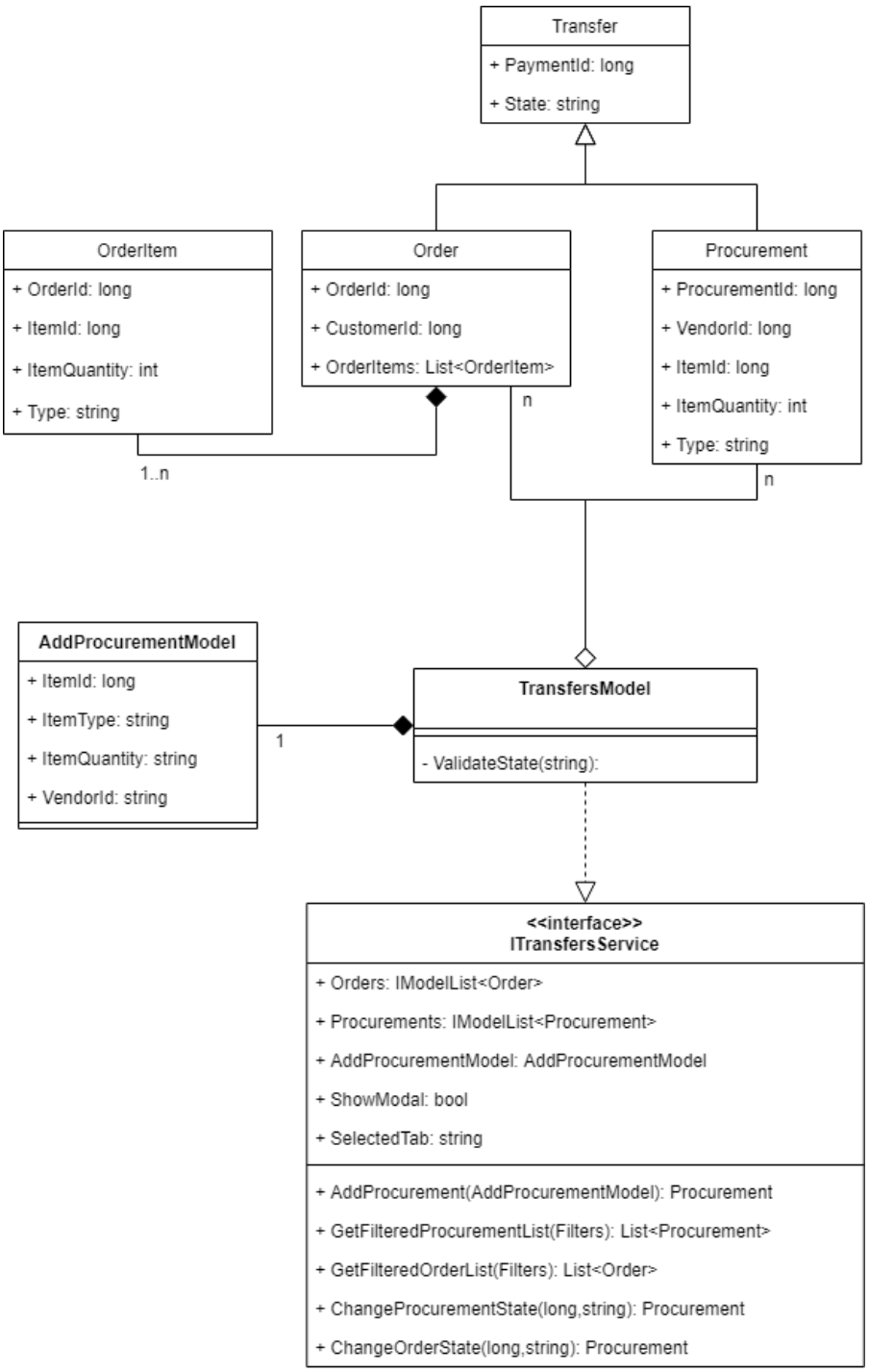

Figure 4: Transfers Class Diagram (Version 2)

4.1.1.3 User Management Class Diagram

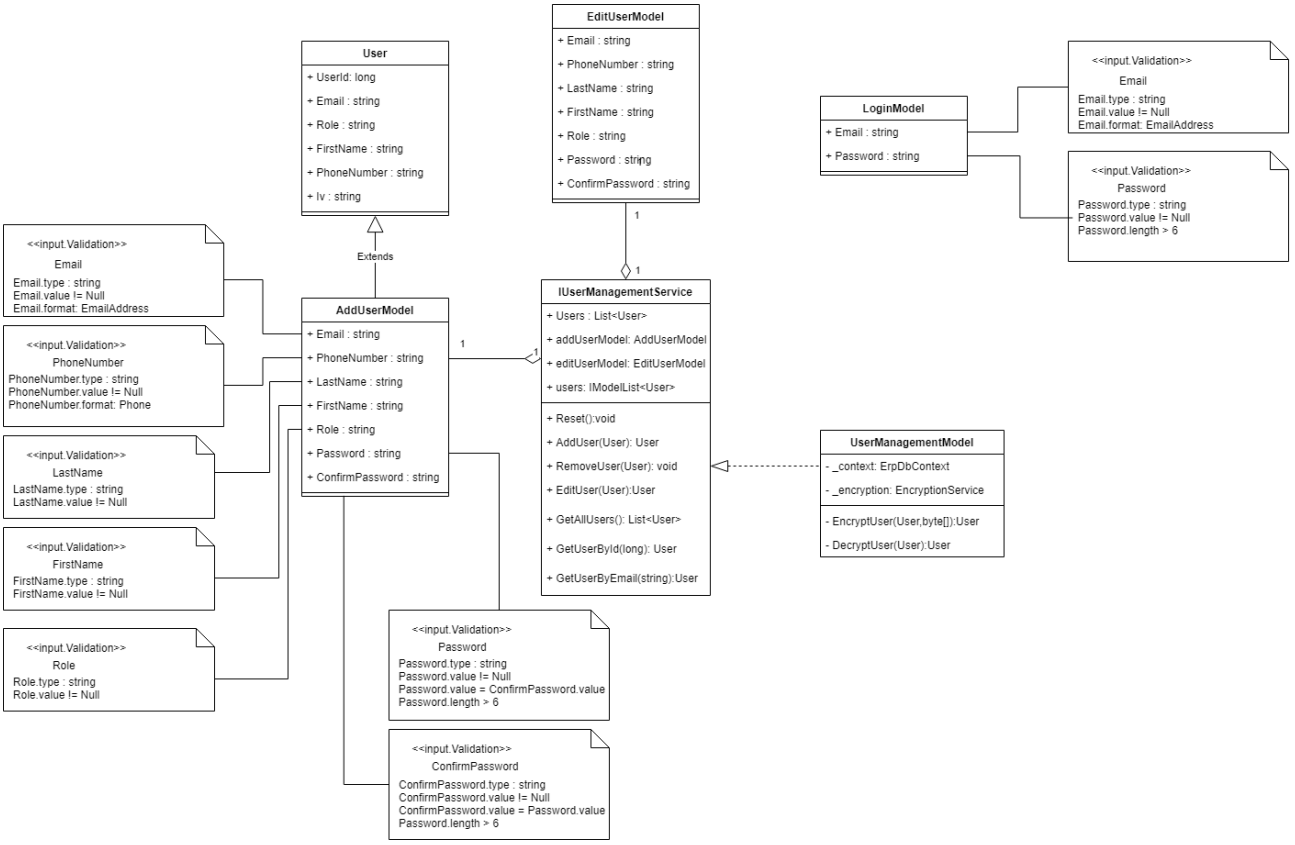

Figure 5: User Management Class Diagram (Version 2)

4.1.1.4 Inventory Management Class Diagram

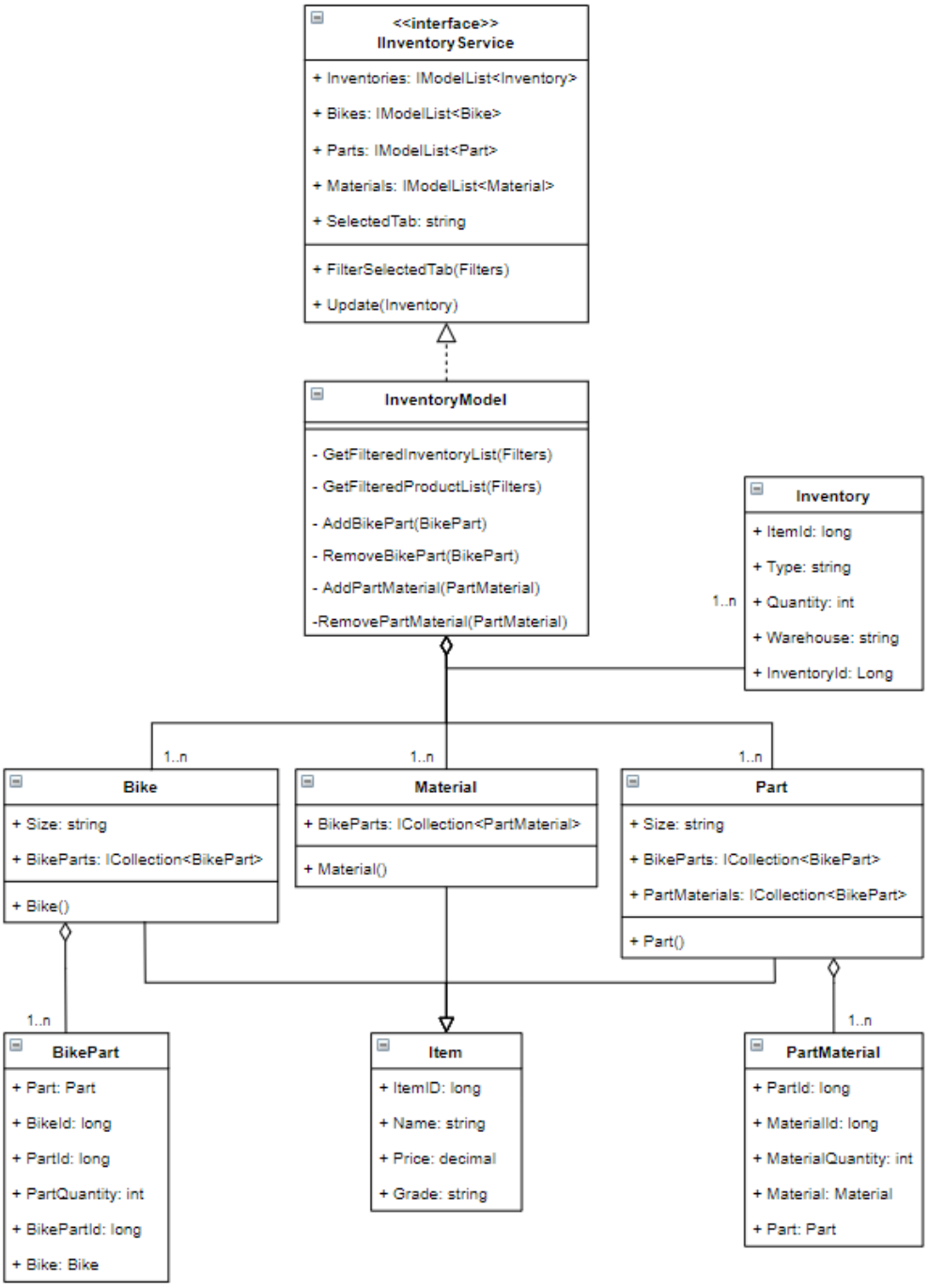

Figure 6: Inventory Management Class Diagram (Version 3)

4.1.1.5 Accounting Class Diagram

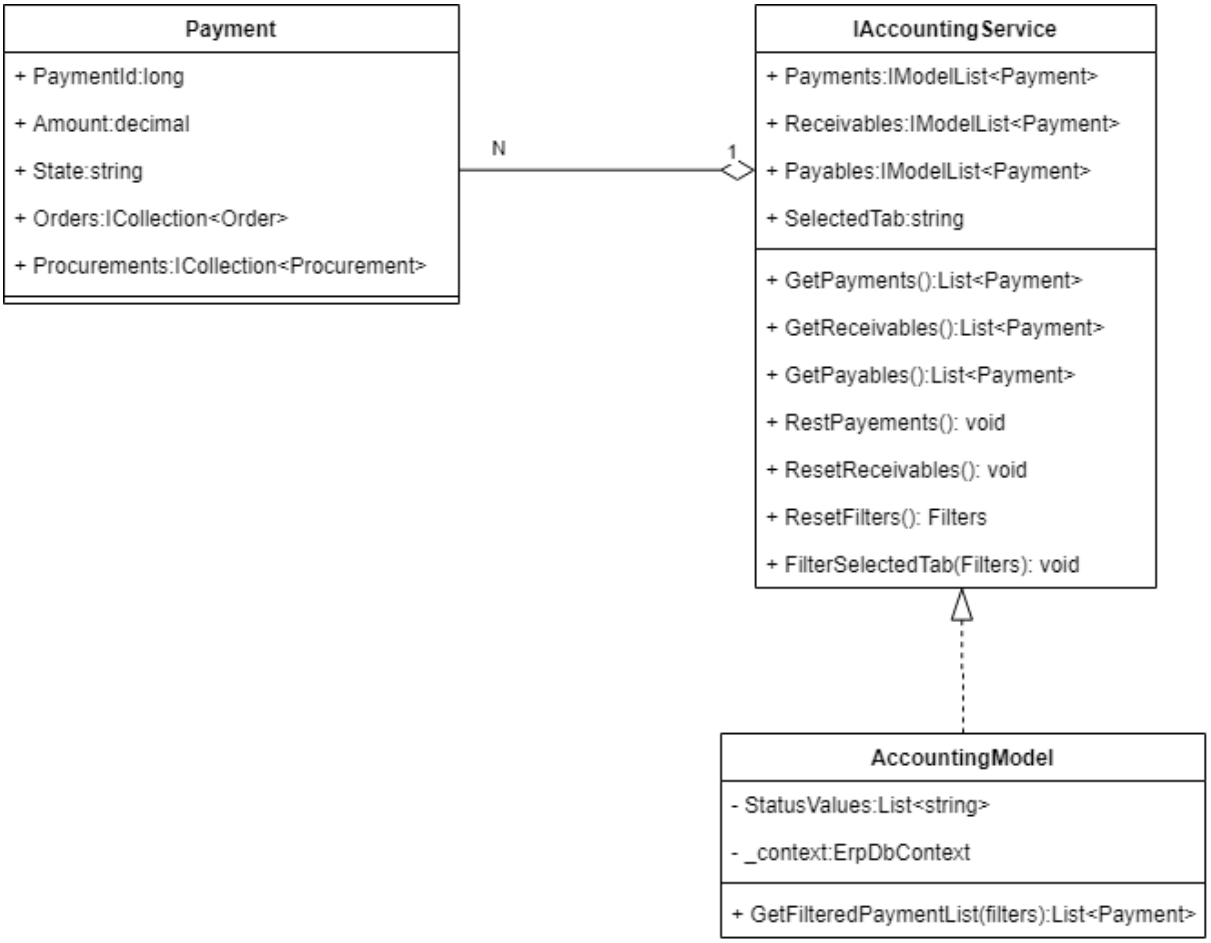

Figure 7: Accounting Class Diagram (Version 1)

4.1.1.7 Assembly Class Diagram

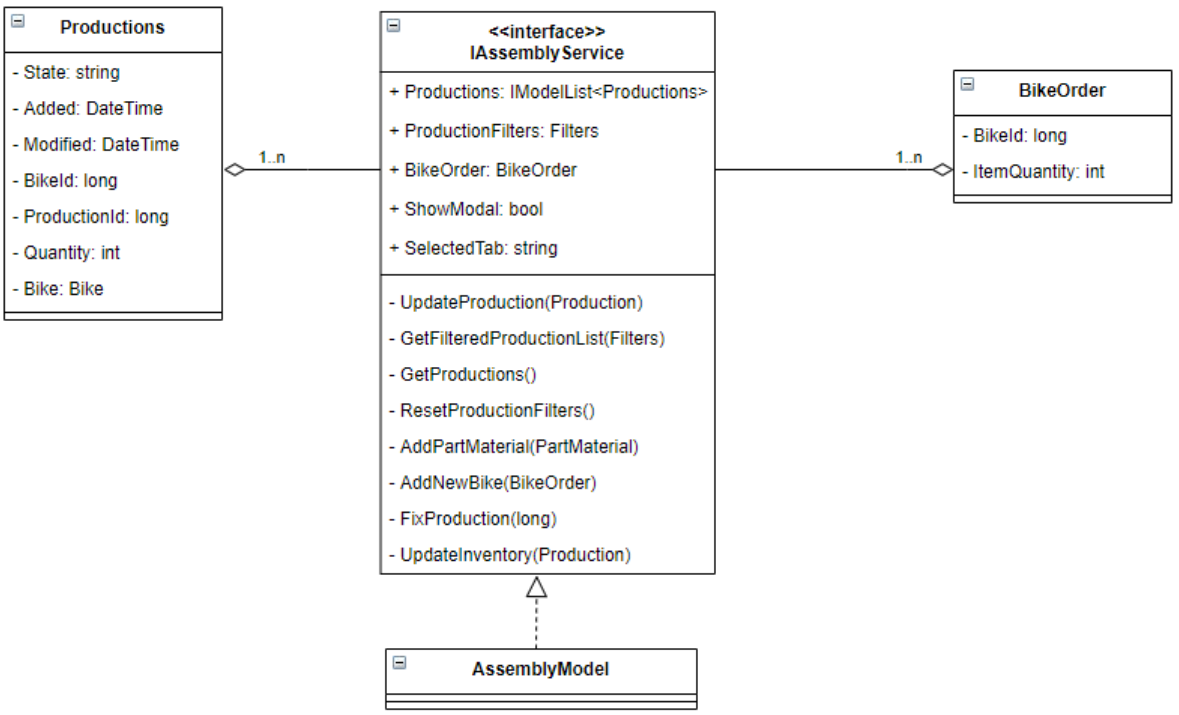

Figure 8: Assembly Class Diagram (Version 1)

4.1.1.8 Triggers Class Diagram

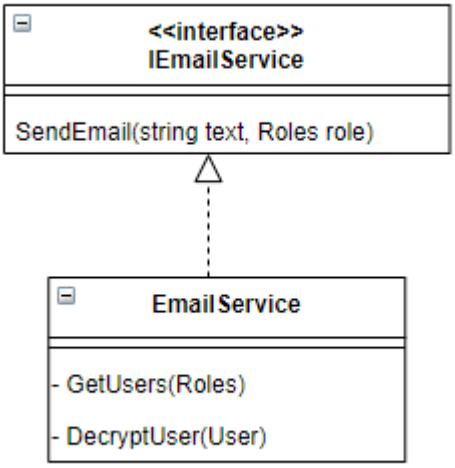

Figure 9: Triggers Class Diagram (Version 1)

4.2 Development view

Like it's name is hinting, the development view describes the system at the implementation level. This translates to the representation of the system's components near the file level.

4.2.1 Models

The development view can include the system's file structure representation through a package diagram regrouping related components, but it mainly uses the component diagram to describe the system's components at the implementation level.

4.2.1.1 Component diagram

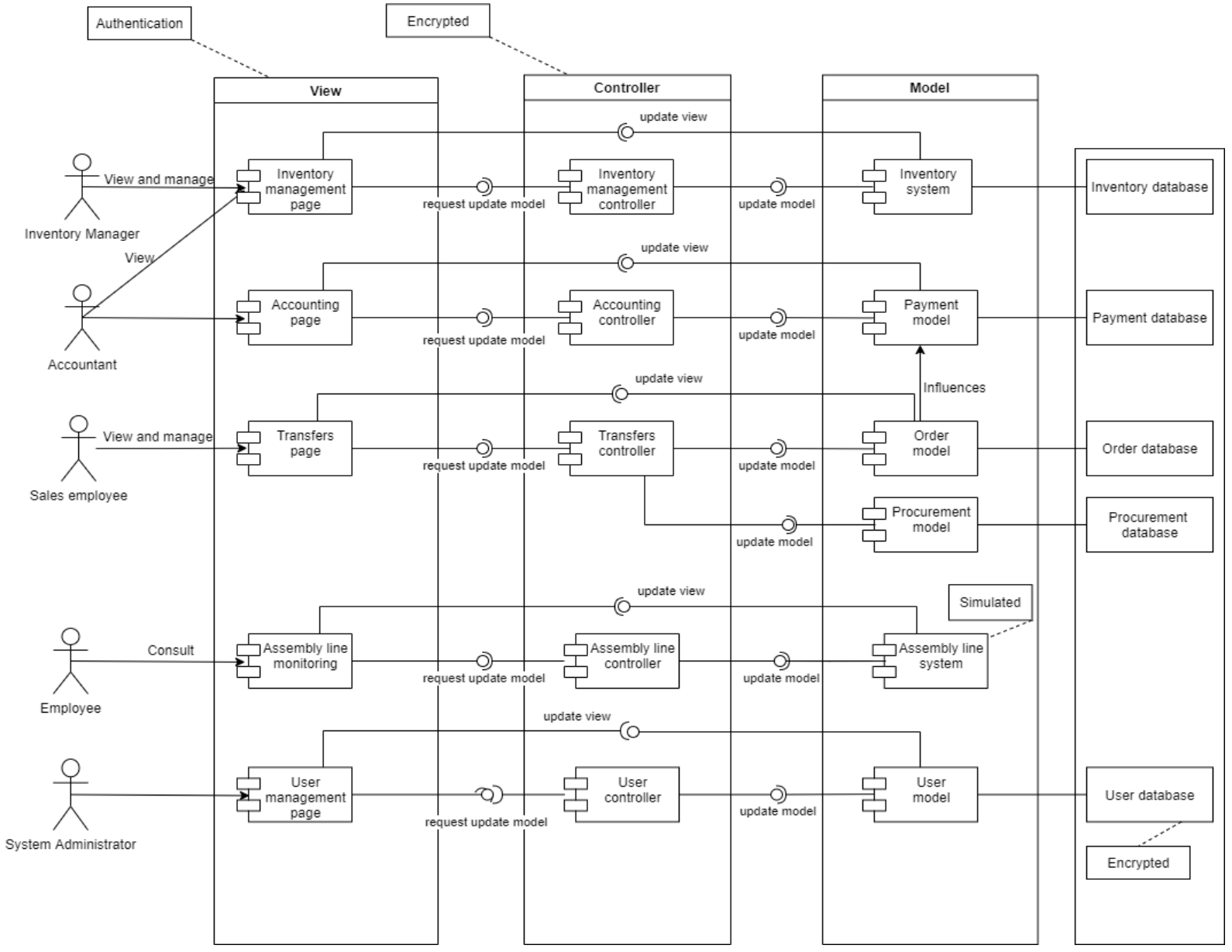

Figure 10: Component diagram (Version 3)

4.2.1.2 Known issues with view

The component diagram in the development view is not necessarily aiming at showing user interactions like it is in a context model (see section 3.1.3.1), so it does not always represent the needed actor environment(s).

Because the component diagram is used to represent the development view, it doesn't really have great value for the product owner, although it will help take care of their concern for the system's usability (see section 2.2.4).

4.3 Process view

The process views deal with the dynamic aspects of the system and focuses on the run time behavior of the system. The process view consists of 3 use case diagrams for the implemented features.

4.3.1 Models

Use case diagrams represent the possible actions actors can make on the system. They are useful to understand what process users go through when using a system feature.

4.3.1.1 Transfer Management Diagram

The transfers view utilizes the use case diagram model in order to show which actions are available to the sales employee when making changes to the database. This model comes in handy as it accurately described the interaction between a sales employee and the ERP system.

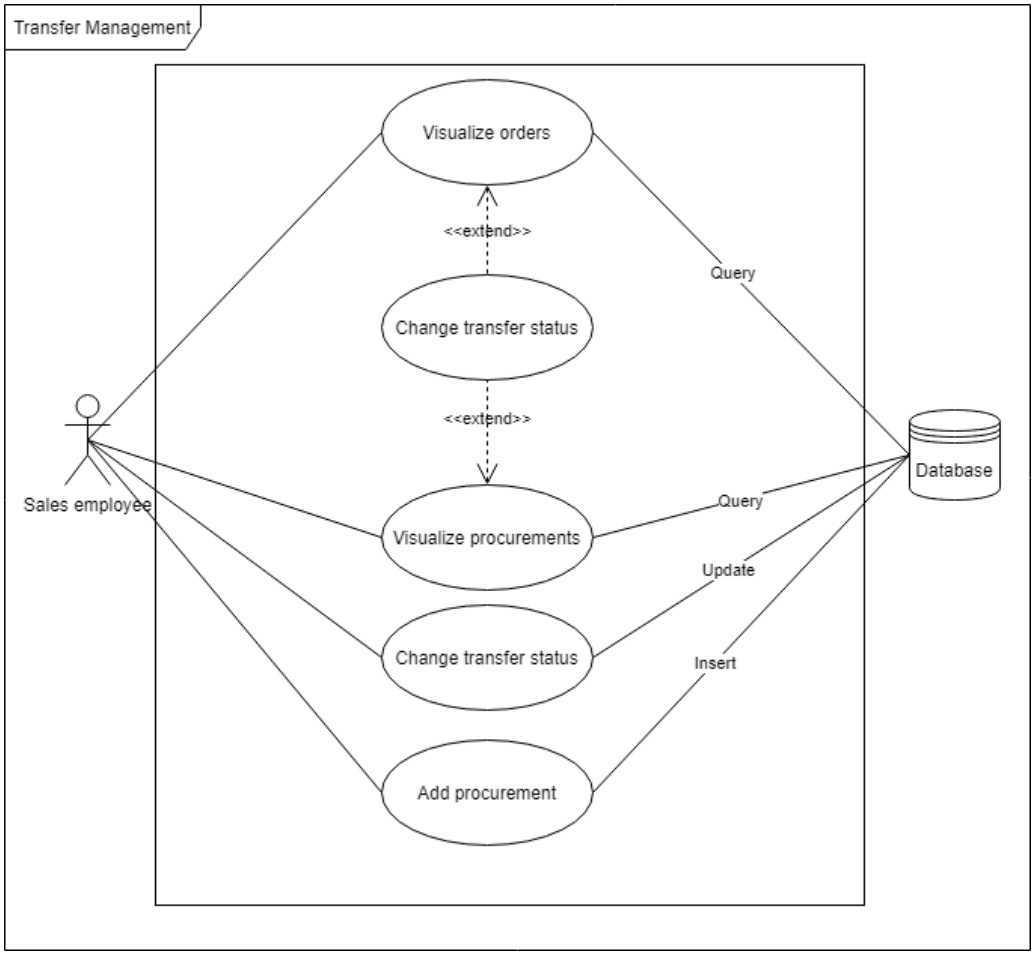

Figure 11: Transfer Management diagram (Version 1)

4.3.1.2 User Management Diagram

The user management view describes the interaction between an employee, and administrator and the authentication process. The authentication process requires two things in order to work properly, Firebase Authentication, which will verify the database in order to check if the user exists, and the database itself which stores the information of employees and administrator. It is best represented by a use case diagram. This is what we have chosen since it allows for a very straightforward representation of the interactions between the different types of users and the components required for authentication.

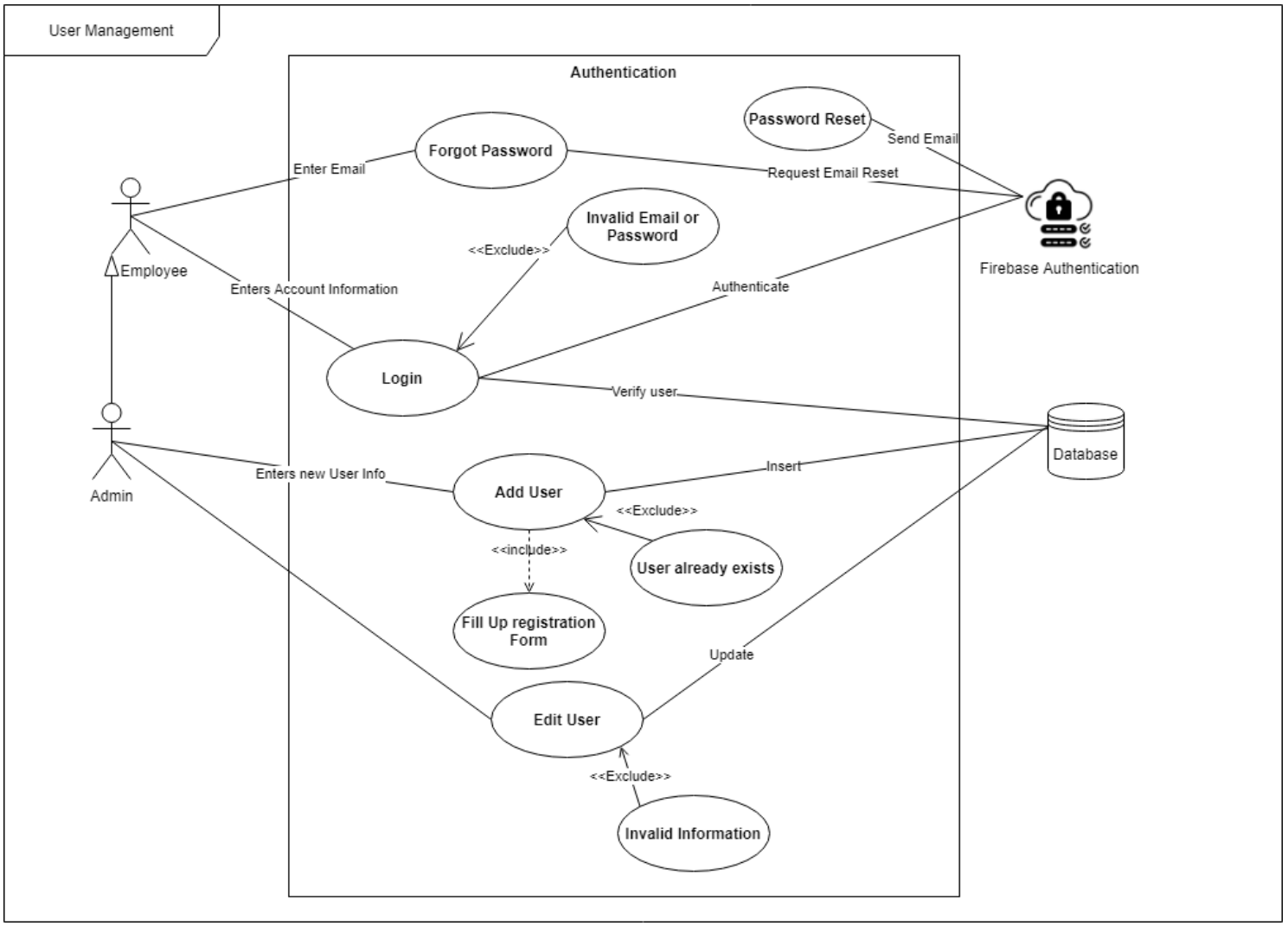

Figure 12: User Management diagram (Version 1)

4.3.1.2 Inventory Management Diagram

The inventory management view describes the interaction between an inventory management employee and the database. It lists all the possible things you can do as an inventory employee. Like the previous two views, the inventory management view utilizes the use case diagram model which is very useful to display the expected behavior of the inventory management process. On on side, you have the inventory employee, on the other you have the database with which the employee interacts with

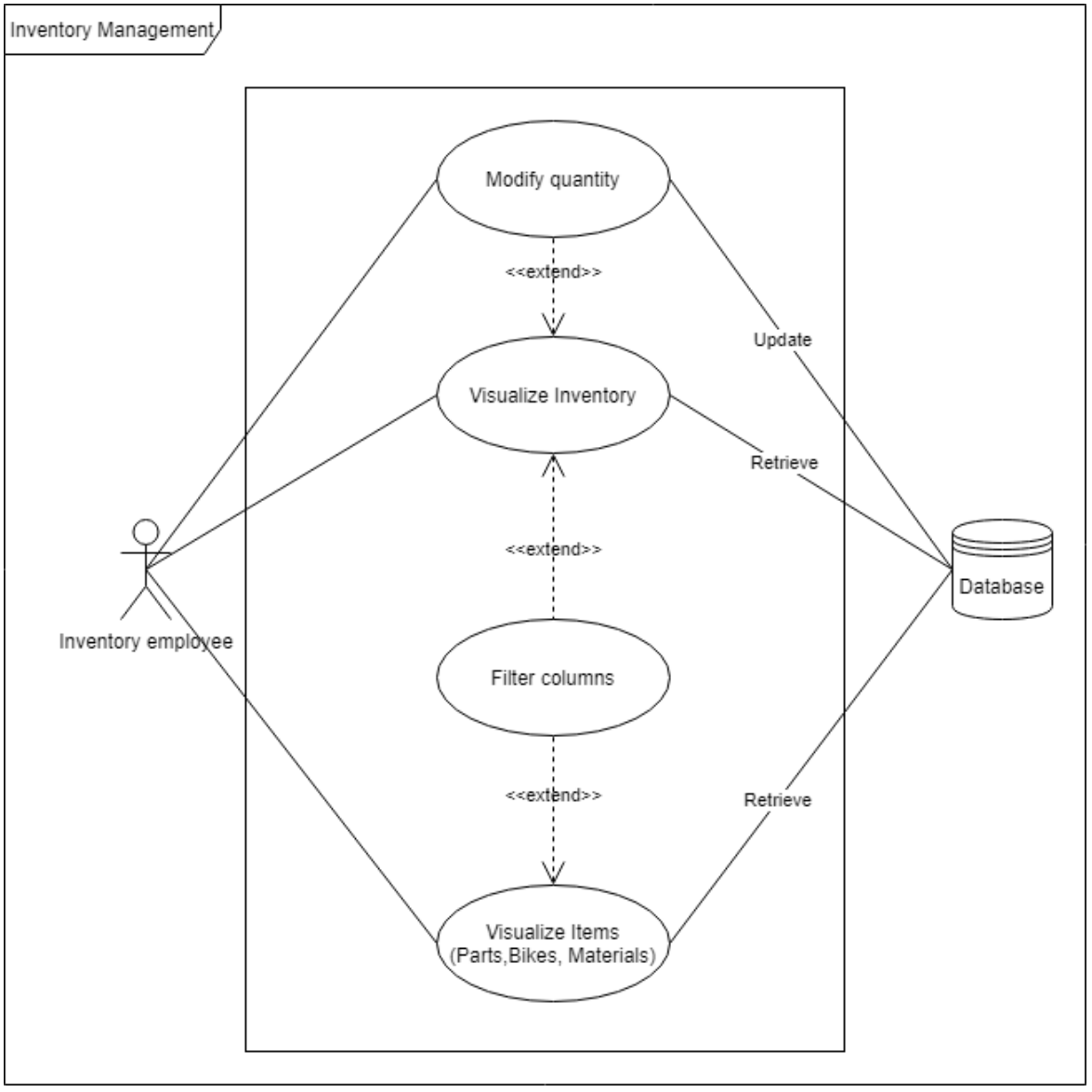

Figure 13: Inventory Management diagram (Version 1)

4.3.1.3 Accounting Diagram

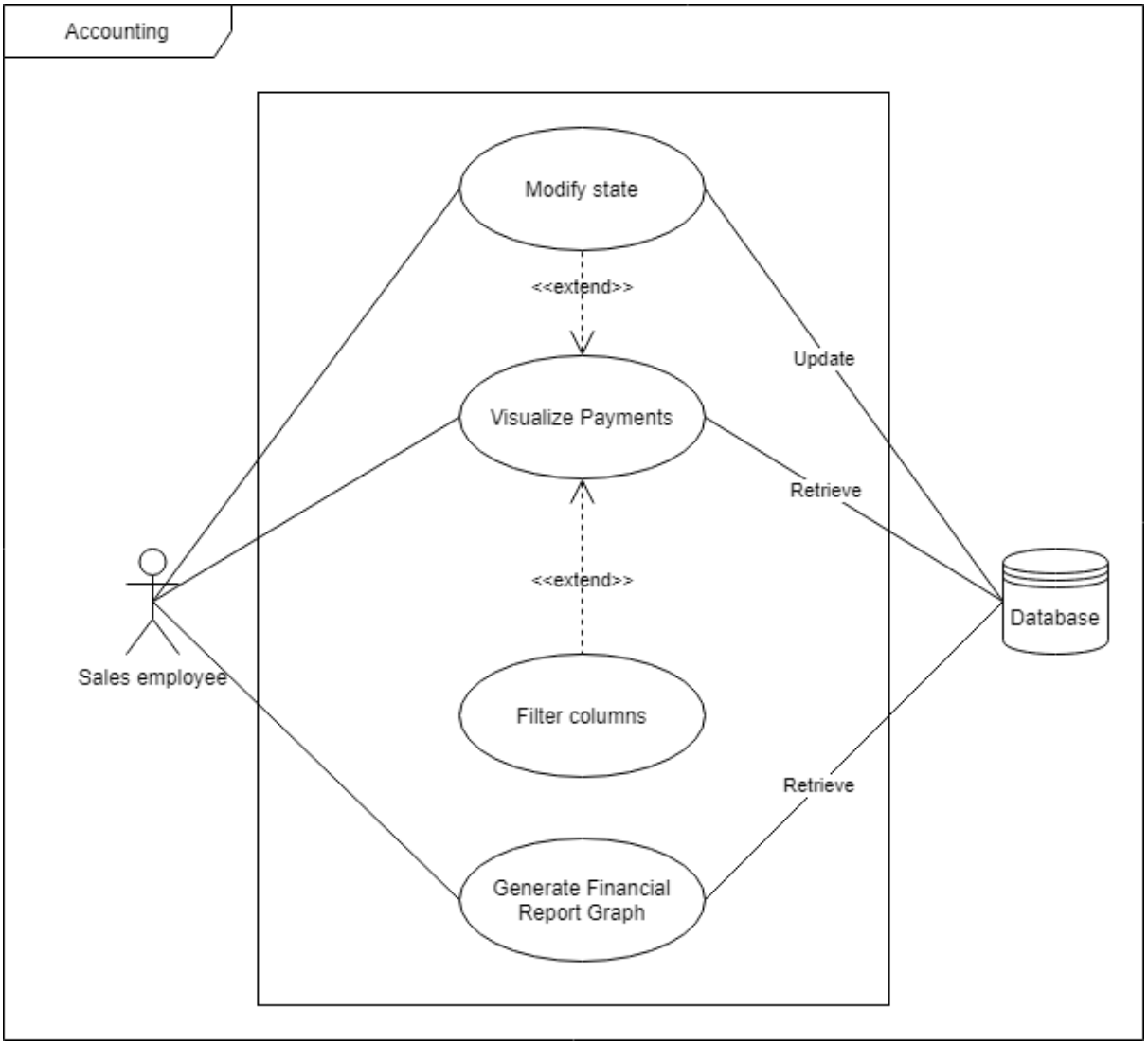

Figure 14: Accounting diagram (Version 1)

4.3.1.4 Assembly Diagram

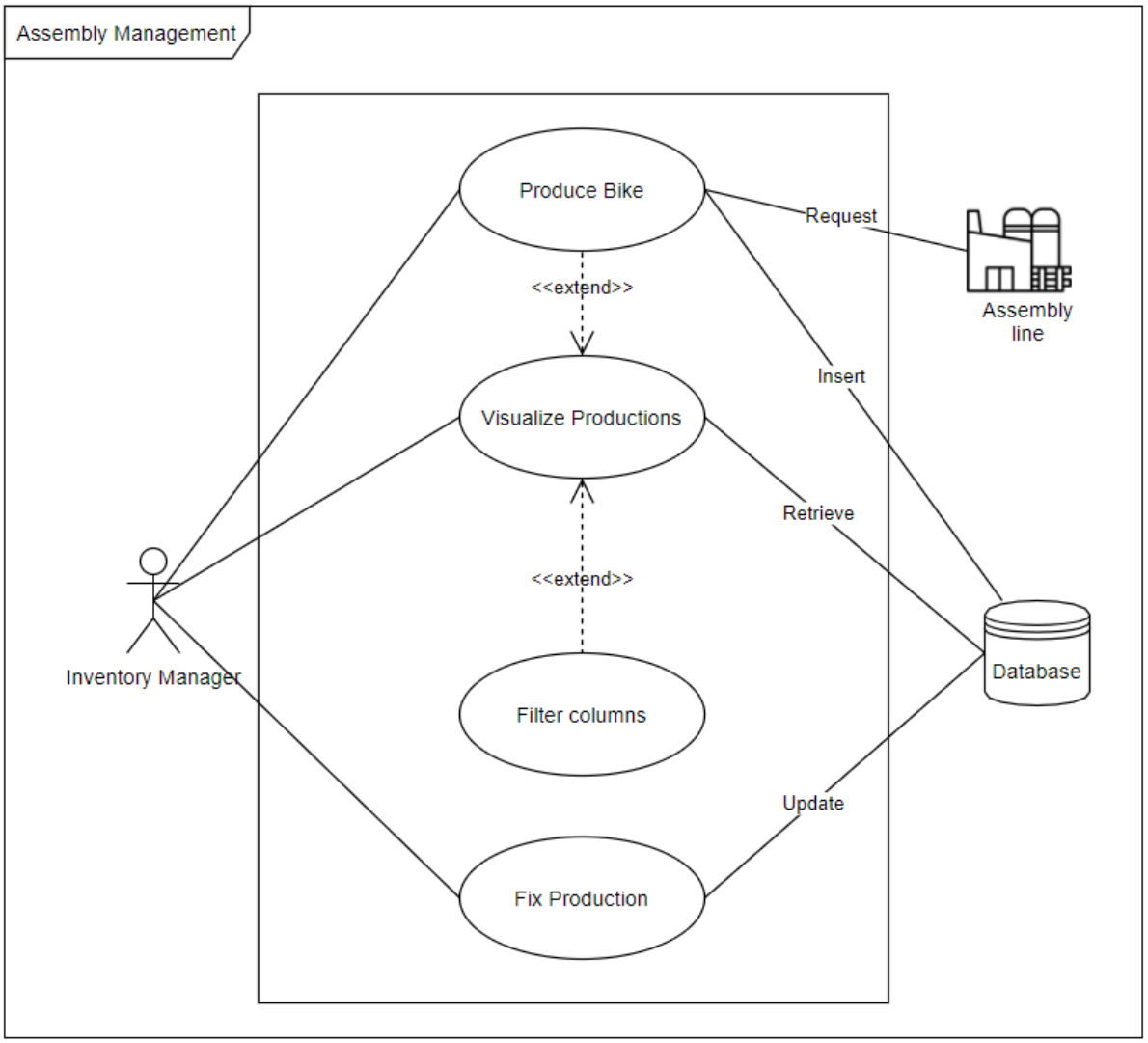

Figure 15: Assembly diagram (Version 1)

4.3.1.5 Triggers Diagram

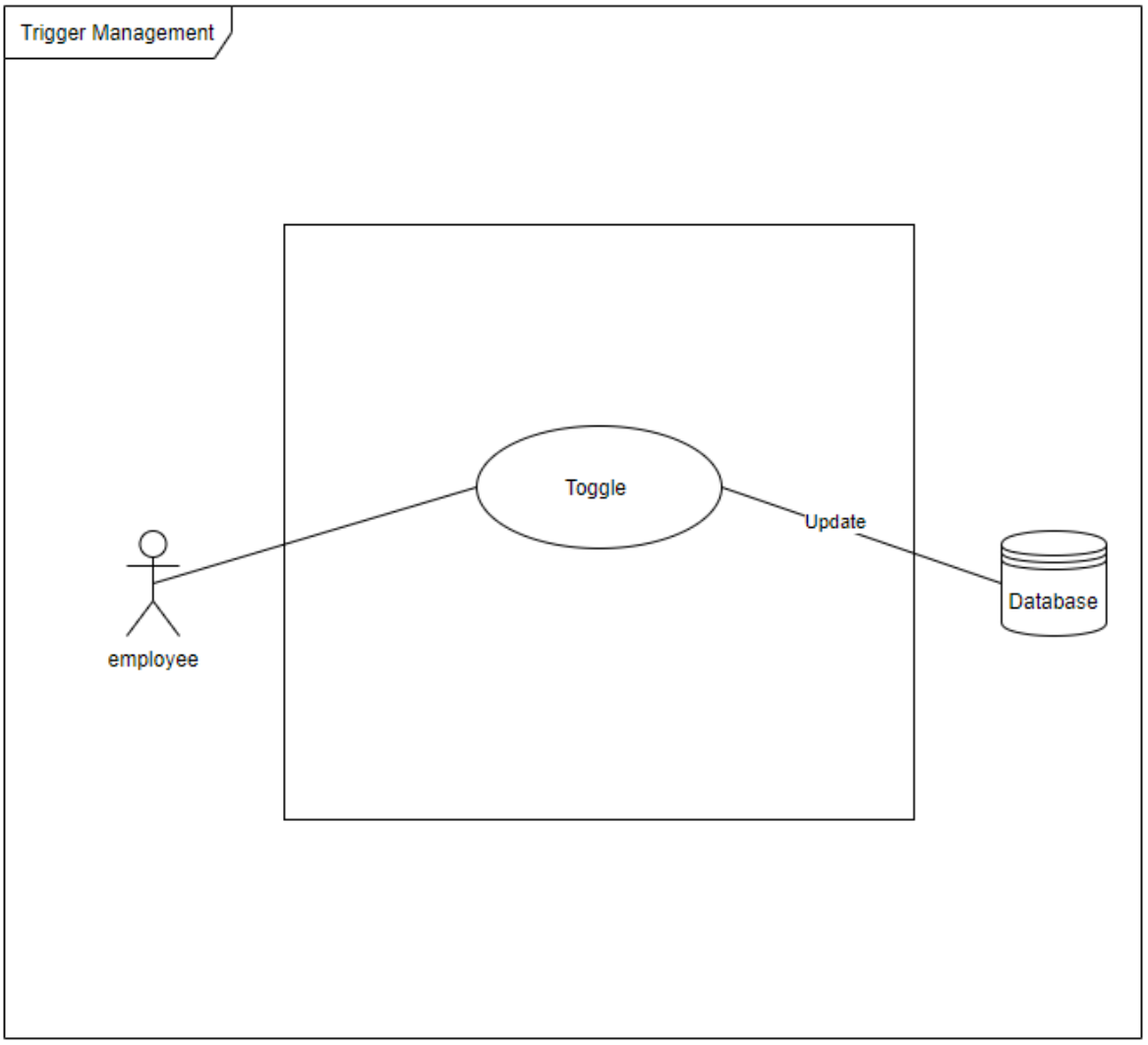

Figure 16: Triggers diagram (Version 1)

5. Consistency and correspondences

5.1 Known inconsistencies

* Record any known inconsistencies in the AD.

Although consistent ADs obviously are to be preferred, it is sometimes infeasible or impractical to resolve all inconsistencies for reasons of time, effort, or insufficient information.

∆ An architecture description should include an analysis of consistency of its architecture models and its views.

5.2 Correspondences in the AD

The AD elements used are models, views, viewpoints, concerns and stakeholders. Concerns are raised by stakeholders and formulated in viewpoints. Viewpoints set conventions for models that compose views. These views also adhere to the linked viewpoint. These correspondences are presented in Figure 13.

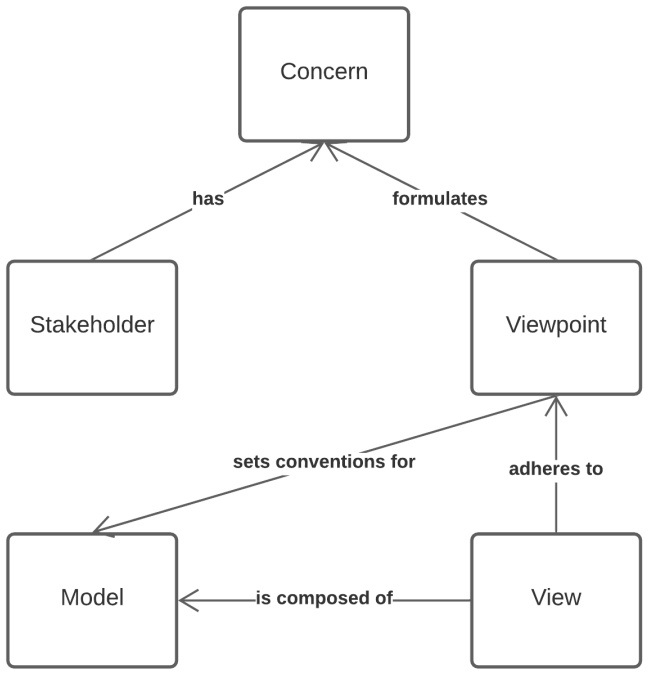

Figure 17: AD correspondences

5.3 Correspondence rules

In this AD, all diagrams are made using UML, so each diagram stays consistent with the language's notations, but the following rules should apply:

- All domain level concepts used in a diagram should be identified in the Domain Model (see section 4.1.1)

A. Architecture decisions and rationale

A.1 Decisions

A.1.1 View choices

- Because the functional requirements are not defined in this document or other ones produced by the team, the Use Case view was discarded, but may be present in the future when creating/refining more user stories.
- Considering the simplicity of the processes to be implemented in the first two sprints, the Process view was ignored.
- The Deployment view is not defined in this document, since the system's deployment plan is not yet defined.