Software Architecture Description of MVC for ERP System

Winter 2021

SOEN 390

Team 2

Contents

1 Introduction

1.1 Identifying information

The architecture used for this system and explained in this document is the MVC (Model-View-Controller) architecture.

The system is an ERP (Enterprise Resource Planning) for a bike company. The system shall be used to plan, manage and track resources and manufacturing of bikes.

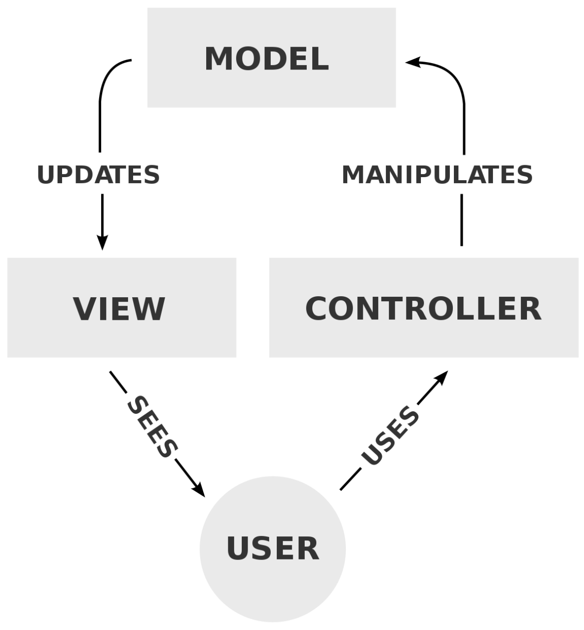


Figure 1 : MVC Overview

1.2 Supplementary information

Date of issue and status

- February 3, 2020, In progress

Authors, reviewers, approving authority, issuing organization;

- Authors: Ayman, Carlin, Mark, Adam, Muhammad

Change history;

N/A

Summary

N/A

Scope

- The scope of the ERP System involves the manufacturing process for bikes, sourcing from existing vendors in the system and only dealing with the data coming from predetermined production facilities.

Context

- The ERP System is designed for a bike company.

Glossary

N/A

Version control information

- Github is used as a version control source and the repository can be found on the following link: https://github.com/nuzurie/Enterprise-Resource-Planning

Configuration management information and references.

N/A

1.3 Other information

1.3.1 Overview (optional)

The MVC architecture is a widely used architecture. It provides modular and separated concerns design. It allows for rapid development and has low impact on the system when refactoring. It also allows multiple views for the same model and is SEO friendly.

1.3.2 Architecture evaluations

N/A

1.3.3 Rationale for key decisions

See section A.

2 Stakeholders and concerns

2.1 Stakeholders

The stakeholders include: Cyclists , Product Owner, Competitors, Development team, Business analysts, UI/UX designers, Marketing team, Sales team, Regulators, Operators, Factory employees, Service providers.

2.2 Concerns

The purpose of the ERP system is to provide a general visibility and control of the internal processes while manufacturing bikes. It provides tools to optimize the manufacturing process by tracking resources, inventory and providing accurate reports to the higher management and decision makers. Managers are concerned about providing high quality products and ensuring efficient production such that idle time is minimized, all the while being able to track orders, production and the inventory. Managers and accountants are the only ones using the system who need to see accounting information, they want to be the only ones who can view this information. In order to limit the ordering, inventory, production and accounting management operations per user role, there must be access rules.

The MVC architecture is suitable for such a system and fits perfectly for it. It has been used for many years and by many companies when designing their systems which makes it an easy and feasible system to design and deploy.

2.3 Concern–Stakeholder Traceability

Table 1: Association of stakeholders to concerns

Concern	**Manager**	**Accountant**	**Employee**	**Software Developer**
Tracking Inventory	X	X	X	
Inventory Management	X		X	
Tracking Orders	X	X	X	
Tracking Management	X	X	X	
Tracking Production	X		X	
Quality Management of Production	X		X	
Accounting Management	X	X		
Audit Trails	X			X
Reporting	X	X	X	X
Customizing Products	X			
Ordering Materials	X		X	

Communicate with Production Machinery	X		X	
Notification System	X	X	X	
Access Control	X			
Sensitive Data Encryption	X			
Usability	X	X	X	X
Performance	X	X	X	X
Modular				X
Mobile Friendly	X	X	X	
Low Latency	X	X	X	
User Friendly	X	X	X	

3 Viewpoints+

3.1 End User

3.1.2 Overview

The end user viewpoint corresponds with the logical view in this report. End-users are the ones who will use the ERP system as a service. The key features of the end-user viewpoint is to see the ERP system from the point of view of the customer, that is in terms of objects. The main systems that are pertinent to the end-user is order management, tracking, accounting, product management, inventory management and access roles.

3.1.3 Concerns and stakeholders

3.1.3.1 Concerns

- How can access roles be managed ?
- What happens when we are low in stock ?
- How can we track orders ?
- How can we receive data about the quality of production ?
- Where can we track operations ?
- How can we customize our products ?
- How can we be notified of products, inventory, or order events ?

3.1.3.2 Typical stakeholders

To name a few stakeholders, they include : client, manager, accountant, marketing team, and sales team

3.1.3.3 “Anti-concerns” (optional)

N/A

3.1.4 Model kinds+

3.1.4.1 Domain Model Diagram

3.1.4.1.1 Domain Model Diagram conventions

I) Model kind languages or notations (optional)

The Domain Diagram model format, content and notation can be found here :
https://homepage.cs.uiowa.edu/~tinelli/classes/022/Spring15/Notes/chap9.pdf

II) Model kind metamodel (optional)

N/A

III) Model kind templates (optional)

N/A

3.1.4.1.2 Domain Model Diagram operations (optional)

N/A

3.1.4.1.3 Domain Model Diagram correspondence rules

N/A

3.1.5 Operations on views

- Assembly actions (how to draw the diagram)
    - The developer shall verify the SAD and functional requirements to check the consistency of the specification. The developer can use a diagram creation application such as draw.io to create views. The components in rectangles represent real world concepts and objects. The lines and multiplicities follow the standard UML conventions.
- Check actions (how to read the diagram)
    - When looking at the diagram, start at the topmost component that says ERP System and follow the links downwards. For instance, the link from ERP System to

Department states : The ERP System has 4 main departments. Then moving downwards from Department to Vendor, it reads : The production department has 0 or more vendors.

- Viewpoint actions (how to understand the diagram)
    - The developer can look at Appendix 1 for the brainstorming done for the domain model and to understand its origins. The diagram essentially represents a high level idea of how the services and entities in the ERP system are to be related. The main services consist of inventory management, production management, accounting, and sales. The main stakeholders involved are managers, accountants, regular employees, vendors, manufacturers and customers. The ERP system will track raw materials, components and the full product which in our case is a customized bike.
- Guide actions (how to go from diagram to code)
    - When converting this diagram to code, the ERP system is the all encompassing application, the departments shall be services that allows for communication between the data and the user of the application. The users, vendors, and products will be modeled for the database.

3.1.6 Correspondence rules

N/A

3.1.7 Examples (optional)

N/A

3.1.8 Notes (optional)

N/A

3.1.9 Sources

N/A

3.2 Developers

3.2.1 Overview

The developer viewpoint corresponds with the development view in this report. developers are the ones who will design and build the ERP system. The key features of the developer viewpoint is to see the ERP system terms of actual source code packages, files, services and database. The main systems constraints in the development view is that it needs to follow the MVC architecture. The main services should be related to inventory, orders, accounting, reporting, tracking, product customization, and quality management.

3.2.2 Concerns and stakeholders

3.2.2.1 Concerns

- How will we handle data persistence ?
- How will we keep the project modular ?
- How will the source code conform to the MVC architecture ?

3.2.2.2 Typical stakeholders

The stakeholders concerned with this viewpoint include developers of the system.

3.2.2.3 “Anti-concerns” (optional)

N/A

3.2.3 Model kinds+

3.2.3.1 Component Diagram

3.2.3.2 Component Diagram conventions

I) Model kind languages or notations (optional)

The component diagram for this report follows the following conventions :
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

II) Model kind metamodel (optional)

N/A

III) Model kind templates (optional)

N/A

3.2.3.3 Component Diagram operations (optional)

N/A

3.2.3.4 Component Diagram correspondence rules

N/A

3.2.4 Operations on views

- Assembly actions (how to draw the diagram)
  - The developer shall verify the SAD and functional requirements to check the consistency of the specification. The developer can use a diagram creation

application such as draw.io to create views. The components in rectangles represent files, services and models. The dotted lines represent a dependency, the lollipop end means the component is providing an interface while the half moon end requires an interface.

- Check actions (how to read the diagram)
    - When looking at the diagram, start at the leftmost side where there is a stickman that represents the customer. Moving to the right, they interact with the login and are then presented with the option of interacting with different services. Let's go with the Inventory Overview, this page requires data from the inventory table in the database and uses the Inventory System to read, update, add, or delete data, then the view is updated based on what happens in the database.
- Viewpoint actions (how to understand the diagram)
    - The developer can look at Appendix 2 for the brainstorming done for the component diagram and to understand its origins. The diagram represents how the source code is to be packaged while adhering to the MVC architecture. Each view has a Service or System which then deals with the modeled data in the database. The view requests an action, the corresponding service accommodates this request and funs the desired functionality using the data from the database.
- Guide actions (how to go from diagram to code)
    - When converting this diagram to code, the files in the view are the UI pages, the Controller section will have classes corresponding to the services, the Model section will be classes that model the various objects. The database management system will be implemented using Spring's built in libraries.

3.1.5 Correspondence rules

The term Service is used interchangeably with System.

3.1.6 Examples (optional)

N/A

3.1.7 Notes (optional)

N/A

3.1.8 Source

N/A

4 Views+

The **Model-View-Controller (MVC)** framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Hence the abbreviation MVC. Each architecture component is built to handle specific development aspects of an application. MVC separates the business logic and presentation layer from each other. It was traditionally used for desktop graphical user interfaces (GUIs). Nowadays, MVC architecture has become popular for designing web applications as well as mobile apps.

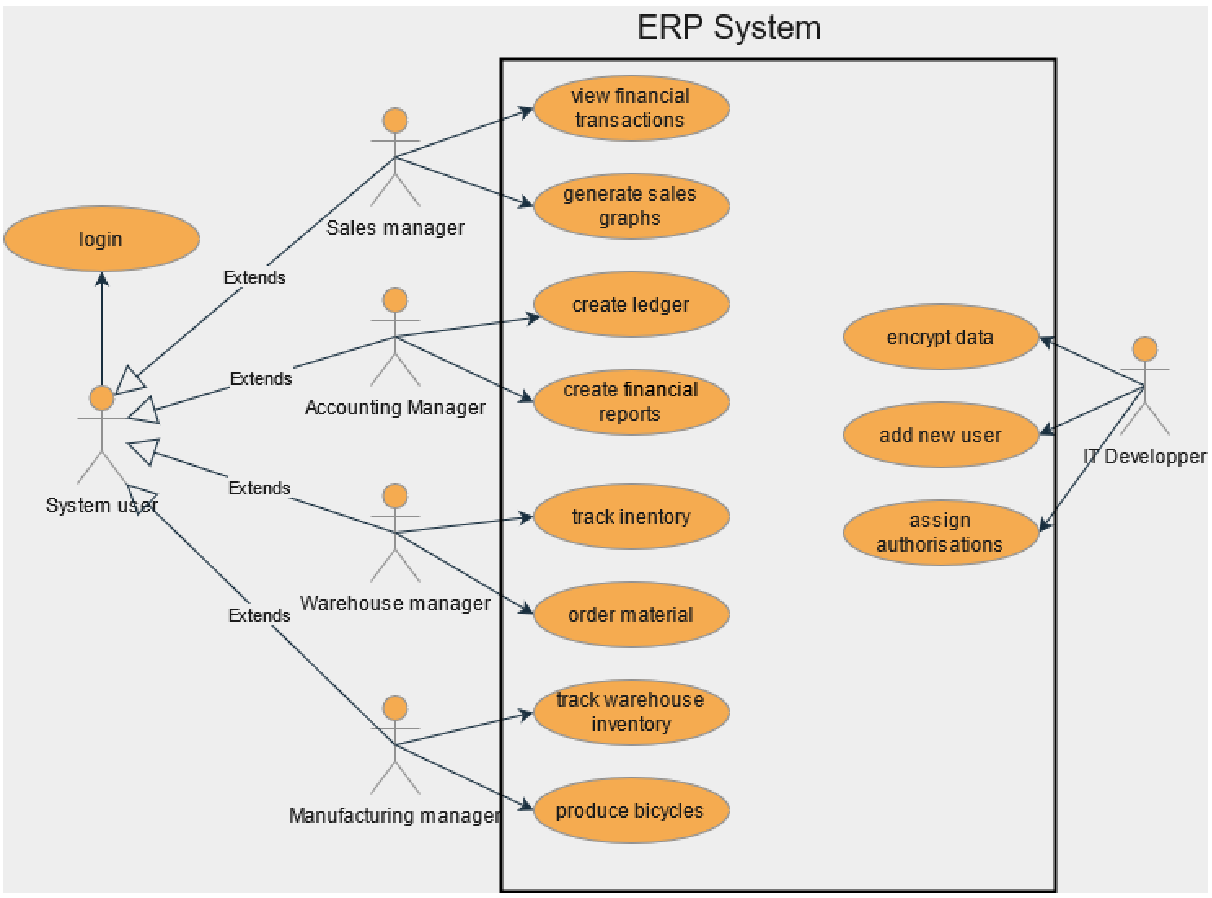
- Use-case/ Scenarios view: focus on the need that an end-user needs to derive from our system.
- Logical View: focus on the functional requirement of the application
- Process View: focus on the process, behavior, and the workflow of the application.
- Development View: focus on the management of the application.
- Physical View: focus on the hardware aspect related to our application

4.1 View: Use-Case View

The Use-Case view, also known as the Scenario view is a view that focuses generally on the point of view of the end-user with respect to the app. our ERP implement the follow user cases:

- Login
- Buy a bicycle
- Request shipping
- Request inventory check
- Inventory Check
- Restock inventory
- Make sub-part
- Produce bike
- Ship order
- Manage ledger

4.1.1 Use Case View Diagram:



4.1.2 Models descriptions :

1. Login : this use case allows any kind of user to log in the ERP system.
2. Buy a bicycle: this use case allow the customer to make a purchase of the bike
3. Request shipping: this use case is used by the manager to inform the worker about an inquiry of shipping.
4. Request inventory check: This use case allows the manager to ask the workers to check the inventory for missing items.
5. Buy sub-part: Use case that allows the customer to buy a sub part of the bicycle.
6. Restock inventory: use case destined for a worker user so that he restock the inventory.
7. Make a sub-part: use case that allows the worker to produce a sub part for a bike.
8. Produce bike: use case that allows the worker to assemble/produce a bike.
9. Ship order: use case that allows the worker to send the order of the customer which is either a sub-part or a bicycle.
10. Manage ledger: use case for the Account in order to manage company finances.

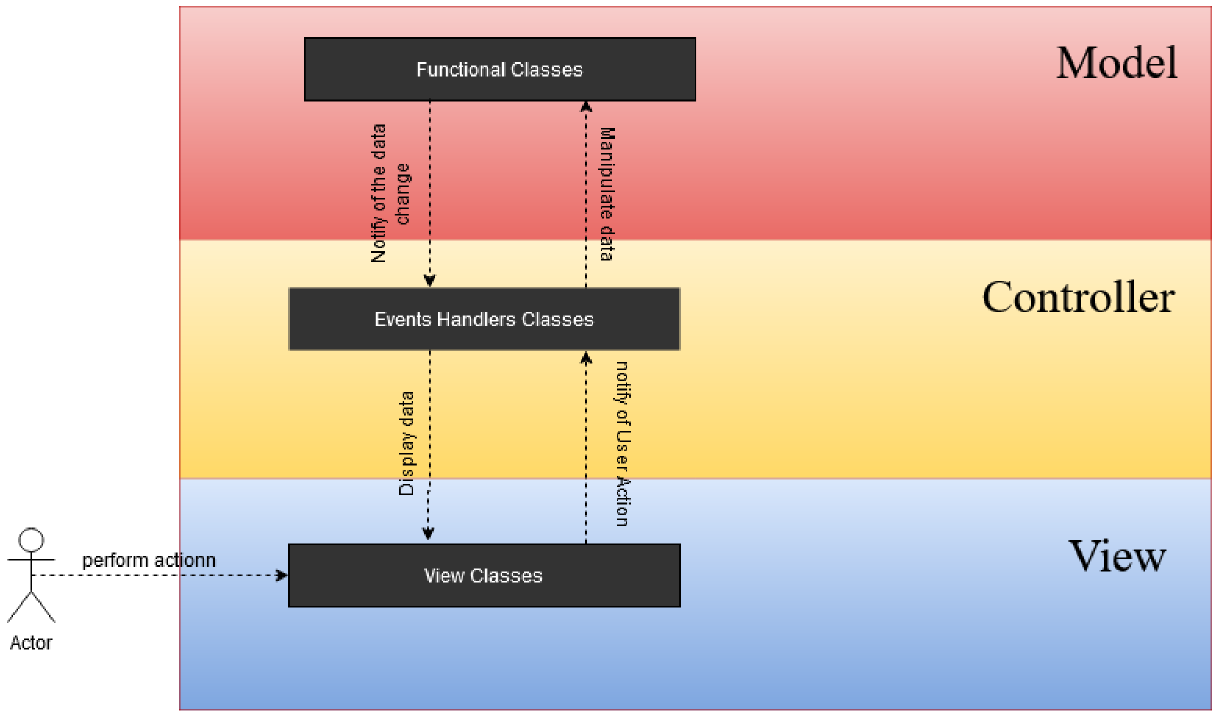
4.1.3 Known Issues with View:

The issue with the use Case view is that it does not capture non-functional requirements properly. For instance, it will address the needs of the user but not the efficiency to implement his needs. Therefore, satisfying a requirement in 1 sec or 10 min is still considered a success for the user case view point. Furthermore, user case view does not address the level of usability and usefulness of a use case requirement.

4.2 View: Logical View

The logical view defines the functionalities implemented in the system in order to satisfy the need of the user. The logical view is composed of the Model part which is the part responsible for the functionalities of our ERP system, it is also responsible for storing any part changes applied to the Data. The Controller part is responsible for handling actions performed by the user, and it is also responsible for notifying the user of the changes requested. The logical view has also a view part that is responsible for showing the user the functionalities implemented in the ERP system.

4.2.1 General Logical view Diagram:



4.2.1.1 Domain Diagram:

Link to Enlarged Domain model Diagram :

https://app.diagrams.net/#G156OE5vnPM5lSL6r0zZ1q4pgVGVHS6eHa

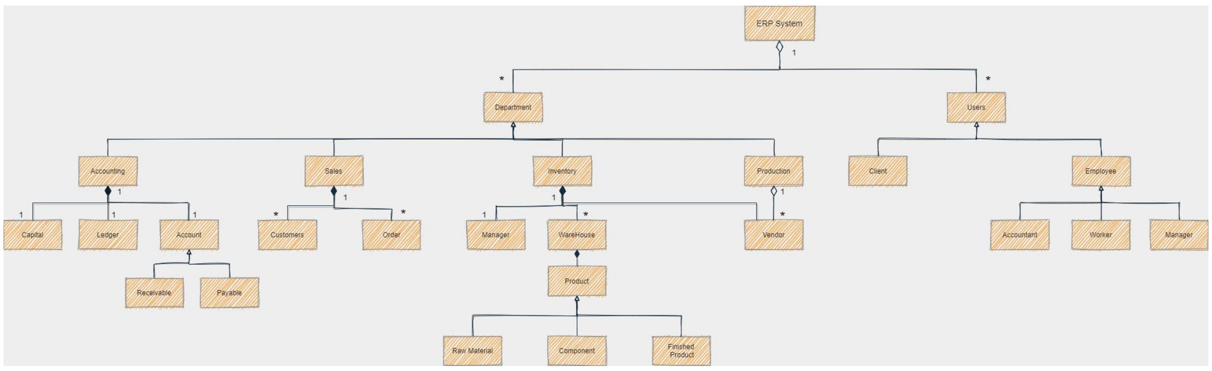


Figure 2 : Domain Model Diagram of ERP System

4.2.1.2 Class Diagram:

Due to its size we weren't able to add the class diagram into our document so we share a link of the draw.io link to enlarged our class diagram :
https://drive.google.com/file/d/1jeuLY5GzYsatZH7qhFqG-fs32px48C25/view?usp=sharing

In brief, the classes are enclosed with manufacturing, inventory and user packages. Each package has its own set of controllers, models, services, repositories and exceptions. The controllers are used to make REST API calls. Each controller has an instance of the associated repository and service. The repository instance is responsible for communicating with the database. The service is responsible for assembling the models when inserting or receiving data to or from the database.

4.2.2 Known Issues with View:

The issue with the logical view is that it is too abstract and does not cover the dataflow between object and class. The logical view does not cover the use case which is necessary to know if we reached the minimum requirement for the application.

4.3 View: Development View

The development view describes the organization of the software modules present in the system. The software is composed of multiple blocks, each containing libraries, subsystems or packages. This view requires a considerable amount of planning and design in order to ensure that each block is integrated properly into the system.

4.3.1 Development View Diagram:

Link to Enlarged Component Diagram :
https://drive.google.com/file/d/1ieWdnTgMDwGxN4kx-1LPjyrRmoXEoHs8/view?usp=sharing

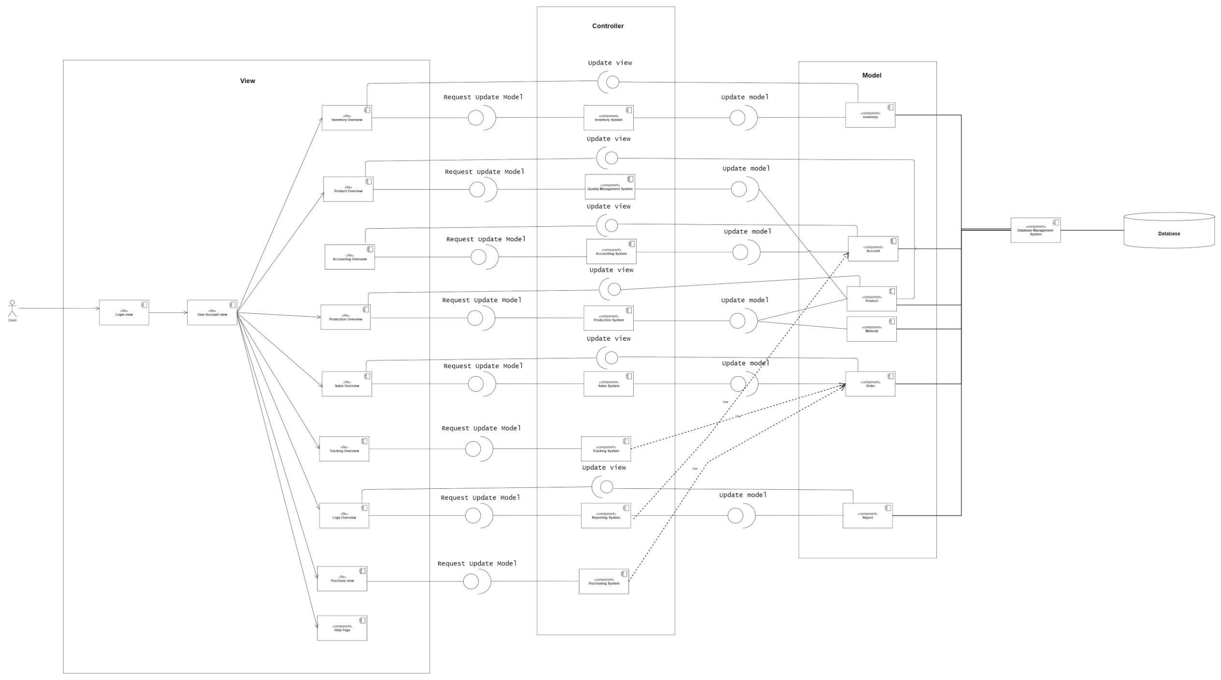


Figure 3 : Component Diagram of ERP System

4.3.2 Models Description

The development view is based on the Model-View-Controller architecture and is thus composed of several different components. The frontend represents the Views and is implemented with React. The various Views of the system allow the Users to interact with the system. The backend of the system is implemented using SpringBoot, which has good compatibility with the MVC architecture. The system will be using REST APIs to communicate between the backend and frontend.

The backend of the system contains 4 components: the controllers, the models, the database management system (DBMS) and the database.

Controllers: The role of the Controllers that are in the system is to react to the user's interactions with the Views and then send requests to the appropriate models to execute the instructions. In the ERP, the controllers handle requests for the main features, which include Inventory Management, Accounting, Sales, Tracking, Production, Quality Management, Purchasing and Reporting. Each of these Controllers communicates with a Model which will process the request.

Models: The role of the Models is to manage the business logic and the data of the system. The model can interact with both the views and the controllers of the system. The former can request information from the models, and the latter can send instructions to change its state. The ERP system is composed of 6 models.

Inventory Model: This model manages all the inventory data and updates the inventory view when prompted.

Account Model: This model manages the account data of the Users and updates the accounting view.

Product Model: The product model manages the different types of bikes that the company produces.

Material Model: The material model contains data about the different materials the company possesses.

Order Model: The order model contains information about the purchase of new materials and the tracking information.

Report Model: The report model is used to generate reports which can contain a variety of information pertaining to the activities of the company.

Database Management System: The role of the DBMS is to interact with the database and such as retrieve, add, remove or change information stored in the database.

Database: The Database stores all data relating to the ERP.

4.3.3 Known Issues with View

The main issue with this view is it provides too much low-level detail. The development view is supposed to provide an abstraction of the system that will be implemented and not the low-level details of the implementation itself.

5 Consistency and correspondences

Since our SAD contains many different views, viewpoints and models, it is a fair assumption to deduce that some inconsistencies will rise from the multiple aspects that we covered so far. Having also the tasks of producing multiple diagrams can contribute to this lack of consistency. This is mainly because multiple people can produce different diagrams of the same category (i.e 100 persons can make 100 different Use Case Diagrams).

5.1 Known inconsistencies

The inconsistency can be very difficult to avoid since our ERP system uses multiple languages, different classes for different purposes, models, and most importantly it includes different stakeholders points of view.

- Stakeholders have different priorities and different experiences so every stakeholder will give his own version of the requirement that must be implemented in each model.

- The languages used might differ depending on the development team's preference and their familiarity with the tools that are needed for the development process.
- Classes and their organizations are also subject to change depending on the preferences of the development team.
- Models are subject to change from one person to another because they are relying on inputs from stakeholders. Furthermore, different models have different ways of expressing some functionalities and features, and they can also omit some requirements from a model to another.

Possible Inconsistency Detection Techniques

Technique	Overlap Detection	Inconsistency Detection
Model Checking	elements with the same name.	use of specialized model checking algorithms
ViewPoint FrameWork	unification	Based on theorem proving
DealScribe	unification	stakeholders identify conflicts between "root requirements"goal monitoring
Synoptic	human inspection (visual aids are provided)	responsibility of stakeholders
Delugach	human inspection	based on comparison of conceptual graphs
Standard compliance	matching similar to standard unification	special algorithm for checking rules expressed in proprietary format

5.2 Correspondences in the AD

Logical and process Views:

The stakeholders of the logical view are the end-users, and the stakeholder of the process view are the system integrators. Despite the different stakeholders there are correspondences between these two views. The correspondence between these two views is based on concurrency. The logical view contains objects that could potentially be concurrent with other objects, but the degree of concurrency is not of great concern. On the other hand, the process view is only focused on the concurrency of the objects and not on the logic of the system. Thus, to achieve the appropriate level of concurrency for the system, both views must be taken into consideration. There are two approaches to determining the concurrency, the first is inside-out: which starts by considering the logical view, and the second is the outside-in: which considers the physical architecture of the system first.

Logical and Development Views:

The logical and development views are closely related but concern different things. The logical view is more focused on the functionalities of the system whereas the development view is concerned with the organization of the system. The difference between the two views is more observable in larger projects.

5.3 Correspondence rules

As seen before, the development process can be full of inconsistencies so in order to prevent this from happening we will follow a set of rules in order to minimize them.

- having brainstorming sessions between different stakeholders from different fields in order to consider different points of views.
- Models building should be done in groups so that everyone agrees on the same representation.
- For classes and coding we decided to follow coding standards to keep the workflow concise during every sprint.

A Architecture decisions and rationale

A.1 Decisions

Identifier	1
Related Requirements	The system shall be mobile friendly web (work well on any common screen resolutions).

Constraints	· Team members not familiar with mobile application development. · Time: A website would take less time to develop than a mobile application.
Alternative	Mobile Application
Decision	The ERP will be a website that is mobile friendly
Owner of Decision	Team decision
Rationale	· Team has more experience in web application development than mobile. · Can access ERP without downloading an application. · The ERP can be used on multiple different
Assumption	· Any device used to access the ERP has a browser. · The devices have access to the internet.

Identifier	2
Related Requirements	The system shall follow the MVC design pattern and at least 50% test coverage for Controllers classes.
Constraints	· Team members have more experience with MVC
Alternative	· MVP · MVVM

Decision	The ERP will be implemented using MVC
Owner of Decision	Team decision
Rationale	· MVC is widely used in the industry. · MVC allows for parallel development, some members can focus on the frontend while the others work on the backend. · MVC allows for multiple views
Assumption	· Each member of the team has knowledge/experience using MVC

Identifier	3
Related Requirements	· The system shall follow the MVC design pattern and at least 50% test coverage for Controllers classes. · The system should take less than 5 seconds to load (on localhost), and no more than 10 seconds in normal network condition (on the cloud).
Constraints	· Provide good performance. · Stable and dependable framework. · Cross-platform
Alternative	· Angular · Vuejs · Inferonjs · React

Decision	The Frontend of the system will be developed using React.
Owner of Decision	Frontend Team
Rationale	· Team has experience with React. · React can be used to create the views of the MVC. · React framework has good documentation. · React has a gentle learning curve.
Assumption	· The framework selected will have the MIT open-source license.

Identifier	4
Related Requirements	· The system shall follow the MVC design pattern and at least 50% test coverage for Controllers classes. · The system should take less than 5 seconds to load (on localhost), and no more than 10 seconds in normal network condition (on the cloud).
Constraints	· Provide good performance. · Stable and dependable framework · Cross-platform
Alternative	· SpringBoot · Django · .NET
Decision	The backend of the system will be developed using SpringBoot.

Owner of Decision	Backend Team
Rationale	· Some members of the team have experience with SpringBoot · All members of the team have experience with Java · Spring boot allows for quick setup and configuration of web applications · Spring Boot is frequently used in the industry thus has a large amount of resources and documentation
Assumption	· The framework will be free to use

Identifier	5
Related Requirements	· Sensitive data must be encrypted (credential details, personal information), so in case of a data leak, the leaked information cannot be traced back to any identity or user.
Constraints	· Provide good performance. · Store large quantity of data pertaining to the products and materials. · Cross-platform
Alternative	· MySQL · SQLite · PostgreSQL
Decision	The ERP will be implemented using MySQL as its DBMS.
Owner of Decision	Team decision

Rationale	· Some members of the team have experience with MySQL. · MySQL is a secure and reliable DBMS. · MySQL is relatively simple to get started with. · MySQL is the most popular DBMS and thus is well documented. · MySQL is one of the faster DBMSs
Assumption	· The DBMS needs to be open source

6 Design Patterns

The Spring Framework already uses several design patterns and so we will discuss how a few in particular were useful in our implementation. We will also discuss a design pattern that is in the process of being implemented.

Spring Framework

Singleton

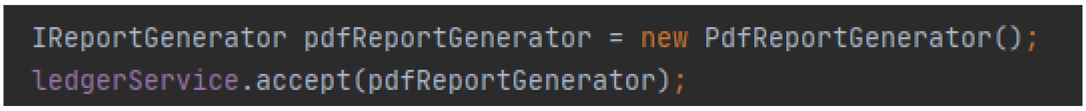
Generally, the Singleton pattern ensures that only one instance of an object exists per application. This comes in handy when we want to use the same instance of a repository amongst many controllers within a single application, the instance is easily injectable.

Builder

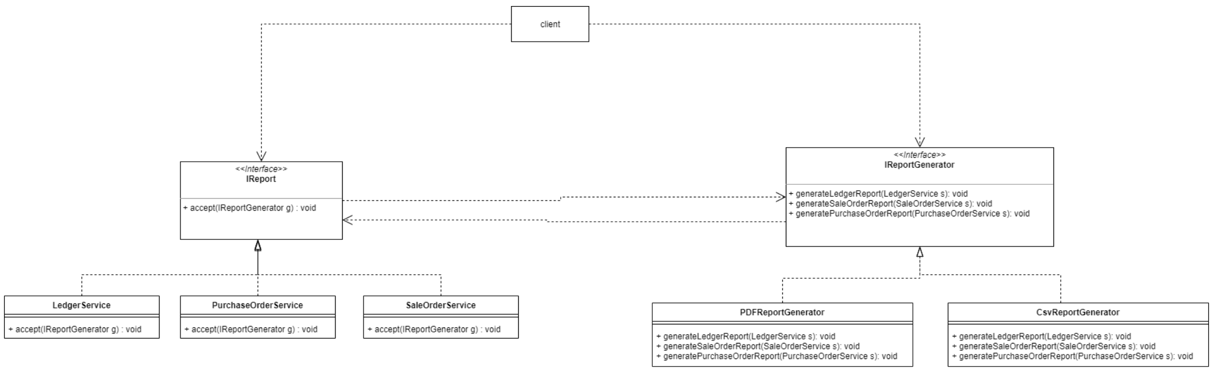
To overcome the issue of constructor telescoping, Spring addresses this issue using the builder design pattern using the Builder annotation. This relieves us from writing any boilerplate code.

Visitor Design Pattern

We used the visitor design pattern to implement different report formatting strategies for different report subjects. This is done so we don’t pollute the node classes; LedgerService, PurchaseOrderService, and SaleOrderService. In our case, the visitors were the services listed prior to this. So a service can accept a specific report generator like so :



```
IReportGenerator pdfReportGenerator = new PdfReportGenerator();
ledgerService.accept(pdfReportGenerator);
```


client
<<Interface>>
IReport
+ accept(IReportGenerator g) : void
<<Interface>>
IReportGenerator
+ generateLedgerReport(LedgerService s): void
+ generateSaleOrderReport(SaleOrderService s): void
+ generatePurchaseOrderReport(PurchaseOrderService s): void
LedgerService
+ accept(IReportGenerator g) : void
PurchaseOrderService
+ accept(IReportGenerator g) : void
SaleOrderService
+ accept(IReportGenerator g) : void
PDFReportGenerator
+ generateLedgerReport(LedgerService s): void
+ generateSaleOrderReport(SaleOrderService s): void
+ generatePurchaseOrderReport(PurchaseOrderService s): void
CsvReportGenerator
+ generateLedgerReport(LedgerService s): void
+ generateSaleOrderReport(SaleOrderService s): void
+ generatePurchaseOrderReport(PurchaseOrderService s): void

Class Diagram for Visitor Design Pattern Implementation

Link to enlarged class diagram of the visitor design pattern :
https://drive.google.com/file/d/1NHuP02kLD-W6D1Dp5r2iEr7-JDJvEkbj/view?usp=sharing

Abstract Factory

The bike class has several attributes, which are all subclasses of the Part class. A bike can have several parts associated with the bike, and to create those abstract factory pattern was used for this project. This allows us to easily exchange parts being used after creation and isolate their creation, particularly for attributes like accessories whose type is of Part.

Inversion Of Control and Dependency Injection

With Spring, we are afforded the option to hand over the object creation over to the framework or container, a form of Inversion of Control. A class may have several dependencies, for example our Controllers are often dependent on the Repositories. We chose to make use of Spring DI, allowing it to inject the correct Beans for the dependencies without specifying the implementation. Spring created these dependent objects for us, and manages their lifecycle.

7 SOLID Design Principles

Single Responsibility Principle

We have designed the classes such that each of them change for similar reasons. The controllers are only responsible for making a call to the database and then making a call to create a model. The Part class only has getters and setters related to its descriptive attributes. We separated the concrete parts because they have unique attributes that the Part should not be responsible of. The service classes are only responsible for creating models. Our classes are on average half a page, the most being a page long of code. This enforces modularity and enhances understandability and maintainability of the code.

Open Closed Principle

Our code is extensible without having to change existing code in the sense that Parts creation is highly customizable. We have preset parts as concrete classes, but if a user wants to create a new bike, part or material type, they are not restricted by what exists.

Liskov Substitution Principle

In our code, we mainly have inheritance for the Parts. It is the case that all the subclasses of Part can be treated the same as the parent class.

Inversion of Control

In Spring, the IoC container is a program that injects dependencies into an object making it ready for use as well as decoupling the object's dependencies. For instance, by using the @Service annotation, we can specify the desired configuration.

Dependency Injection

Spring uses dependency injection to implement IoC in applications.

Appendix

Appendix 1 : Domain Model brainstorm

Users:

- Employees:
    - Sellers
    - Manufacturers
    - Managers
- Clients:
    - Buyers

Products:

- Raw Materials (steel)
- Intermediate products (wheel)
- Finished product (bike)

Inventory dept:

- Has warehouse/inventory
    - Products:
        - Has products
        - Has Raw Material
        - Has Intermediate product
- Has a manager
- Has workers

Sales dept:

- Connected to inventory
- Customers
- Orders

Supplier.:

- Provides products
- Connected to inventory

Production dept.:

- Has workers
- Connected to the inventory

Accounting dept:

- Has a Capital
- Has a ledger
- Account
    - Accounts receivable
    - Accounts payable

Appendix 2 : Component Diagram Brainstorm

Models

- User (abstract)
    - Employee
        - Seller
        - Manufacturer
        - Manager
    - Client
        - Buyer / customer
- Product (abstract)
    - Raw material
    - Intermediate product
    - Final product
- Inventory (abstract)
    - Inventory item
- Account (abstract)
    - Account payable
    - Account receivable
- Order
- Material

Views (bundle as UI Files ?)

- Login
- Logout
- User account
- Product Overview
- Inventory overview, for tracking and material & product availabilities
- Accounting overview
- Production overview, Connect to production machinery
- Order overview, current orders, backlog, Create / define material list
- Edit material list
- Track overview
- Logs
- Help page

Controllers

- Order system / service
- Inventory system
- Sales management
- Production system
- Quality management system
- Purchasing system
- Tracking system
- Accounting system
- Reporting system
- DB management

Bibliography

[1] Paul C. Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. Documenting Software Architectures: views and beyond. Addison Wesley, 2nd edition, 2010.

[2] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. International Journal of Software Engineering and Knowledge Engineering, 2(1):31–57, March 1992.

[3] IEEE Std 1471, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, October 2000.

[4] ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description, December 2011.

[5] Alexander Ran. Ares conceptual framework for software architecture. In M. Jazayeri, A. Ran, and F. van der Linden, editors, Software Architecture for Product Families Principles and Practice, pages 1–29. Addison-Wesley, 2000.

[6] Nick Rozanski and Eo´in Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison Wesley, 2nd edition, 2011.

[7] Uwe van Heesch, Paris Avgeriou, and Rich Hilliard. A documentation framework for architecture decisions. The Journal of Systems & Software, 85(4):795–820, April 2012.