Architecture description template
for use with ISO/IEC/IEEE 42010:2011

Architecture Description of ERP-team12

Tyler Znoj 4005987
Killian Kelly 40014508
Abdul Sirawan 40074202
Adrien Kamran 40095393
Qandeel Arshad 40041524
Cedrik Dubois 40064475
Anthony Chraim 40091014
Matthew Giancola 40019131

Contents

Figure 2

Figure 2: Components diagram of the ERP

1. Introduction

1.1 Identifying information

ERP-Team12 is an Enterprise Resource Planning system for bike manufacturing. It integrates all of the manufacturing operations management (MOM) tasks into one software application. ERP-Team12 is an evolutionary application that is designated to assist any small/medium organization in executing their day-to-day tasks. ERP-Team12 ensures that tasks are automated and executed reliably to reduce cost and the chance of errors. The system consists of inter-connected services that exchange materials and information. The system supports planning and scheduling of operations. Also, the system facilitates procurements of goods and services, and distributing those services to clients or vendors. The system registers any financial transaction in the database and displays various financial reports upon request. Thus, each service is a system itself that is connected to other systems.

For example, users can plan operations to be executed, when to execute them, and by whom they will be executed. This feature offers users the ability of managing labor and tasks efficiently. Also, it offers some sense of version control or performance tracking of employees. Moreover, users can place orders for raw materials from various vendors, in case of shortage in a department. The system generates purchase requisitions, receipts, and bills. The system also supports shipping and tracking of items (finished products) to customers. The system has also an interactive tool that notifies customers or vendors regarding order updates/delivery or any other reason. The system is also capable of handling any financial task and it generates reports based on business performance, value or any other criteria.

1.2 Supplementary information

ERP-Team12 is an open-source application that will be launched by April 20. It is being developed by a team of 8 software developers. The development method that is adopted in this project is an agile development. Agile practices involve discovering requirements and developing solutions through the collaborative effort of self-organizing and cross-functional teams and their customers. This method promotes a quicker adaptation to changes in requirements. Pair programming and peer-review is also a notion that is being adopted in the development process of this project. The software code of this project is being tracked and managed by the version control tool git. The link to the project repository is attached below.

https://github.com/AdrienKamran/soen390-team12

1.3 Other information

1.3.1 Overview

ERP-Team12 is an Enterprise Resource Planning system for bike manufacturing. It integrates all of the manufacturing operations management (MOM) tasks into one software application. ERP-Team12 is an evolutionary application that is designed to support and integrate almost every functional area of a business process such as procurement of goods and services, sale and distribution, finance, accountings, human resource, manufacturing, production planning, logistics & warehouse management.

1.3.2 Architecture evaluations

ERP-Team12 has not been evaluated extensively yet. However, the system at the moment supports the creation of users and authenticating their credentials. The system logs out the user when inactive, logs their activity on the application, and links to a cloud storage for data backup. Those implemented services have been tested and evaluated. Unit testing is the type of test used for this project.

1.3.3 Rationale for key decisions

Multiple meetings have been carried out to discuss the requirements, and the software technologies to be used in this project. Those meetings aimed to identify the necessary requirements that need to be developed for sprint1, and which should be left for later sprints. The creation of the login and registration page was a priority since it is the first thing users see when they navigate to ERP-Team12. Of course with that decision, the database environment must be set up and connected to the application.

Also, sprint1 consisted of lightweight tasks that are easy to implement without involving the core features of the application. This is because the team needs to be more comfortable and familiar with the software tools that are used. Also, setting up the environment for the project is a hard task and needs to have priority in the first sprint.

Moreover, after elaborate discussions between team members, it was decided that the project will be developed by Python and django as a framework. The rationale behind this decision is that the majority of team members are familiar with Python and django framework. Members' familiarity with the software tools speed up the development process and ensures that the work is reliable due to the high confidence of team members in their knowledge about the used tools. Also, Vue.js was discussed as a potential front-end framework among Angular and React, but it was decided that Bootstrap Studio is sufficient for this project. The reason behind that, is again all team members are somehow familiar with bootstrap, but none of them is familiar with Vue.js. Also, Bootstrap studio has great features that allows the creation of many stylish templates without writing a single line of code.

2 Stakeholders and concerns

This chapter contains information items for stakeholders of the architecture, the stakeholders' concerns for that architecture, and the traceability of concerns to stakeholders. See also: ISO/IEC/IEEE 42010, 5.3

2.1 Stakeholders

Table 1. Description of the ERP-Team12 stakeholders with their descriptions and responsibilities

Name	Description	Responsibilities
Bike Manufacturers	The bike manufacturers are the organizations that will be using our services in order to execute their day-to-day tasks	These stakeholders will be our customers and the ones using our services. They will be the ones listing their needs for such a service and providing us with feedback in order to improve certain features
Users	The users will be the individuals creating accounts and navigating the ERP website to retrieve the relevant information they are looking for	These stakeholders will be the ones using the product on behalf of the bike manufacturer. They will be providing us with data and feedback on how to improve the navigation of the website
Software Developers	The software developers are the ones that will be developing the website	These stakeholders oversee the software development process. They will be implementing the features in the most time efficient ways, in adherence to the software requirements
Marketing Department	The marketing department will be the one promoting and advertising the product	These stakeholders will ensure that there is a market demand for this product. They will target specific manufacturers that are more likely to show an interest towards such a service
Project Manager	The project manager will oversee the whole project	This stakeholder will be planning the project, creating a schedule and a timeline, and managing the budget of the product
QA testers	The QA testers verify the quality of the deliverables	These stakeholders make sure that every single piece of software

		delivered does not contain bugs or errors. They also ensure that the system is maintainable and that the specifications of the software are always respected
System Architect	The system architect will design the architecture of the system	This stakeholder will create the right combination of components in order to achieve the desired product
Investors	The investors are the people investing money and/or resources into the project	These stakeholders ensure that the product meets its end goals and that they benefit financially from the returns of the finished products

2.2 Concerns

1. Security issues that might put some confidential data at risk
2. Universal healthcare and regulation
3. Insufficient funding for the project
4. Bad team cohesion which would result in bad teamwork and low progress
5. Conflict of interests
6. Conflicting priorities
7. Poor documentation of important concepts
8. Ambiguous or undefined scope

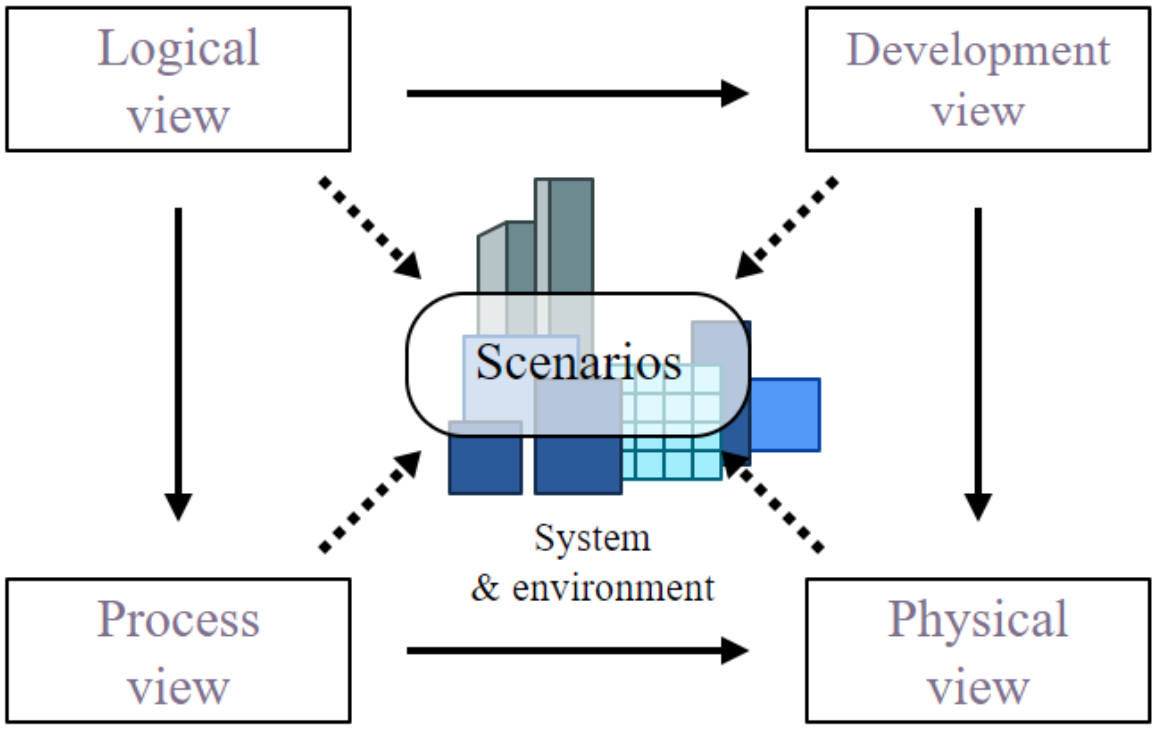
2.3 Concern–Stakeholder Traceability

Concern #	Bike Manufacturers	Users	Software Developers	Marketing Department	Project Manager	QA testers	Investors
Concern 1	x	x	x		x	x	x
Concern 2			x		x		
Concern 3	x			x	x		x
Concern 4	x		x		x		
Concern 5	x				x		x
Concern 6	x				x		

Concern 7		x	x		x		
Concern 8			x		x	x	

3 Viewpoints

In order to describe the architecture of our software system, Kruchten's 4+1 architecture view model will be used. Each view will be describing the system from the viewpoint of the different stakeholders mentioned above. The 4 views of this model are the logical view, the development view, the process view and the physical view. The scenarios (use cases) are perceived as the +1.



3.1 Logical View

The logical view is also known as the logical entity model, or the conceptual viewpoint, but will only be referred to as the logical view.

3.1.1 Overview

This view shows all of the functionality that the system will be able to provide to the end user. This viewpoint also describes the most important use-case realizations and business requirements of the system.

3.1.2 Concerns and stakeholders

The main stakeholder of this viewpoint will be the analysts and the designers. The logical view takes place early in the project (requirements analysis).The main concern of this viewpoint will be the functionality of the system. This viewpoint mainly focuses on the object/class decomposition of the system

3.1.2.1 Concerns

The first functionality concern will be that the technical alignment of the full solution is in fact compatible. This does not include any specific components, but rather how each component fits into another. These concerns could result in usability and performance concerns, which would be crucial for the project.

The second functionality concern would be the definition of the scope from the project manager. A poor scope definition could result in an incompatibility of the components.

The third functionality concern is poor documentation. If there is a lack of documentation and diagrams, following the software architecture could be harder than usual, misinterpreted, and even result in the failure of the system.

Another functionality concern is the maintainability of the system. The system architecture should take into considerations that some softwares can be deprecated and stop being compatible with other components of the system.

Since the analysts are the ones defining the requirements of the whole project, any type of negligence of miscalculation could greatly impact the whole project further in the development of the system.

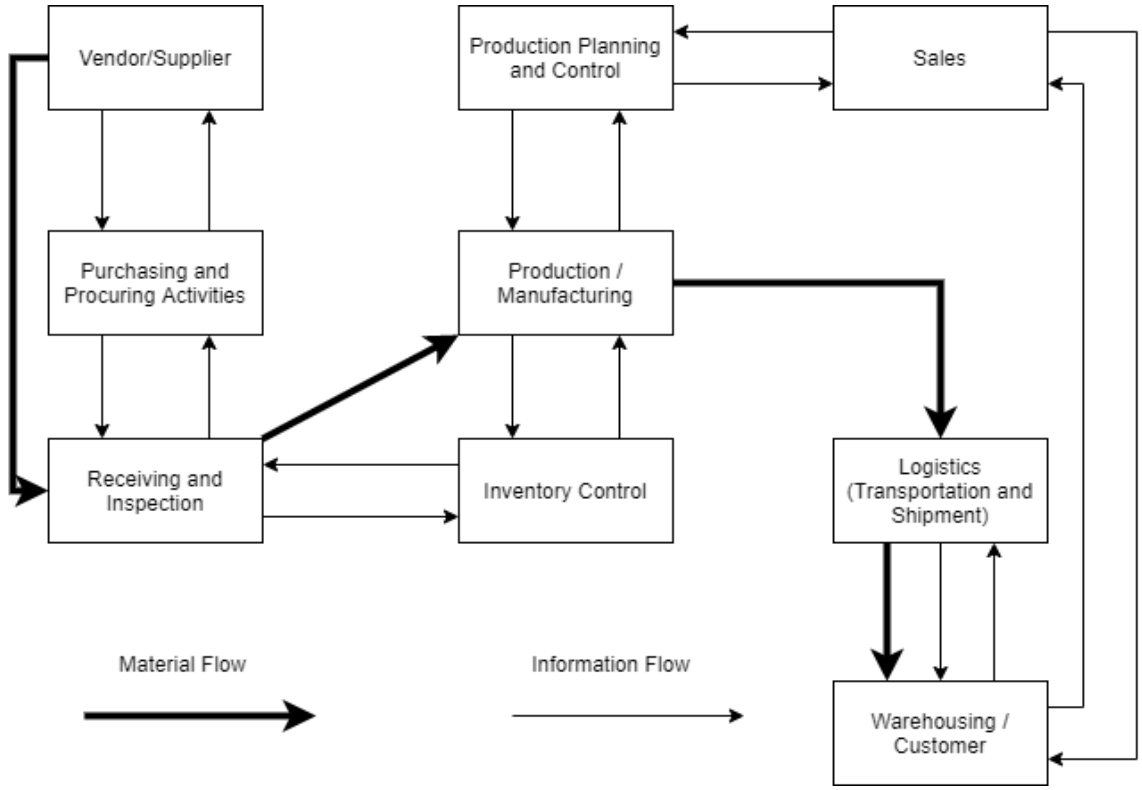
3.1.2.2 Typical stakeholders

The following list contains all the stakeholders relevant to the logical view;

- System Architects
- Analysts
- Designers

3.1.3 Domain model

Usually, relevant diagrams for the logical view are class diagrams, object diagrams and other kinds of composite structure diagrams. But for an ERP, a domain model diagram seems to be more applicable.



3.1.3.1 Domain model conventions

For this diagram, we decided to represent its components and in a high-level displaying the process that our application should perform. This model assisted us in understanding how the functional requirements relate to each other. Also, it shows how the material and information should flow from one entity to another. Later on, each entity will have its own model if its complexity level is high and requires further clarification.

I) Domain model diagram languages or notations

The model language used for this diagram is UML. Each rectangle represents a core feature of a manufacturing ERP system. The arrows represent relationships between the features. There are two types of arrows in the above diagram; thick and thin arrows. The thick arrow shows the direction in which the material flows, and the thin arrows shows the directions of the information flow.

3.1.2.3 Domain model diagram correspondence rules

In this diagram, the entities are linked together by two distinct types of arrows. Thick arrows represent material flow, and thin arrows represent the data flow.

3.1.4 Operations on views

Construction methods:

1. Researching ERP systems and understanding how they work
2. Analyzing the similar features of our application and other existent ERP software systems
3. Representing each core feature of the manufacturing process of the ERP system as an entity
4. Understanding how the material and data flow in the system and representing them

Interpretation methods:

The diagram is better understood if the reader starts with vendors/suppliers and traces the flow of materials. The material flows gives a very high-level overview of what is expected from the application.

Analysis methods:

This diagram is used at the early stages of the development to offer some guidance on the functionality of the ERP Systems. Comprehending all aspects of an ERP system is extremely challenging because of its various components. Thus, this model separates major functionalities of an ERP system and represents them independently. Each entity is connected with a data flow arrow. Tracing this arrow gives an idea of how the data should travel in the system.

Implementation methods:

This diagram is merely represented for understanding and simplifying the requirements of the system. However, each component must be analyzed separately and implemented, then that component should be able to send/receive data to the component it is connected to in the diagram.

3.1.5 Correspondence rules

In this diagram, the thick arrows correspond to material flow and the thin arrows correspond to data flow. Also, multiplicity of the arrows is one to many but since each two adjacent entities share two arrows, the multiplicity is considered many to many. However, it is represented this way for traceability purposes of the information and material flow.

3.2 Development View

The development view is also known as the implementation viewpoint but will only be referred to as the development view. This view, like the logical view, also focuses on the functional requirements of the system.

3.2.1 Overview

This view, like the logical view, also focuses on the functional requirements of the system. It takes place during the design stage of the project and focuses on the subsystem decomposition.

3.2.2 Concerns and stakeholders

The main stakeholder of this viewpoint will be the developers, the project manager, and the product manager. The main concern of this viewpoint will be the software management of the system.

3.2.2.1 Concerns

The first software management concern is the use of older/legacy systems. Since our product contains confidential information about bike manufacturers, these systems should not be used since they represent a huge risk when it comes to security. They do not receive updates to ensure that there are no security vulnerabilities and should be avoided.

The second software management concern is also related to older/legacy systems. There is a possibility that these systems stop being compatible with other technologies that are used in the project. The addition of new features would also be tricky if there are legacy systems in use since we would be less flexible while choosing the other systems.

The third software management concern is the difficulty to translate some of the more complex user level requirement into actual components

Another software management concern is that the approximation of the deliveries for certain features can be hard to do this early in the project. This can cause a disorganization for the devs and can lead to missed deadlines and incomplete features

3.2.2.2 Typical stakeholders

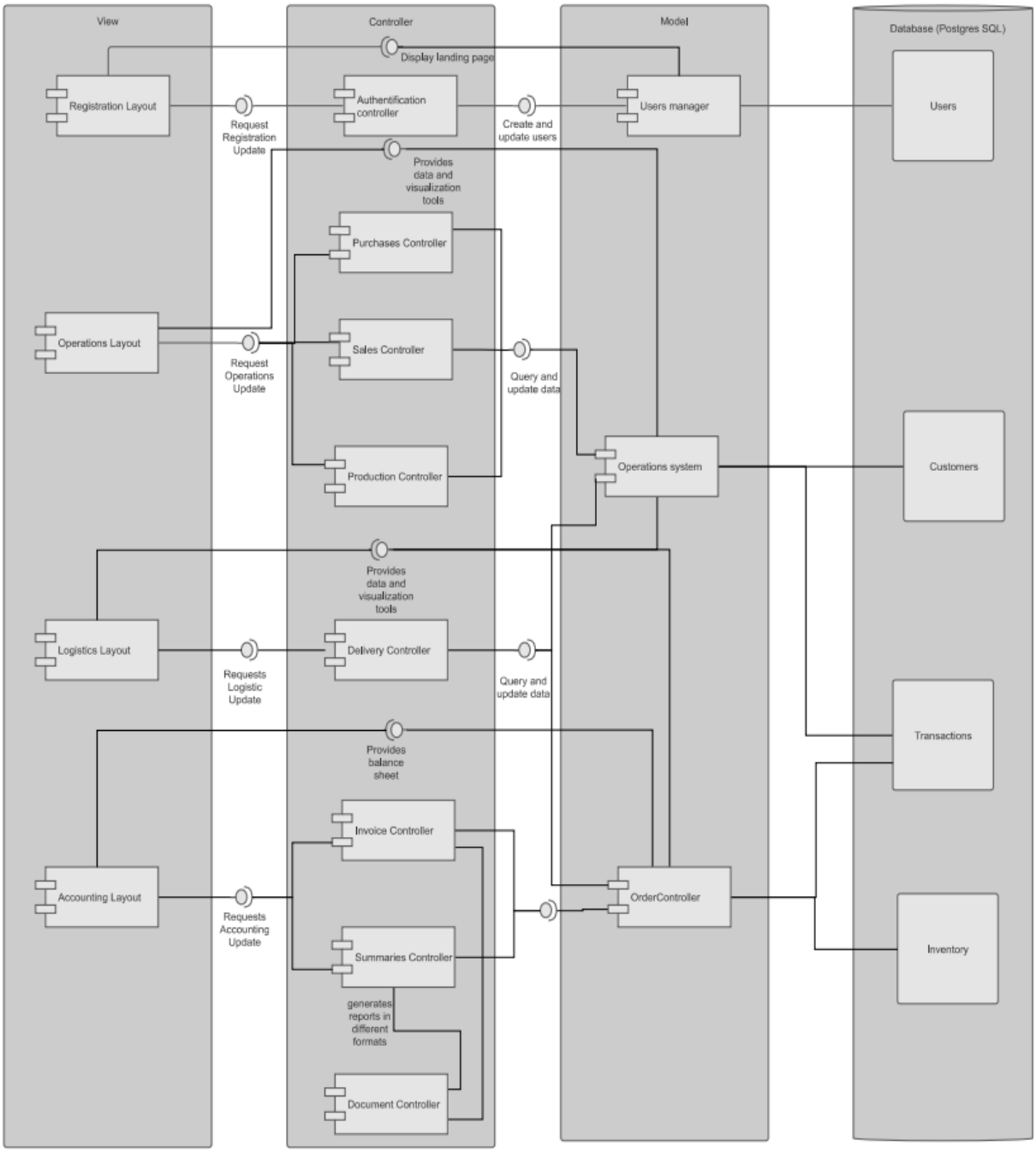
The following list contains all the stakeholders relevant to the logical view;

- Project Manager

- Developers

3.2.3 Component diagram

Relevant diagrams for the developer view are component diagrams and package diagrams. But since the project is still at the earlier stages and all the features are still not fully determined, only the component diagram will be presented.

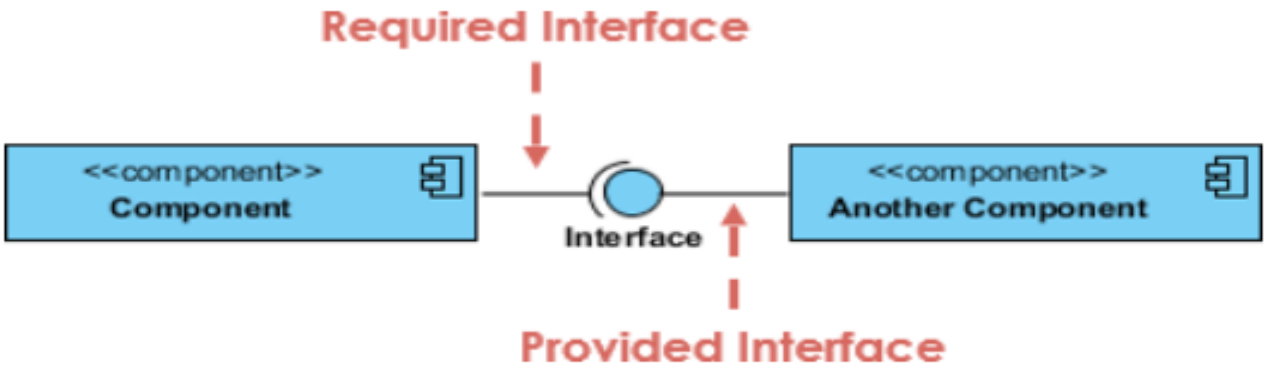


3.2.3.1 Component conventions

For this diagram, we decided to separate the components into multiple categories in order to follow the MVC structure. Every main page/layout of the website are placed as components of the view section. Each view component is connected to their one or more controller that communicates with the model that queries the database when necessary. When the view component requests an update, the controller component provides the data and visualization tools.

I) Component diagram languages or notations

The model language used for this diagram is UML. Each section of the MVC architecture is represented by the large rectangles with the dark grey background and the database is also the same size and has the same color background. The smaller rectangles represent the components of the systems. The symbols that link these components represent the interfaces.



3.2.2.3 Component diagram correspondence rules

The correspondence rules depends from a component to another but are all annotated

3.2.4 Operations on views

Construction methods:

1. Reading and understanding the requirements of the system
2. Deciding the technology that will be used
3. Build an erd for the database
4. Deciding the architecture the system will be build on (MVC)
5. Based on the architecture, translating the views, models and controllers to components
6. Determining how each controller will affect the view they are built for

7. Connecting the views with their respective controller and annotation how they will updating them
8. Connect the model to the controller and the database

Interpretation methods:
All of the dark rectangles represent a part of the MVC architecture. The components for the views, models, and controllers are then added to their respective category and linked to the other components that they influence. The database is added to the other components with entities that represent the tables names

Analysis methods:
When the SAD was first drafted, the view was consistent with the implementation of the features, therefore there is no need to alter the component diagram to be persistent with the actual system. Our predictions are that new UI views will be created further in the project and they will also be added to the component diagram with their corresponding controllers, models and database entities.

Implementation methods:
Every time a component is added, it will also be added to the system following the MVC architecture. This will cause no problems since the UI views are all independent of each other and the controllers depend on the views.

3.2.5 Correspondence rules

In the components diagrams, the correspondence between the UI views, the controllers and the models are all interfaces. One side requires an interface while the other provides it. The correspondence between the model section and the database means that the model will query the specific tables they are linked to in order to get the information needed.

Each UI view must have at least one controller.

3.3 Process View

The process view is also known as the behaviour viewpoint but will only be referred to as the process view. Contrary to the two views mentioned above, this view will be focusing on the non-functional requirements of the system

3.3.1 Overview

This viewpoint also takes place in the design stage of the project and will focus on the process decomposition. Its purpose is to make sure the system is scalable and performant. The run time behaviour of the system will be the main target of this viewpoint.

3.3.2 Concerns and stakeholders

The main stakeholders of this viewpoint are the system integrators, developers, and qa tester. They will be concerned about performance, scalability and throughput of the system

3.3.2.1 Concerns

The first concern will be about the scalability of the system. If the backend of the system is not able to adapt at the same speed as the users flood the website. This concern is not only caused by the sheer amount of users, but also by the amount of interaction these users have with the website. If a new feature is added and only tested with a few users at a time, this opens up the possibility that the servers will not be able to handle a bigger amount of people using this same feature at the same time.

The second concern will be about the performance of the system. While scalability concerns can cause performance issues, these issues can also have other sources. If the system relies on outdated technology it is more likely to encounter these problems.

The third concern will also be about the performance of the system. A bad database design can also lead to performance problems. If the database is not designed by following the third normal form (3NF), the duplication of data is more likely to happen which will make queries take longer

3.3.2.2 Typical stakeholders

The following list contains all the stakeholders relevant to the logical view;

- Project Manager
- Developers

3.4 Physical View

The physical view is also known as the deployment viewpoint but will only be referred to as the physical view. Just like the process view, this viewpoint will also be focusing on non-functional requirements

3.4.1 Overview

This viewpoint also takes place in the design stage of the project and will focus on mapping the software to the hardware. The physical layers and physical connections between the components will be the focus of the viewpoint

3.4.2 Concerns and stakeholders

The main stakeholders of this viewpoint are the system engineers. They will be concerned about the system topology, the delivery, the installations and the communication.

3.4.2.1 Concerns

Since our system does not have any physical parts, it will not have any concerns for this viewpoint

3.4.2.2 Typical stakeholders

The following list contains all the stakeholders relevant to the logical view;

- System engineers

3.4 Scenario View

The scenario view is also known as the use case viewpoint but will only be referred to as the scenario view. This viewpoint constitutes the +1 in kruchten's 4+1 architectural view model

3.4.1 Overview

This viewpoint will take place in all stages of the project, when putting everything together. It will focus on feature decomposition.

3.4.2 Concerns and stakeholders

The main stakeholders of this viewpoint are the end users and the bike manufacturers. They will be concerned about the understandability and the usability of the website

3.4.2.1 Concerns

The first concern is about understandability. If the system is not clear, concise, organized and complete, the end user will have a hard time using it. This is not the experience an ERP should provide to its users. The information should be found in a rapid manner and intuitively.

Another concern is that if the system is not understandable, the users will not immediately know if it is behaving as expected and won't know the difference of the normal flow of the website or if a bug has occurred.

Another concern is about usability. Since the ERP will be providing a huge set of data to the user, it is easy to get lost with the presentation of the data. If the user is not able to customize their experience, the usability of the website will be impacted. It is also difficult to organize usability tests for the system.

3.4.2.2 Typical stakeholders

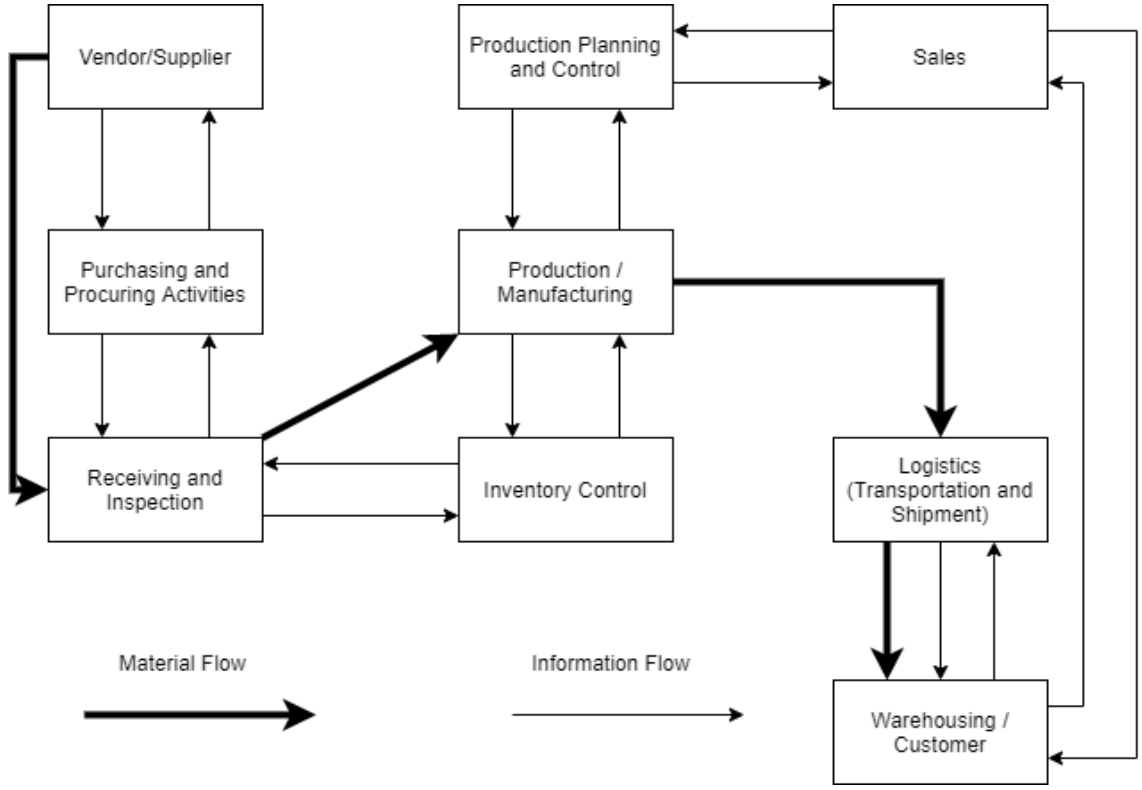
The following list contains all the stakeholders relevant to the logical view;

- Users
- Bike manufacturers

4 Views

In this section, we will be taking a closer look at the views corresponding at the viewpoints of the section above.

4.1 View: Domain Model Diagram



The above domain model is a great guide for implementing the core features of the project. It provides completeness to the requirement as it depicts how modules of the system should relate to each other. Functional and Non-functional requirements do not provide a thorough view of how modules relate to each other, but this view of the domain model certainly does.

This view is an abstract overview of the functionality of ERP systems. Later in the development process, each entity will have a view of its own to offer greater details of the functionality of the module.

4.1.1 Conceptual Model

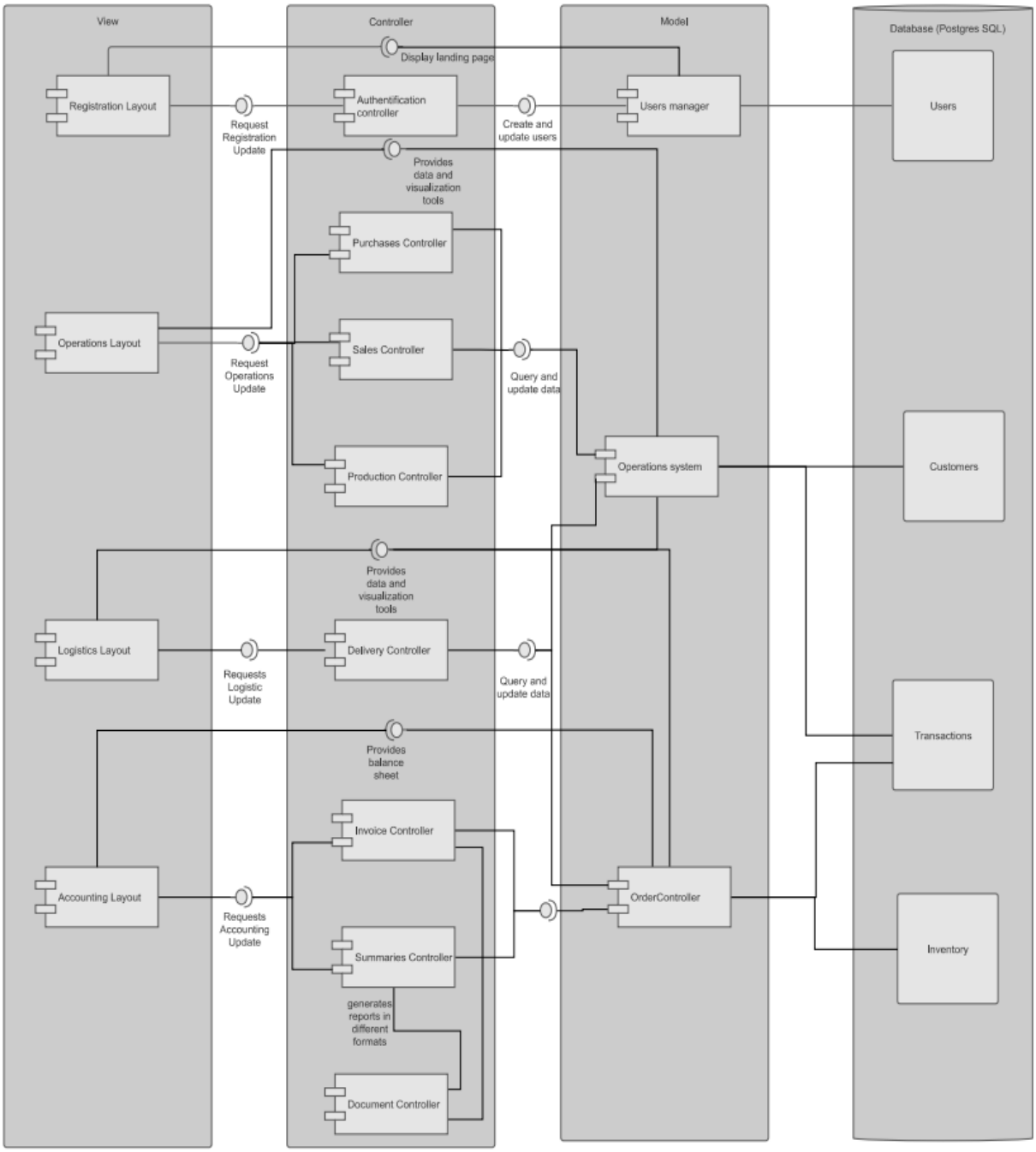
The architecture model of this view is conceptual. This model conceptualizes the operations that take place in a real-life ERP Systems. The conceptualization of this model is done by representing each department of the business model that is responsible for manufacturing operations in a separate entity. Then, the arrows come in play to demonstrate the material and data flow between those departments. Therefore, this model's mere job is to view how the data should flow at an abstraction level.

4.2.2 Known Issues with View

This view displays the interactions of the system's entities at a high-level. Thus, this limits the understanding of the details that are hidden inside the entities. Each entity represents a step in the manufacturing operations. However, those steps are complex and are composed of many processes and operations. Therefore, we need to perform a further analysis on each entity to study its data and material flow.

Also, the multiplicity of the arrows is many to one but they are used back and forth for some entities. This is typically not the ideal way to present a domain model, however, it is a decision that was made to simplify the tracing of data and material flow in the system.

4.2 View: Component diagram



The model language used for this diagram is UML. Each section of the MVC architecture is represented by the large rectangles with the dark grey background and the database is also the same size and has the same color background. The smaller rectangles represent the components of the systems. The symbols that link these components represent the interfaces.

The viewpoint governing this view is the development view is the development view

4.2.1 MVC

The architecture model of this view is MVC. The model is the most important component of this pattern. It manages the data from the database and the controllers of the systems while being independent from the UI views. The UI views will be the layout of the website pages that will be possible to navigate to for the bike ERP we are building. Finally, the controller will convert the UI inputs into commands for the view or the model. An architecture view is composed of one or more architecture models.

4.2.2 Known Issues with View

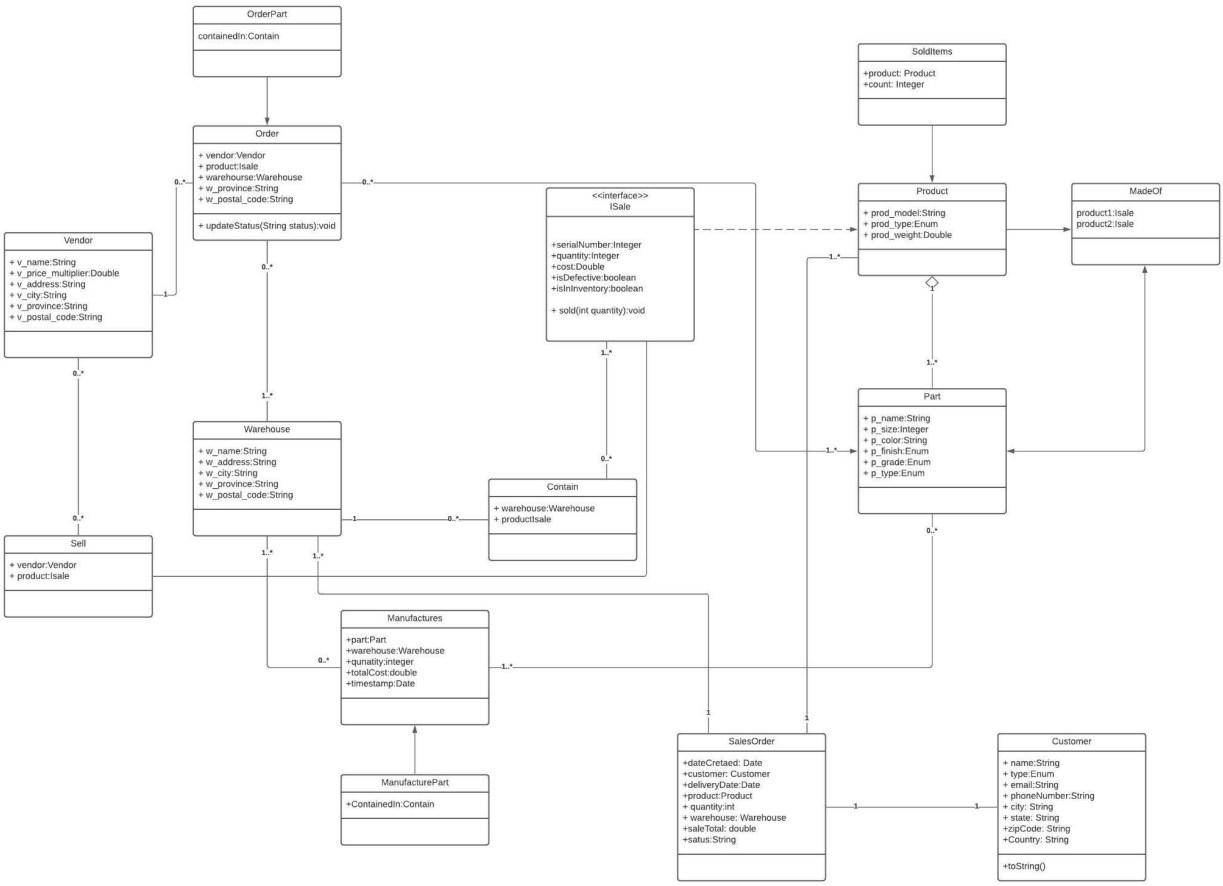
The viewpoint convention would require a package diagram as well as a component diagram. Since it is too early in the project to determine all the dependencies of the packages that will be used, only the component diagram was done for this viewpoint.

There are no known inconsistencies between the diagram and the system at the time of the writing of the software architecture document, but if any are discovered later on in the project, the component diagram will be altered to fix the issues.

The tables in the database sections are incomplete and will be added once the erd has incorporated all the desired additional features.

Building the component diagram with the MVC format is not too conventional but it seemed appropriate for the system in question

4.3 View: Class Diagram



The class diagram is expressed in the UML model language. It represents the structure of our system by showing its classes, the attributes of said classes and their operations, and the multiplicity and relation between these classes. This diagram provides a basic notation for the structure without imposing any coding/programming language to it. It is also useful for everyone in the team in order to understand the underlying system without having worked on every single part. The principles of this class diagram were respected when implementing the django models.

4.3.1 Logical View

The viewpoint governing this view is the logical view. Its purpose is to provide a view of the classes in the model. The application’s layout will be gathering information from the models (containing all the business logic), the urls, as well as the forms that will pass information concerning the user.

4.3.2 Known Issues with View

Since the system is built through the django framework, the logical view is based on the application's models. While the classes, interfaces, and relationships presented in the class diagram are relevant in the django models, they are not represented as accurately as they would have been on a system that integrates actual classes (Object Oriented).

Also, since functionalities will be added in the following sprints, some changes might arise and we might realize that the models we have are not suited for the new features. The class diagram will be changed as a consequence.

5 Consistency and correspondences

5.1 Known inconsistencies

In this section we outline all the inconsistencies between the different views and inside the architecture description. These inconsistencies are either calculated or are things that will need to be fixed in the future.

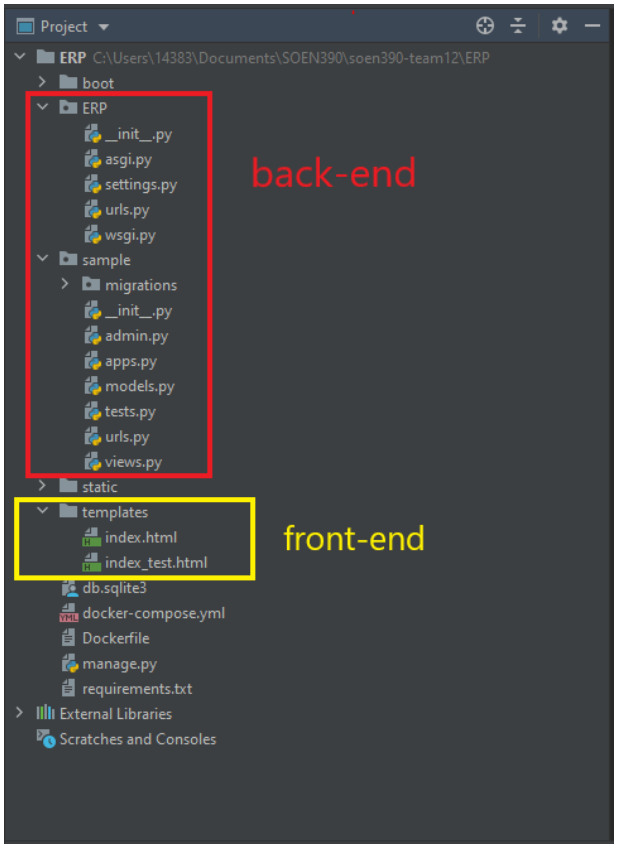
- Due to the early age of the project there are currently no views for the physical viewpoint.
- Due to the early age of the project there are currently no views for the process viewpoint.

5.2 Correspondences in the AD

Correspondences between the Logical View and the Development View:

Logical View Component	**Development View Component**
Vendor/Supplier	Users manager, stored in the Users table
Purchasing and procuring activities	Purchases and sales controller, stored in Customers and Transaction tables
Receiving and inspection	Delivery controller, stored in Transaction and Inventory tables

Correspondences showing that the front-end and back-end have been separated in the project structure:



Correspondences showing that the front-end is using Bootstrap CSS and Javascript:

```
<!-- Bootstrap core CSS -->
<link href="{% static 'vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">

<!-- Custom fonts for this template -->
<link href="{% static 'vendor/fontawesome-free/css/all.min.css' %}" rel="stylesheet">
<link rel="stylesheet" href="{% static 'vendor/simple-line-icons/css/simple-line-icons.css' %}">
<link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Catamaran:100,200,300,400,500,600,700,800,900" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Muli" rel="stylesheet">

<!-- Plugin CSS -->
<link rel="stylesheet" href="{% static 'device-mockups/device-mockups.min.css' %}">

<!-- Custom styles for this template -->
<link href="{% static 'css/new-age.min.css' %}" rel="stylesheet">
```

```
<!-- Bootstrap core JavaScript -->
<script src="vendor/jquery/jquery.min.js"></script>
<script src="vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

<!-- Plugin JavaScript -->
<script src="vendor/jquery-easing/jquery.easing.min.js"></script>

<!-- Custom scripts for this template -->
<script src="js/new-age.min.js"></script>
```

5.3 Correspondence rules

The following table shows the correspondence rules related to the relative stakeholders. It explains which views in the architecture description are important for each stakeholder. These views must therefore be understandable by the appropriate stakeholders and tailored to their needs.

AD element/AD View	Logical View	Development View	Process View	Physical View
Bike Manufacturers				**X**
Users				**X**
Software Developers	**X**	**X**	**X**	**X**
Marketing Department				**X**
Project Manager	**X**	**X**	**X**	**X**
QA testers		**X**	**X**	
System Architect	**X**	**X**	**X**	
Investors	**X**			**X**

Below are the architecture decisions that were taken. These decisions serve as correspondence

rules since the views and AD elements need to follow these requirements for the architecture description to be consistent.

Project Structure

Related requirements	1. The back-end will be separate to the front-end 2. System needs to be maintainable and expandable
Alternative	1. Separate back-end and front-end 2. Integrated back-end and front-end
Constraints	1. Developer skills 2. Front-end compatibility with back-end framework
Assumptions	1. Modern browsers all support and separated front-end back-end architecture.
Architecture Decision	
Description	The ERP system will be composed of a front-end and back-end. Front-end will be charged with displaying data while the backend will take care of storing and querying data.
Rationale	1. The separations of the 2 components makes the code much more manageable. 2. Front-end changes can be done without affecting backend architecture and vice-versa. 3. Easier to split tasks within project team

Back-End

Related requirements	3. Maintainable backend infrastructure
Alternative	1. Django 2. Flask 3. Java 4. Node.js
Constraints	1. Developer knowledge about the framework/language

	2. Framework/language documentation and tutorials 3. Performance
Assumptions	1. Selected framework must be free or open-source. 2. Every developer has a certain amount of knowledge on the framework. 3. Easy access to online documentation and good community support
Architecture Decision	
Description	The ERP backend will be constructed using the Python Django framework.
Rationale	1. Excellent online documentation and community support 2. Used in industry 3. Developers familiar with Python 4. Easy learning curve

Front-End

Related requirements	1. Maintainable frontend infrastructure 2. Easily customizable
Alternative	1. CSS and HTML 2. Bootstrap 3. Vue.js 4. Angular.js 5. React.js
Constraints	1. Developer knowledge about the framework/language 2. Framework/language documentation and tutorials 3. Performance
Assumptions	1. Selected framework must be free or open-source. 2. Every developer has a certain amount of knowledge on the framework. 3. Easy access to online documentation and good community support
Architecture Decision	
Description	The ERP front will be constructed using the

	Bootstrap CSS classes and javascript
Rationale	1. Excellent online documentation and community support 2. Used in industry 3. Developers familiar with Python 4. Easy learning curve

Database

Related requirements	1. The system needs a database to store information. 2. Database needs to be hosted either locally using docker or online. 3. Database needs to communicate with the backend.
Alternative	1. MySQL 2. PostgreSQL 3. Firebase 4. AWS 5. Docker
Constraints	1. Database queries need to be quick and efficient due to high volume. 2. Easy backend to database communication 3. Minimal setting up
Assumptions	1. Database hosting will be free 2. Database provides user interface to manage.
Architecture Decision	
Description	The ERP system will be connected to a PostgreSQL database hosted within each individual Docker container.
Rationale	1. Hosting in docker is free and easy to set up. 2. PostgreSQL has great communication with the Django backend. 3. PostgreSQL comes with a user interface.

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.