GÖTÜR Project

Architecture Notebook

Version 1.2

Prepared By:

MESE Company

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 22/04/2022 | 1.0 | Draft 1 | MESE TEAM |
| 21/05/2022 | 1.1 | Justifications updated. | MESE TEAM |
| 29/05/2022 | 1.2 | Dependencies updated.  Section 7 – Key Abstractions added.  Section 9 – Architectural views enriched with extra visuals | MESE TEAM |

**Table of Contents**

[1. Purpose 4](#_Toc104752772)

[2. Architectural goals and philosophy 4](#_Toc104752773)

[3. Assumptions and dependencies 4](#_Toc104752774)

[4. Architecturally significant requirements 4](#_Toc104752775)

[5. Decisions, constraints, and justifications 5](#_Toc104752776)

[6. Architectural Mechanisms 5](#_Toc104752777)

[Architectural Mechanism 1 – Presentation 5](#_Toc104752778)

[Architectural Mechanism 2 – Controller 5](#_Toc104752779)

[Architectural Mechanism 3 – Model 5](#_Toc104752780)

[Architectural Mechanism 4 – Reliability 6](#_Toc104752781)

[7. Key Abstractions 6](#_Toc104752782)

[8. Layers or architectural framework 6](#_Toc104752783)

[9. Architectural views 7](#_Toc104752784)

[Operational View 7](#_Toc104752785)

[Database Model View 7](#_Toc104752786)

GÖTÜR

Architecture Notebook

# Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

# Architectural goals and philosophy

The system architecture is aimed to be robust and efficient enough to handle deliveries of fast moving consumer goods. In order to achieve this main goal, system architecture shall;

1. Enable rapidly adapt new set of features.
2. Support external configurability.
3. Comply with web security standards.
4. Enable modular structure that can be taken down to smaller services as needed.
5. Support operability to work with multiple identical replicas of deployments.
6. Follow well-known architectural framework(s) to make easier onboarding process for newcomers.
7. Have the capability to communicate with external APIs both synchronously and asynchronously (e.g. Payment Process REST API).

# Assumptions and dependencies

The system architecture is influenced by the following assumptions:

1. The overall size of the company, hence the development team is very small. Consequently, the boilerplate code and necessary configurations to deploy minimum viable product shall be as little as possible.

# Architecturally significant requirements

1. The system shall deliver a valid response within 5 seconds upon user request.
2. The system must communicate all data including its internal services via TLS.
3. The system must allow user to change/edit their selection until their order is complete.
4. The system must allow user to cancel their order within 30 seconds starting from the order success message.
5. The system must deploy on any Linux server or container.

# Decisions, constraints, and justifications

1. The system shall never contain business logic within the representation (i.e. UI components). The rationale behind this is both for security and modularity purposes. User interface is always due to change, by isolating business logic from the UI changes on UI will never impact other modules in the system.
2. In order to justify the purpose noted above, the system is implemented with server-side rendering. This approach makes it easier for all validations to be done in the server-side, thus ensuring data integrity.
3. System will utilize AWS for easier scalability and out of the box accessibility.
4. Modules and services implemented within the system shall serve a single purpose (e.g. Login system should not contain order information as it has nothing to do with the ordered goods). This approach allows us to eject modules into services as needed.
5. The system will be developed with Java programming language as it allows to develop a system on all major OS environments.
6. The system will be prototyped and then migrated to a container native environment, hence the target system shall be Linux based.
7. The system will run on instances or containers hosted on AWS.
8. The system shall use MySQL Server as a relational database system as it has a community edition for zero cost development and have native support for both Java platform and AWS systems.

# Architectural Mechanisms

## Architectural Mechanism 1 – Presentation

This mechanism is used for the how we present the data to accomplish the purpose of the system. It contains only styling and visual structure of the data shown to the user. It contains no actual code whatsoever.

## Architectural Mechanism 2 – Controller

This mechanism serves as an intermediary component between user, model and presentation. It captures the user input, delivers it to the model and delivers a response as a presentation to the end-user. It can contain various business rules such as validation logic etc.

## Architectural Mechanism 3 – Model

This mechanism is defined as the dynamic data model that contains the state of an operation or data required to fulfill the purpose of the system. This is where the data is persisted and rules apply to the persistence logic.

## Architectural Mechanism 4 – Reliability

System will run as a systemd service for the initial version on a Linux box with a restart rules to self-heal in case of unexpected error cases. Containerized version will be subject to a generic health check mechanism on both application and container level. In case of an unexpected failure occurrence, a new container will be spanned and replaced with the stale version.

# Key Abstractions

1. Thymeleaf will be the main templating engine for the HTML content (i.e. View portion of the architecture).
2. Thymeleaf templates will be filled with the data supplied from the controller classes (e.g. MainController, WebSecurityController etc.).
3. Every request made by the clients will directly invoke the related controller on the URL path and the requested data will be sanitized and verified by the controller.

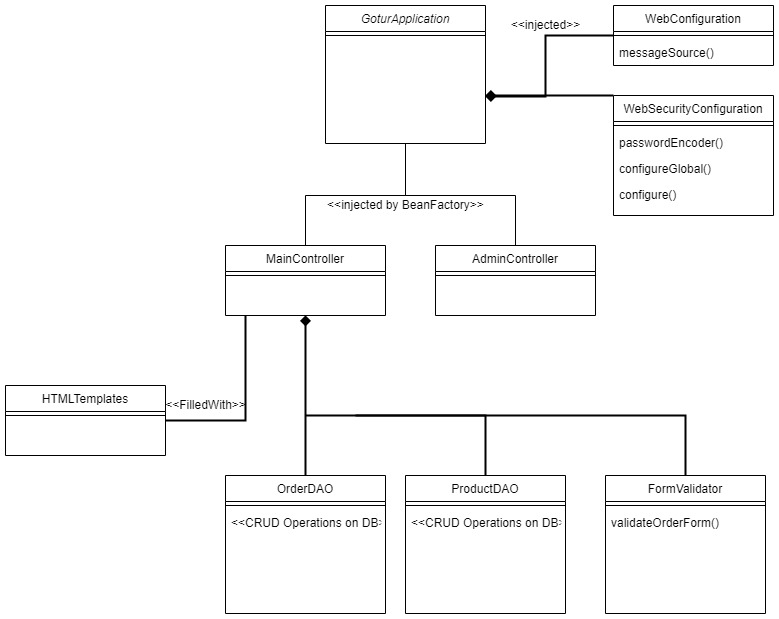
# Layers or architectural framework

The system will be implemented with well-known Model-View-Controller (MVC) architectural pattern. This pattern enables the modular structure of the system while it requires minimum amount of boilerplate code for developing minimum viable product.

It is also extensible and flexible enough to implement new feature sets. Its monolithic structure makes it easy to deploy for demonstration purposes and works well for small systems that does not require scaling exponentially.

# Architectural views

## Operational View



## Database Model View

