Architecture Notebook

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author(s) | Changes | Version |
| 03/05/2021 | Özgün Şen  Berfu Anıl | initial document | v1.0 |
| 31/05/2021 | Özgün Şen  Berfu Anıl | Review document | v1.1 |

# **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**

1. Architecture should provide conceptual simplicity, performance, and maintainability.
2. All performance and loading requirements, as stipulated in the Vision and the Use-case Specification, and System-wide Specification must be taken into consideration as the architecture is being developed.
3. System data will be kept in a central DB. Their activity, expiry and type attributes will be kept in the same place.
4. All interfaces and server-side operations will be appropriate to run on Azure Cloud
5. The system should be extensible without modifying or rewriting existing components.

# **Assumptions and dependencies**

In this section, there is a list of the assumptions and dependencies that drive architectural decisions.

1. The technologies that will be used in the project must be discussed with all team presence.
2. Implementation of the project should consist of C# language limitations.
3. All functionality must be available to PCs with internet connections.
4. The software must ensure complete protection of data from unauthorized access. All operations are subject to user identification and password control (except some of the temporary visit operations).
5. The software will be implemented as a client-server system. The client-side doesn't need any dedicated additional software. Clients should access the system on their web browsers.

# **Architecturally significant requirements**

1. [SWReq#3.1] The interface must be resizable for all devices such as mobile, tablets etc. The interface must show the user experience design principles such as eye-friendly texts, colours, element patterns to ease the use of it for people of all ages and multi-language UI screens must be supported.
2. [SWReq#4.1] The system will interact with the users through an easy to use web interface. This interface will work with the Chrome browser without experiencing any design problems.
3. [SWReq#2.1.2] The system shall allow the user to log in to the system.
4. [SWReq#3.2] The system should be up and running in a 24 hours operation period and wrong entries made by the user will be handled and will not interrupt the system.

# **Decisions, constraints, and justifications**

Styling in this section is based on a *technical memo.*

Introduction: The final solution will run on Azure Cloud to provide scalability. The cloud-based solution is preferred which keeps the system always up and balance loaded at very low prices compared to running a standalone server system. Azure will be responsible to protect the system from suspicious attacks (such as. DDOS). There is no need for an extra system to maintain security. MSSQL will be used as a database which is also an Azure Platform Service with the name “SQL Database”

* The development will follow the MVC pattern provided by Microsoft ASP.NET Core Web Application.
* Database first approach will be used.
* Dapper is used as an object-relational mapping product. This platform provides a framework for mapping an object-oriented domain model to a traditional relational database.

Experimental: Azure database is observed and chosen as a database connection tool.

Discussion: Azure database provides its own security checks so we didn’t have to configure security issues such as firewall, IP protection etc. Also Azure provided a fast connection to the user database (approximately 1 sec) including the getting request time from the server.

Conclusions: The expected performance was useful while implementing the cloud database along with maintaining the connection when we tried to log in. MVC pattern is a widely used and useful pattern to implement a web-based system. We conclude to continue with MVC.

# **Architectural Mechanisms**

Decisions regarding database systems, MVC patterns are described in the previous section. Below are the explanations of those mechanisms.

## MSSQL

One of the most commonly used RDBMS produced by Microsoft.

## MVC

MVC is the design pattern that separates application concerns into three containers: Model-View-Controller. Briefly; Controller handles incoming requests and controls the flow. Model stands for Data layer. The view is the concern working on user interfaces. The fifth version of the Microsoft MVC framework is used.

## Database First Approach

In this approach Database and tables are created first by developers. Then they create entity data models using that Database.

# **Key abstractions**

# 

# **Layers or architectural framework**

As mentioned above MVC pattern will be followed in structure, the architecture follows layered architecture principles. There are three main layers in the architecture.

## UI Layer

The user interface should be placed in this layer. The Stockupy.UI(view) namespace is used to specify this layer’s elements.

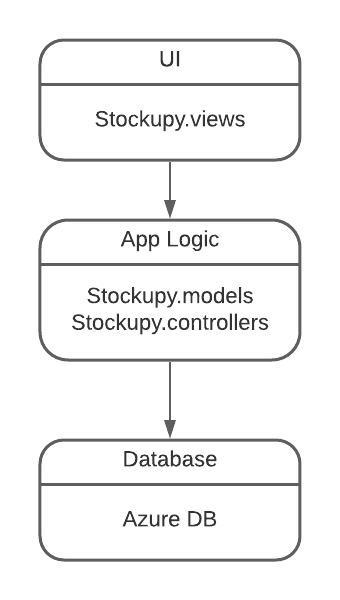
## Application Logic Layer

The application logic, business operations and domain classes are kept in this layer. Stockupy.Controllers and Stockupy.Models namespaces are used to specify this layer’s elements.

## Database Layer

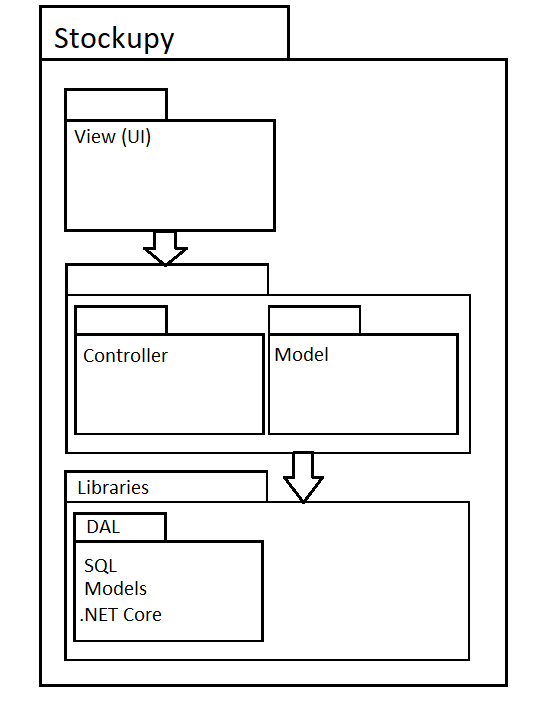
The database is kept in the Azure cloud platform.

## Diagram

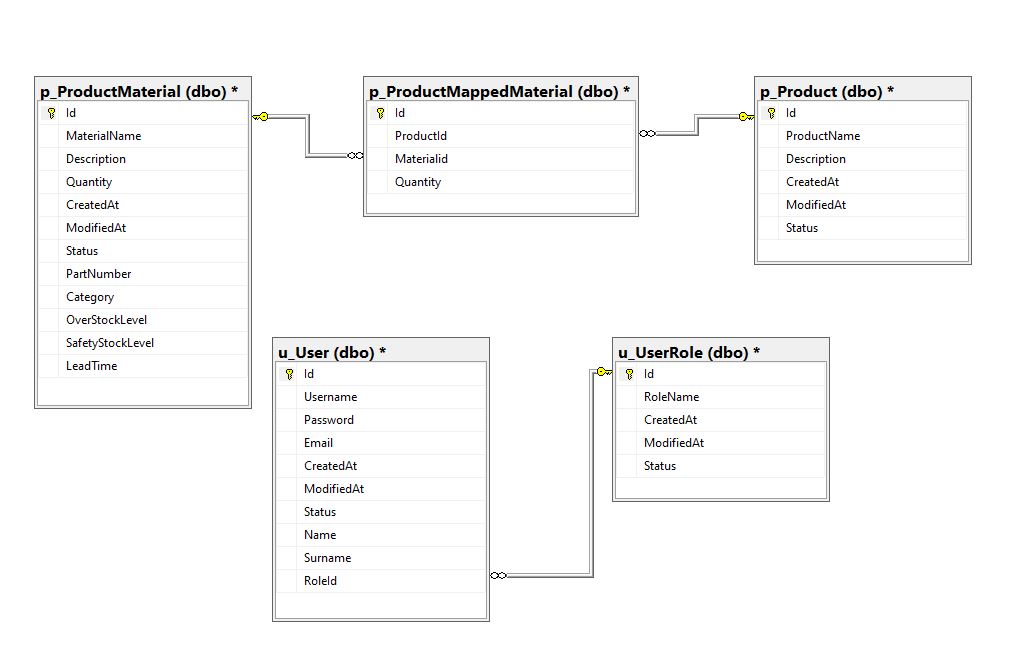


# **Architectural views**

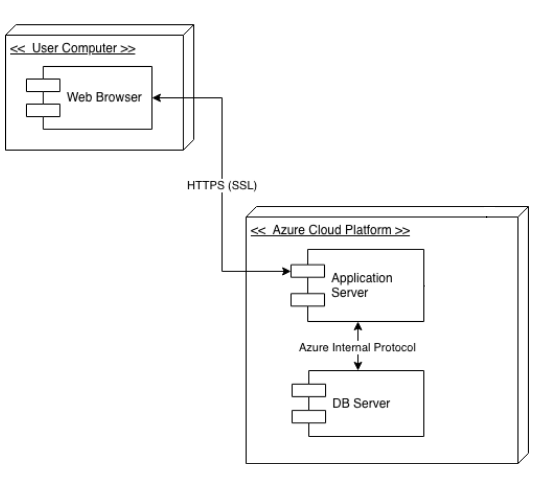
## Logical:



## Data View:



## Deployment:



## Use-case View

## 