VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

**Andrius Bertašius**

andrius.bertasius@mif.stud.vu.lt

**Rytis Čepulis**

rytis.cepulis@mif.stud.vu.lt

**Juozapas Gelčys**

juozapas.gelcys@mif.stud.vu.lt

**Eligijus Mickus**

eligijus.mickus@mif.stud.vu.lt

Table of contents

[1 About us 4](#_Toc73299492)

[1.1 Introduction 4](#_Toc73299493)

[1.2 Vision 4](#_Toc73299494)

[1.3 Mission 4](#_Toc73299495)

[1.4 Aim 4](#_Toc73299496)

[1.5 Problem 4](#_Toc73299497)

[1.6 Solution 4](#_Toc73299498)

[2 Changes 4](#_Toc73299499)

[2.1 Reasons for implementing new changes 5](#_Toc73299500)

[2.1.1 In-app chat. 5](#_Toc73299501)

[2.1.2 Location sharing 5](#_Toc73299502)

[2.1.3 Notifications 5](#_Toc73299503)

[2.1.4 List of orders 5](#_Toc73299504)

[2.1.5 Order status 5](#_Toc73299505)

[3 System stakeholders’ responsibilities 6](#_Toc73299506)

[4 Requirements 6](#_Toc73299507)

[5 System Architecture 7](#_Toc73299508)

[5.1 Service-oriented architecture 7](#_Toc73299509)

[5.2 Client-Server 7](#_Toc73299510)

[5.3 MVC 7](#_Toc73299511)

[5.4 Data-Centered 8](#_Toc73299512)

[5.5 CRUD 8](#_Toc73299513)

[5.6 Agent model 8](#_Toc73299514)

[6 Viewpoints 8](#_Toc73299515)

[6.1 Context Viewpoint 8](#_Toc73299516)

[6.1.1 Glossary……………... 8](#_Toc73299517)

[6.1.2 Use cases… 9](#_Toc73299519)

[6.2 Functional Viewpoint 9](#_Toc73299520)

[6.2.1 Components 9](#_Toc73299521)

[6.2.2 Components in more detail 10](#_Toc73299522)

[6.3 Information Viewpoint 10](#_Toc73299523)

[6.3.1 Static Information diagram 11](#_Toc73299524)

[6.3.2 Information Flow 11](#_Toc73299525)

[6.3.3 Information life cycle 14](#_Toc73299526)

[6.4 Development Viewpoint 14](#_Toc73299527)

[6.5 Deployment Viewpoint 15](#_Toc73299528)

[6.6 Operational Viewpoint 16](#_Toc73299529)

[6.6.1 Support……. 16](#_Toc73299530)

[6.6.2 Administration 16](#_Toc73299531)

[6.6.3 Backup and restore 17](#_Toc73299532)

[6.6.4 Performance monitoring 17](#_Toc73299533)

[6.6.5 System updates 17](#_Toc73299534)

[7 Perspectives 17](#_Toc73299535)

[7.1 Security 17](#_Toc73299536)

[7.2 Performance and scalability 18](#_Toc73299537)

[7.2.1 Performance 18](#_Toc73299538)

[7.2.2 Scalability… 18](#_Toc73299539)

[7.3 The Availability and Resilience 18](#_Toc73299540)

[8 Testability 19](#_Toc73299541)

[8.1 Testing environment 19](#_Toc73299542)

[8.2 Testing tools 19](#_Toc73299543)

[8.3 Implemented tests 19](#_Toc73299544)

[9 Traceability 20](#_Toc73299545)

# About us

## Introduction

For many people, the current pandemic period is tough. It is prohibited meeting friends, having parties and celebrations, going to cafes and restaurants. But many people have been able to adapt to these challenges: we can meet and talk with others using “ZOOM”, restaurants and cafes have started taking orders for takeaway. But not everything is as good as it sounds, many small businesses or individual sellers do not have huge production power to become partners of “Wolt” or “Bolt”. Such businesses should not be left alone and so our team is ready to help them.

## Vision

System which gives a simple way to buy and sell your own produce.

## Mission

Create a high quality and simple platform for small businesses, farmers and ordinary people, who want to sell their own production, that will help these sellers increase their number of customers.

## Aim

Motivate people to be interested in smaller businesses and help them grow.

## Problem

It is hard for small businesses to grow, get new customers, currently their main source of income are people living nearby. This has become even harder during quarantine, when people count is limited in markets.

## Solution

Platform which lets its users (sellers and buyers) engage in trade and communicate with each other, help perform contactless deliveries and increase popularity of sellers.

# Changes

The document focuses on these system modifications:

* a live in-app chatting system, where messages can be sent by both parties.
* notifications, which inform users with useful information related to their orders.
* the user's current location sharing, which shows current user’s location.
* orders list where users can check what orders they have made.
* statuses of orders, which can be set to canceled, delivering, or delivered.

## Reasons for implementing new changes

### In-app chat.

The communication between seller and buyer is a crucial part for this project to succeed. By implementing in-app chat, we will provide our customers an opportunity to achieve goals faster, exchange latest information about goods and deliveries. This will vastly improve user experience since they will be able to get all needed information directly from other people without using external devices or other communication services. Using in-app chat will save time for both seller and buyer.

### Location sharing

Users might have trouble finding each other in real life when an order is being delivered, therefore we provide an ability to show the user’s current location. This way users can easily find each other.

### Notifications

Notifications allow to keep in touch with your users in a non-intrusive way by providing timely messages and helpful and relevant information.

### List of orders

The Seller needs to know what orders need to be delivered and the buyer needs a way of checking what orders they made. Also, it is convenient to have all essential information about orders in one place, so the users can simply check what they ordered or sold.

### Order status

It becomes hard to track orders and if they were delivered, when they will be delivered, or when a buyer simply changes his mind about an order when the amount of them is increasing. To avoid misunderstandings and for easier orders control orders will have these states:

*Order status “delivering”*

For buyers it is very important to know when the products they have ordered are going to be delivered. A lot of times buyers must schedule their timetable to accept an order, for this reason it is crucial to inform them beforehand.

*Order status “delivered”*

It is important for seller to update status of delivery. Only by checking and changing statuses of order seller can document and make sure he did not forget to deliver an order to his customer.

*Order status “canceled”*

Under the European consumer law, a buyer's right to cancel an order starts the moment you place it and does not end until 14 days have passed from the day your goods were delivered to you. It is important to have this possibility if you want to ensure your customers rights.

# System stakeholders’ responsibilities

User. User can be either a buyer or seller. The main functions that both buyer and seller can perform is: item search, send message, share location, view orders list, view message and set order’s status to “canceled”. The main functions that only buyer can perform is: order an item (this includes add a product to shopping cart, remove product from shopping cart, view what is inside a shopping cart, confirm the order). The functions that only seller can perform is: upload a new product, set order’s status to “delivered” or “delivering”.

Administrator. System administrators are responsible for monitoring trouble-free operation of the system. Their main goal is to make sure the information is reliable which includes reviewing and managing shops and their products. They can verify sellers and ban users who do not meet certain criteria.

Developer. The main duties and responsibilities of developers include smooth operation of the system, creating maintainable and manageable platform for all system users.

# Requirements

Functional:

1. Chat system.
   1. A real-time transmission of text messages between buyer and seller.
      1. Users can send messages to other users that have order connection with.
      2. Users can view old and new messages in the chat window.
2. Location sharing between users.
   1. Using chat system users can share their location.
      1. Users can share their current location with another user. The shared location is updated every 30 seconds.
      2. Users are notified whenever their device’s GPS cannot be accessed.
3. Orders table.
   1. Buyer can see a table of orders they had ordered with relevant information (status, price, seller, produce list).
   2. Seller can see a table of orders their clients have ordered with relevant information (status, price, buyer, produce list).
4. Order has an updatable status.
   1. Order status can be changed to “canceled”.
      1. Users can change order status to “canceled” by clicking on the status drop-down list next to the order.
   2. Order status can be changed to ”delivering”, ”delivered”.
      1. Sellers can change order status to “delivering” by clicking on the status drop-down list next to the order.
      2. Sellers can change order status to “delivered” by clicking on the status drop-down list next to the order.
5. Notification sending.
   1. Notification sending when order status is updated.
      1. Users receive notification when order status is updated by the other user. The user can click the notification and “Chat” window will open and notification will be displayed.
   2. Notification when new message is received
      1. Users receive notification when a message is received from another user. The user can click the notification and the “Chat” window will open and a message will be displayed.

# System Architecture

## Service-oriented architecture

The system is going to be made based on service-oriented architecture. This type of style makes the system run a little bit slower, however it offers a lot of advantages:

* The system is going to use pre-built external services such as google maps API. Also, in the future external sign-in options such as Google, Facebook sign-in systems. SOA style makes it simpler to implement these services.
* Easy reusability. The UI will be a bit different for buyers and sellers however both clients will have a lot of same functionality and SOA style offers the needed reusability.
* Easy extendibility. As the website picks up more popularity we might want to offer more and new features for the users and SOA style makes it easy to extend the system.

## Client-Server

The structural style of the system is going to be three tier client-server architecture. It is a common architecture for web applications. The system has three tiers – presentation, logic, data. The tiers can be directly mapped to usual fields of software development – front end, back end, database management. This offers some advantages:

* Security. The system is more secure when database cannot be accessed by anyone. The users can access only client (front end).
* Separated client from API makes it possible to implement several clients easily reusing some of the same functionality.

## MVC

We will use very popular MVC pattern. Advantages are very similar to advantages mentioned in Client-Server style: security, reusability.

## Data-Centered

The system will use data-centred pattern That means there will be only a single database. This pattern makes the system a little slower, however it is irrelevant for this system because speed is not crucial for this kind of system. For example, the user will feel no difference if they receive a message in 10 seconds or 15 seconds after the sender sent it. Main advantage of the data-centred pattern is the simplicity of it. That makes it easy to maintain and extend it in the future.

## CRUD

Communication between front-end and back-end is written with CRUD pattern in mind. That means the system only uses create, read, update, delete functions to modify data. It allows the system to be easily modified and understandable for possible new developers.

## Agent model

System’s chatting and notification functionality is based on an agent model pattern. Whenever a user sends a message it is put into the database along with a notification and the recipient client gets the message and the notification by requesting messages from the database.

# Viewpoints

## Context Viewpoint

The Context view of a system defines the relationships, dependencies, and interactions between the system and its environment—the people, systems, and external entities with which it interacts. It defines what the system does and does not do; where the boundaries are between it and the outside world; and how the system interacts with other systems, organizations, and people across these boundaries. [[1]](#footnote-2)

### Glossary

### User is a buyer or a seller.

Buyer is a system user who buys or intends to buy products.

Seller is an approved system user who sells products.

Product is an asset supplied by the seller.

Order is a confirmed arrangement between seller and buyer containing the products that a buyer has bought from the seller.

Order list is several connected orders to one particular buyer or seller. Order list can be empty.

Message is a text sent to a buyer or seller. Messages can also contain current location. Current location is a particular position where the message sender is at the present.

Sharing of location – updating current location every 30 seconds until the user cancels sharing.

Chat – real-time transmission of text messages between sender and receiver.

Status – the particular condition that someone or something is in at a specific time. Status can be canceled, delivering, delivered.

### Use cases

The change of the system focuses on adding more functionality:

* chatting system (view messaged, send message, share location) which any user will be able to use.
* orders list – view orders list, changing orders status (to “canceled” for any user, to “delivered”, “delivering” for seller.)

They are shown in a use case diagram figure 1.

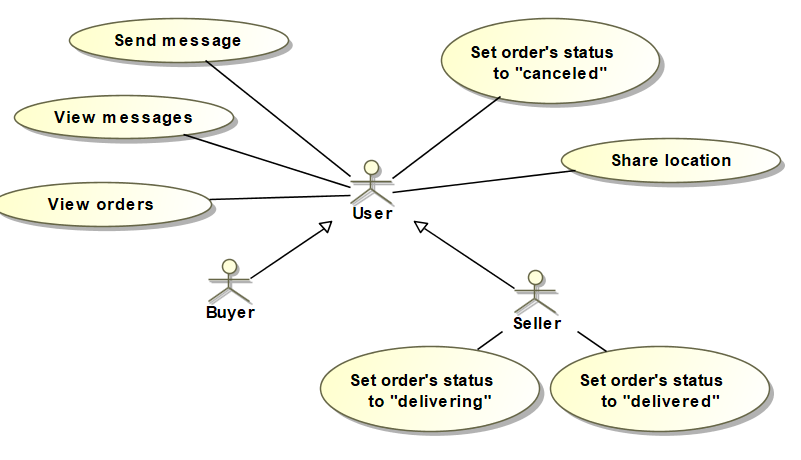


Figure 1. Use cases for modification.

## Functional Viewpoint

The functional view of a system defines the architectural elements that deliver the functions of the system being described.

### Components

The system change will be made according to MVC pattern. Therefore, will have 3 components defined in MVC: model (for data management), view (to provide interface for user’s interaction with the website), controller (for communication with view and model), also database which will contain all the data. It is shown in the diagram figure 2.

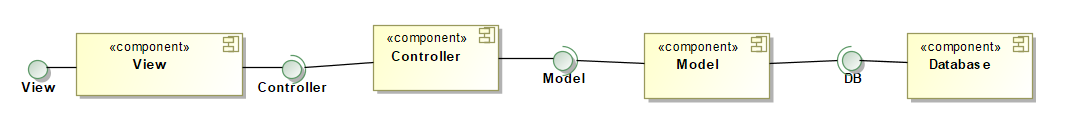


Figure 2. MVC model for components.

### Components in more detail

Gariunai cloud service system will have two user interfaces, so our view component is split into two: buyer view and seller view. Our change to the system will add three more controllers: chatting controller, notifications controller, orders controller for managing chats, notifications and orders respectively. Also chat controller will communicate with external interface for user location which will be provided by google maps. These less abstract components are illustrated in the detailed components diagram figure 3.

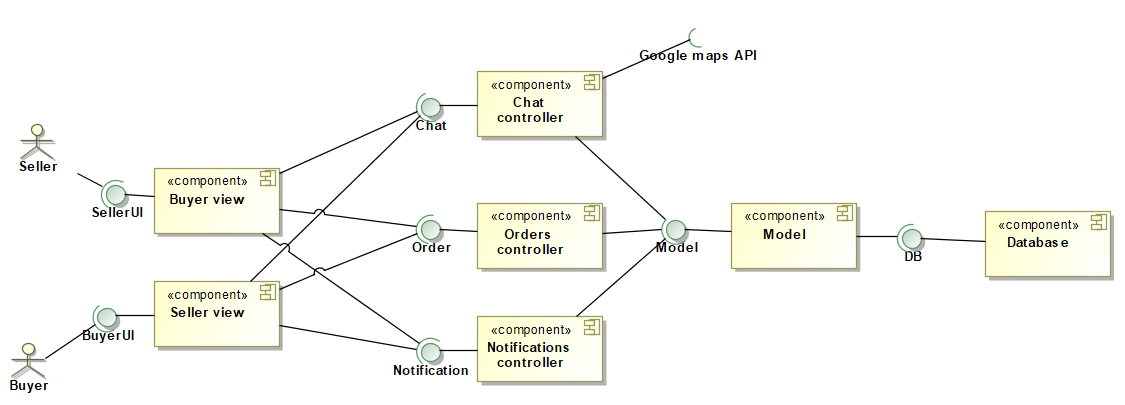


Figure 3. More detailed component diagram for modification.

## Information Viewpoint

The Information View details how the system will manipulate, manage, and distribute information.

Our system change will add few new entities to the database. Firstly, chat which will have user1 and user2 attributes which will show which users is the chat between, a list of messages, this list will be used when loading the chat window and the system will have to show the previous messages. Also, a big part of our change will need an order entity, however it is already implemented I the system. This is shown in figure 4.

### Static Information diagram

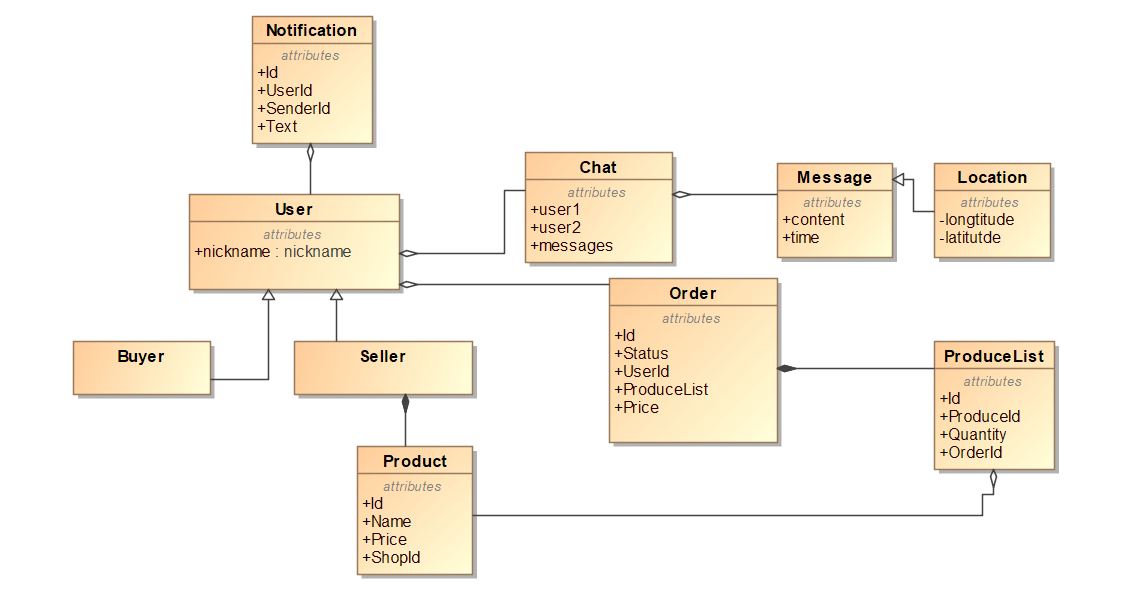


Figure 4. Static information diagram.

### Information Flow

Information flow is shown in the figure 5. An action from Buyer View and Seller View (both shown as View in the diagram) is sent to orders, chatting or notification controller (all of them shown as Controller in the diagram), controller updates model accordingly. Then the model updates database. Model receives information from the database, model updates controller and finally updated data is sent to View.

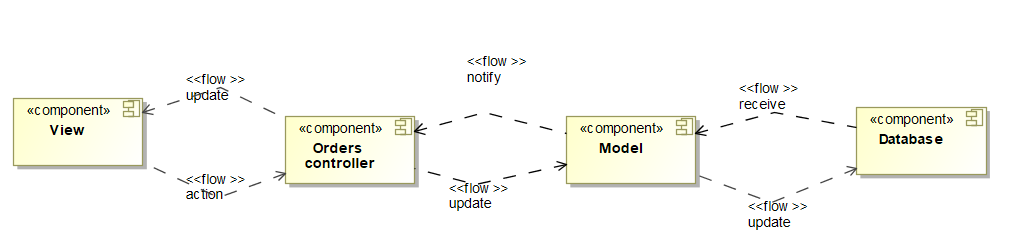


Figure 5. Information flow between components.

When a user (either buyer or seller) clicks to check orders list a request with his id is sent from View (buyer or seller view) to orders controller. Inside the orders controller component another request is sent to model and then model queries the database. Database returns orders which unambiguously defines user to the model, which then returns it to the controller. The controller processes received orders and sends them back to view. View receives orders and displays them for the user. This is shown in a figure 6.

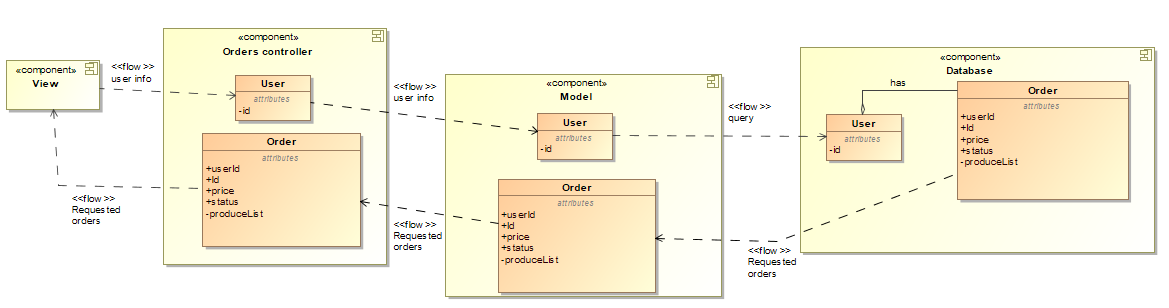


Figure 6. Information flow when user opens orders list..

When a user (either buyer or seller) clicks to change status a request with the order is sent from View (buyer or seller view) to orders controller. Inside the orders controller component another request is sent to model and then model updates the database. Database returns updated order to the model, which then returns it to the controller. The controller sends the updated order to the view, which then displays it for the user. This is shown in a figure 7.

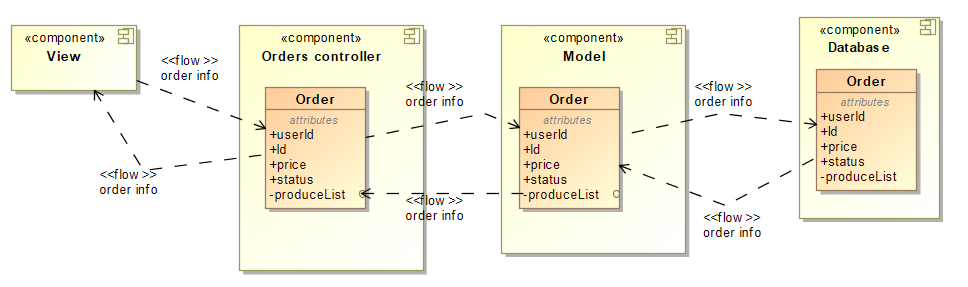


Figure 7. Information flow when user changes status of order.

When a user (either buyer or seller) clicks to check the chat a request with his id is sent from View (buyer or seller view) to orders controller. Inside the orders controller component another request is sent to model and then model queries the database. Database returns chat which unambiguously defines user to the model, which then returns it to the controller. The controller processes received chat and sends them back to view. View receives chat and displays them for the user. This is shown in figure 8.

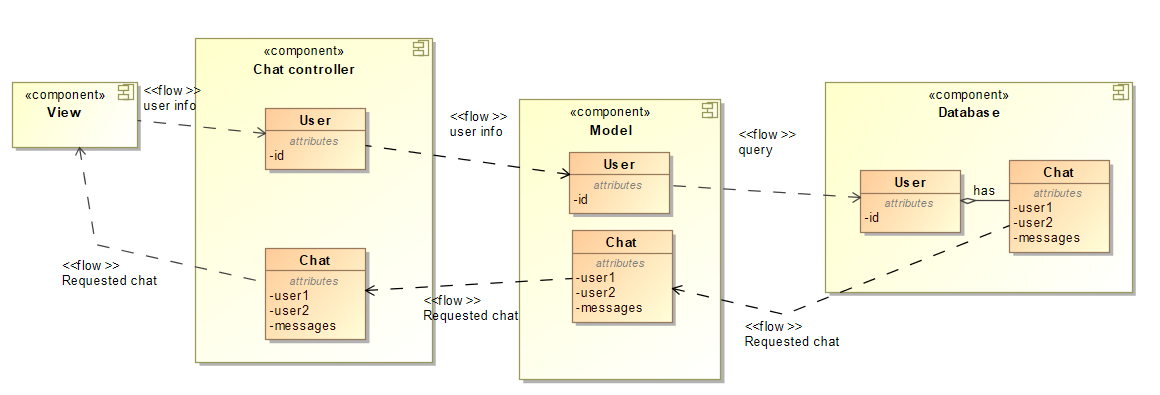


Figure 8. Information flow when user checks his chat.

When a user (either buyer or seller) sends a message it is added to chat class which is then sent from View (buyer or seller view) to chat controller. Inside the chat controller component another request is sent to model and then model updates the database. Database returns updated chat to the model, which then returns it to the controller. The controller sends the updated chat to the view, which then displays it for the user. This is shown in a figure 9. In addition to that when a user sends a message a notification is sent from view to notifications controller which is then sent to model and model adds it to the database. This is shown in the figure 10. When a user clicks to see notifications a user id is sent to notification controller, which sends request for notifications to the model, then it sends query request to the database and receives notifications for the specified user. Finally model returns the notifications to the controller which returns then to the view to display it for the user. This is shown in the figure 11.

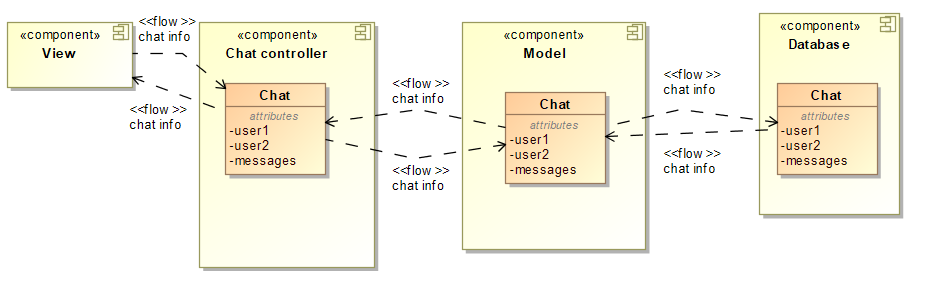


Figure 9. Information flow when user sends a message.

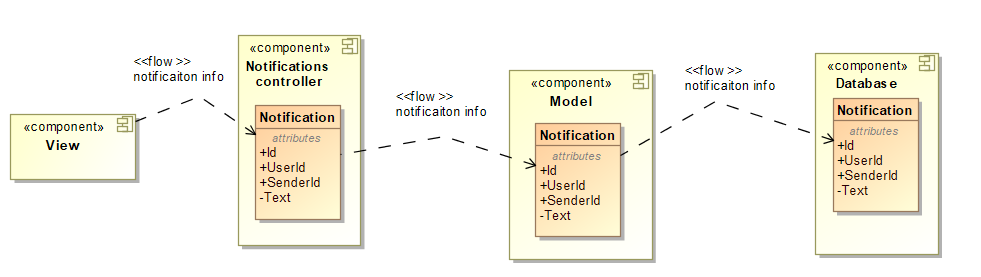


Figure 10. Information flow for notification when user sends a message.

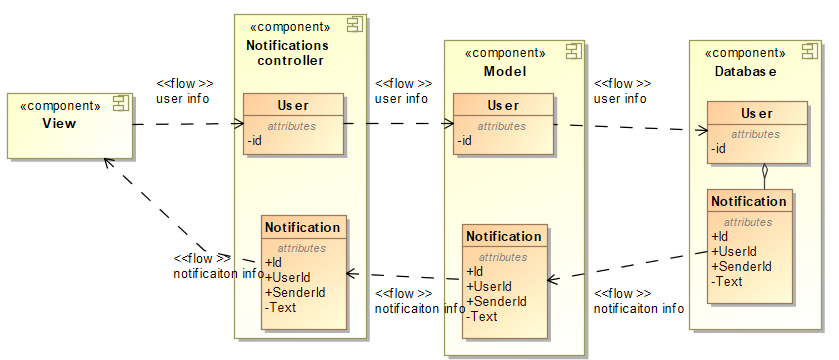


Figure 11. Information flow when user opens notifications list.

### Information life cycle

An order is created the moment buyer pays for his shopping cart – the system saves it to the database. User can view and modify state of his orders in orders list. After order status is set to “delivered” or “cancelled” the system moves order to the archive and makes it inaccessible to users. If order status was never changed, after 6 months it is put to archive. After a year of storage data is considered no longer needed and order is removed from database.

A chat is created when one user (either buyer or seller) starts a chat with another user. The chat between users are never deleted, however one year old messages are deleted from the database.

Notifications are created when a user receives a notification and deleted after one year.

## Development Viewpoint

The Development View defines how to implement the system.

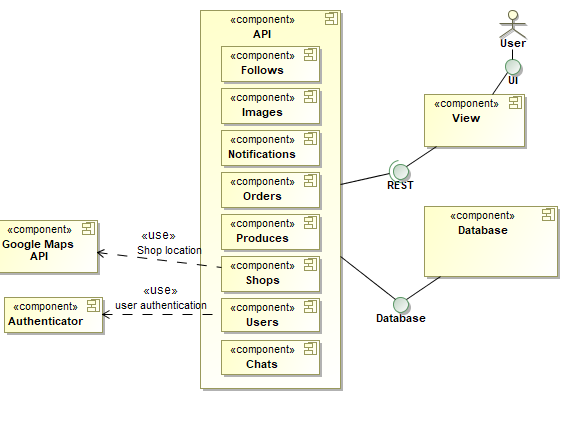
UI connects to API, which contains multiple controllers, connected with different UI pages. Each controller is designed only for specific data gathering, updating and UI page. API is connected to database, where all the information for each controller needs is stored. User can only access UI, and the rest is being controlled by API. This is shown in figure 12.

Figure 12. System components.

## Deployment Viewpoint

The Deployment View defines how to transition the system to live operation.

The user will access our system via web browser by going to our website. The web server which will be hosted on cloud.google.com will send an html file to the user’s web browser. When the user does something the web server sends a request to API server using HTTP protocol. When the user wants to send location the google maps will be needed, therefore the API server will need to communicate Google maps API through HTTP protocol. Finally, most of the data (User data, shops data, orders data, chatting data) is stored in the database, so whenever we need to access that data Web API will communicate with database server which will be hosted on Azure.com website. This is shown in figure 13.

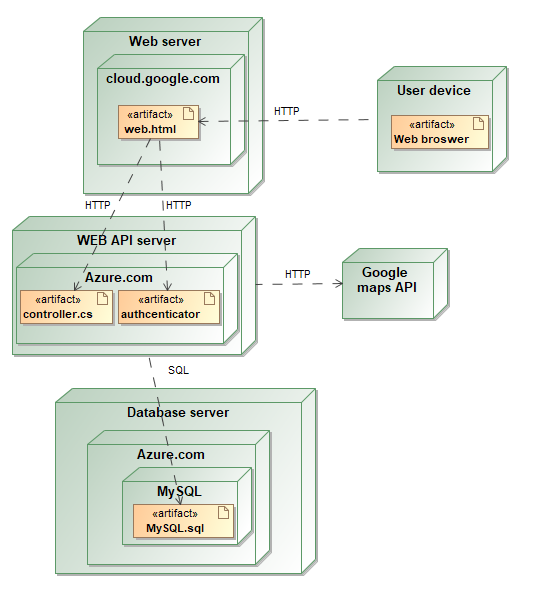


Figure 13. Dependencies.

## Operational Viewpoint

The Operational View defines how to keep the system alive in the field.

### Support

This system is based mostly on two users interaction(buyer and seller).Their communication might not always end in good way, user might have few questions about the system, how to use it, or anyone might have their own concerns. To help our users experience better we decided that we need to have support service. Our support service will be automated and this part of service will have users with simple questions, that they will be able to choose from list. If user will require further assistance we will contact him with one of our support workers. This will provide additional costs to project maintainability but it will increase user experience overall.

### Administration

System administrator plays huge role in our system. Admin must confirm new shops, their addresses, also, just like support workers, might get involved into dispute resolution. Sometimes administrator might have to confirm real world shop location. Administrator might also check shops reviews, their produces to see if no illegal activities are being done and if they are, take care of them.

### Backup and restore

System will have on Azure hosted MySQL database. Copy of database will be done every night, when there is lowest user count using system. Only last 5 days copies of system will be saved, older ones will be deleted.

### Performance monitoring

As we have no experience in this type of field, we haven't decided what hardware are we going to choose. Our main goal is to keep system operating without any interference all the time, so we will try to choose hardware accordingly to that.

### System updates

When updating a system we must ensure that all the currently running and newly added features are functioning correctly, plugins are up-to-date. Most of this work will be automated and testing tools will make sure that previous features are correct. New features will have to be carefully reviewed by our team. Changes will be applied to the website directly, during night, about 3am, the system will shut down, changes will be applied and it will start running again.

# Perspectives

## Security

System security is important not only for system developers but for all stakeholders. Sometimes database user data might contain sensitive information, sometimes ordering or any other process might be interrupted, which will lead to unexpected and unwanted results. To prevent from anything like this happening to system we must take care of security.

|  |  |
| --- | --- |
| Viewpoint | Possible threats/solutions |
| Context | * Unwanted users stealing accounts. To protect system from this kind of harm, system has implemented password hashing. * Direct access to database is not available. All information flows through API service, entity framework takes care of further data management. |
| Functional | * Database must be protected from SQL injections. Entity framework can protect from straight up SQL injections, but not from parameters change (example: if user, that wants to harm the system, somehow finds out http request link, he can change the parameters to others, in that way he can get access to admin privileges, change actual request data, etc.). |
| Information | * Database server might stop working and there are no data backups being done, so in that case all the data can be lost. To prevent this from happening we must implement server data backup system. * Only user can access his personal information, orders, etc. Passwords are hashed. |
| Development | * User only has access to UI. This prevents users from direct access to database and other services. |
| Deployment | * CI/CD prevents system from unwanted updates, tests make sure that system functionality is stable. Only master branch is being hosted. |

Table 1. Security perspective. Analysis of viewpoints.

## Performance and scalability

### Performance

The performance in our system is not the key goal, however, it needs to meet certain criteria.

### Scalability

A successful web application needs to seamlessly and efficiently accommodate growth, and be designed with scalability in mind.

|  |  |
| --- | --- |
| Viewpoint | Possible threats/solutions |
| Context | * Since system is based on SOA design it is easy to expand it. |
| Functional | * System uses as little as possible and only necessary microservices. * Database is being updated only with important information, that must be saved. No unnecessary changes are being made. |
| Information | * While using data-centered design a database can become a bottleneck thus diminishing performance. * To make information flow faster the system could use several databases to avoid bottlenecking. |

Table 2. Performance and scalability. Analysis of viewpoints.

## The Availability and Resilience

System must stay accessible and must maintain basic functionality even when updates are being developed. So far, our team has not implemented such feature, but we decided to forecast the problems and solutions we might face, when developing it.

|  |  |
| --- | --- |
| Viewpoint | Possible threats/solutions |
| Functional | * API, WEB, database servers must be accessible 24/7 because of users' needs and overall system stability and monitoring. * When new functionality is being added to controllers, components, pages, other system functions must remain fully functioning, if they don’t contain components that are being updated. |
| Information | * In case of emergency database must have a backup. Database backup might be done every day at midnight, when users count using system is lowest (only last 5 days backups are being saved, others are being deleted). Database copy should be saved on another private server that does not have contact with main server directly. |
| Deployment | * We think that system servers should be located in Lithuania, because most of users will use it there. But the internet is fast these days, so few milliseconds loading difference will not make huge influence on user experience or database management. |

Table 3. The availability and resilience. Analysis of viewpoints.

# Testability

To make sure that our system’s main functionality is working correctly, and nothing is broken we have implemented automating testing for running unit and integration tests. Automated testing is not only useful but, in some cases, even necessary, because they help developers catch unforeseen errors before making the code go into production. This way money and time is saved.

## Testing environment

To implement automated testing, we have used GitLab runner to automatically run all test cases when a new branch is created and every time a new push is made to that branch. GitLab runner is configured in Vilnius University’s private cloud platform, and this is where all of these tests are run.

## Testing tools

We used xUnit NuGet package for adding testing project and running unit tests, we also made use of MoQ framework for mocking objects also to have better and more readable assertions we used FluentAssetions package. For storing data required for testing we are making use of In-Memory database that like its name suggest uses computers memory for data storage.

## Implemented tests

When creating tests, it is important to identify crucial system parts. Because we did not have enough time or resources for creating test for the whole system, we identified that our implemented changes should take priority and only after that to the other system parts.

# Traceability

Chatting system is delivered by the chat controller component, also the same controller using google maps API delivers the location sharing. Items ordering and order status is delivered by the orders controller. Finally, notification feature is delivered by the notification controller. This is illustrated in table 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | FR1 | FR2 | FR3 | FR4 | FR5 |
| Chat component | X | X |  |  |  |
| Order component |  |  | X | X |  |
| Notification component |  |  |  |  | X |

Table 4. Components to functional requirements matrix.

1. https://www.viewpoints-and-perspectives.info/home/viewpoints/context/ [↑](#footnote-ref-2)