Food Waste

Arentas Meinorius,
Jaunius Tamulevičius,
Martinas Mačernius,
Pijus Petkevičius

Matematikos ir informatikos fakultetas
Vilniaus universitetas
Lietuva
May 26, 2022

Summary

The primary objective of the fourth laboratory assignment is to define the architecture of the system. This time we will write about the viewpoints and perspective of the system for what the system is and what we added to it.

The main tasks of this iteration:
1. Allergens feature
2. E2E test
3. Document the E2E test
4. Viewpoints
5. Perspectives
6. Traceablity tables

Contents

1 Context

For a system to be successful, it must be developed with the intention of solving a real-world problem, which, in our case, is reducing food waste in restaurants and shops. The software is useless if it does not solve required problem. In this part we analyse our problem and how it is intended to be solved.

1.1 Goal of the system

Reduce food waste by distributing it.

1.1.1 The problem

Not all food products are sold before spoiling, sometimes restaurants do not use all the food they have bought.

1.1.2 Solution

Prepare a plaftorm that would stand as a middle man helping people sell excess food while allowing others to buy it cheaper.

1.2 Planned changes

To further develop and increase the functionality of the existing system we want to realise a new functionality:

Every food Product has an allergens list. The list can be changed when the product is added or modified.

1.2.1 Change list

- Allergens tags on food products.
- User have their own allergens in their profile.
- Tests for the new functionality.

2 Viewpoints

2.1 Context viewpoint

2.1.1 Context system scope and responsibilities:

The whole system: FoodWaste systems tries to eliminate hunger and binned food. Providers such as shops and restaurants inform users about foods that would be thrown away in the near future by giving them access to before mentioned food. Users that are interested in zero waste lifestyle as well as those in a financial troubles are welcome to use FoodWaste.

The system update: Allergen functionality aims to protect users from the ingredients that are harmful to them.

- Allergens are properties of food products that are likely to cause unwanted reactions with allergic individuals.
  - All food offerings may contain a list of allergens that have to be set by the restaurant
  - All users may provide a list of allergens that are dangerous to them

- System will use the user and product allergen information to:
  - Alert if user is trying reserve product that may be dangerous to the individual
  - Provide easy filtering of allergic products (in the future)

2.1.2 Context external entities and services and data used:

The whole system: FoodWaste uses external providers for: authentication, monitoring, data storing, payments, accounting and live support agents.

The system update: Annex 2 naming system for allergens will be used. No other external entities are concerned with this change.

2.1.3 Context impact of the system on its environment:

The whole system: FoodWaste uses CI to automate deployment process so no system should be impacted by different updates. Restaurants and shops that use FoodWaste API will have to make updates to their systems when there will be any breaking changes or new feature releases.

The system update: Not applicable.

2.1.4 Context consistency and coherence

The whole system: FoodWaste uses single point of truth: the database. Data is inserted via transactions to ensure validity of entries. Validation of information is executed in the FE as well as BE. Currently the system is not complex enough for coherence problems to arise.

The system update: Not applicable.

2.2 Functional viewpoint

2.2.1 Functional capabilities:

The whole system: FoodWaste provides platform for providers and users to list, reserve and retrieve foods that would be thrown away in the near future. FoodWaste does not provide transportation or courier services.

The system update: Allergen functionality provides information on allergens and alerts. It does not protect users that ignore warnings.

- Each restaurant will have an option to add allergens to products:
  1. While creating offers
  2. Anytime while offer is not reserved
- Each restaurant will have an option to remove allergens from offers anytime while products are not reserved
- Each user will be able to add allergens to their profile anytime

2.2.2 Functional external interfaces:

The whole system: FoodWaste communicates with providers systems to automate the process of listing foods. Providers may use GUI if they choose to. Other systems are used to ensure functional work of FoodWaste (described in context)

The system update: Allergen functionality uses only Annex2 naming system.

2.2.3 Functional functional design philosophy:

The whole system: FoodWaste aims to reduce food waste while helping individuals. This is a non profit system looking for better ways to utilize planet's resources.

The system update: Allergen functionality aims to provide all the tools for the individuals to choose foods that do not harm them.

2.3 Information viewpoint

2.3.1 Information Structure and Content

The whole system: The main information that our system manipulates is user data, restaurant information and its food products. User class has all the information necessary to effectively communicate with them as well as information on their enjoyed products and restaurants. The restaurants have information about their location, contact information and mainly about their sold food products. The food product has basic information like its name, cost and description.

The system update: The update added a new Allergen class to the system. It carries the name and description information of an allergen. It is used by the User and FoodProduct classes. For user it records what the user is allergic to, and for FoodProduct it records what kind of allergens it has.

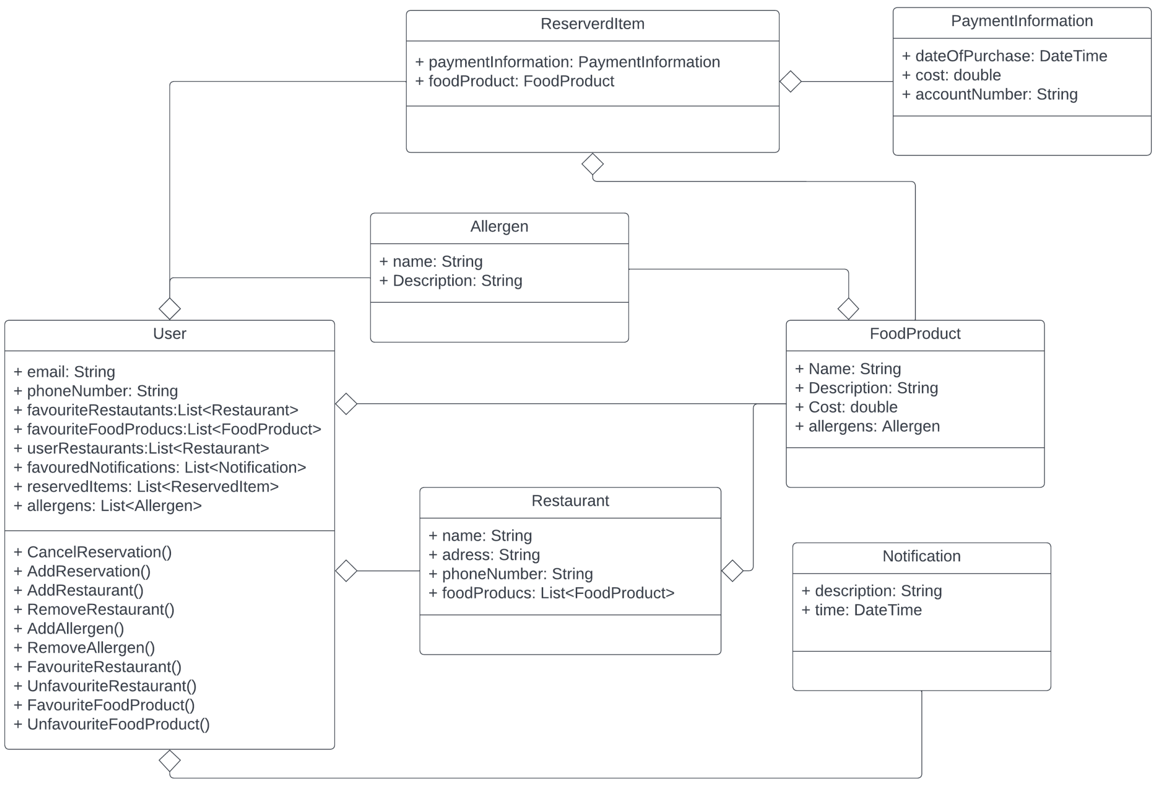


Figure 1: Class Diagram

2.3.2 Information Purpose and Usage

The whole system: Our system uses information in multiple ways. There is contact information – it is used to get into contact with users or restaurants outside of our system. This information includes email, phone number, address, etc. Then there is reservation information – it has the information about users that have reserved some products. For example, there are reserved items, payment information, food products, etc. Then we have the payment information with cost, date, account numbers. And finally, we have information about user preference. It includes favoured restaurants, favoured food products, notifications, etc.

The system update: The update added a new Allergen information to the system. The purpose of this information is to avoid any deadly allergic reaction for our users. It is used to match what our users know they are allergic to with products that can cause a reaction and to notify the user of the present danger to their health.

2.3.3 Information Ownership

The whole system: The system so far is only planned to have a single data store so no information conflicts should happen. Any choice by the user is only recorded in the database only when it is for sure final and can not have information conflicts due to, for example, infrequent synchronisation with the main database.

The system update: No changes in the number of data stores, so the system works the same as was descried above.

2.3.4 Enterprise-Owned Information

The whole system: The enterprise-owned information would be the restaurant information and their sold products. It would be a serious inconvenience to, for example, sell a product to a user that the restaurant does not has anymore, so this information should be updated frequently.

The system update: The change adds allergen information. It is extremely important to have updated and correct form of this information. Havin too many allergens than user has means we sell less food products and having to few allergens means that the user can have a dangerous allergic reaction to their health, at beast reducing their trust in our system and at worst costing us the life of our customer

2.3.5 Identifiers and Mappings

The whole system: From the systems perspective every single class that can be unique has a unique guid that allows it to be differentiated. Unique guid is given to users, food products and restaurants. From user perspective the unique guid is not shown, the system makes sure that user would only see unique products. The restaurant adds a new food product by just entering its name, not selecting it from existing ones. The possible problem of creating an entirely new food product by misspelling an existing one is not a big problem, as it should not cause any big mix ups, and considering, that there might be a lot of changing products in our system it would not be wise to have a long process of adding a new food product to a list of possible to update food products.

The system update: The change adds allergen information. From system perspective the allergen has a unique guid. From user perspective the guid is not shown. A user can select only an already existing allergen in the system, as to prevent any misspelling of allergen, which could result in a user having no waring about a dangerous allergen that he has recorded in his profile simply because the allergen was misspelled.

2.3.6 Volatility of Information Semantics

The whole system: No attention was paid when thinking about possible future changes of the system, as no work is planned to be done to this system after this update.

The system update: Same as above.

2.3.7 Information Storage Models

The whole system: The system uses a basic third-normal-form relational database, managed with PostgreSQL. It is a standard solution for a simple system like ours, providing good performance, flexibility and avoids data duplication if design correctly.

The system update: We continue using third-normal-form relational database.

2.3.8 Information Flow

The whole system: The data is created when a new user profile is created or updated with new information, and when a restaurant adds a new food product. Information is destroyed when a user deletes his profile, updates his profile by removing information or when products expiration date is reached or when a product is bought. This data is used when user is browsing food products in our system. This is where collected data is shown, combined, compared. The change in data items happen when food product state is changed.

The system update: Allergen information is created periodically when new products that have new allergens are added into a system. The allergens are not meant to be destroyed. Products and users can change to what allergens they have or are allergic to. This data can be accessed or modified by the user in the profile tab. It is made use of when browsing food products and being warned of dangerous ones.

2.3.9 Information Consistency

The whole system: Our system will use atomic operations to ensure, that any order or action will have its intended full effect rather than a half-done operation that will leave us or the user in a losing situation.

The system update: Allergens don't have any way to be inconsistent.

2.3.10 Information Quality

The whole system: The information given by the restaurants is to be trusted as high-quality information, as we expect to work with professionals. If they provide low quality information they are the ones that will suffer from it. For the users it depends mainly on how active they are. The main information that will age fast is their address, but if they use our system often they will notice, that any order is destined to go to their old address, so it is on them to update it.

The system update: Only medically proven allergens will be added to our system. No user will have to enter their own allergen – they will already choose an allergen that is deemed to be dangerous.

2.3.11 Timeliness, Latency, and Age

The whole system: Our information is held in a single data store and is always accessed synchronously in real time, timeliness, latency, and age are not significant issues. The system keeps track when product is spoiled and should be removed from store, so there is no need to constantly update information.

The system update: Allergen information doesn't age, so it does not cause any problems as well.

2.3.12 Archiving and Information Retention

The whole system: There is no useful information in our system that should be archived after it is no longer needed. It is illegal to keep personal information of a person for no reason and there is no need to know about spoiled food that we once had. Even if we would collect this information we couldn't effectively use it as we don't control the supply of it – we are supposed to take in any product that is close to expiration. Knowing that usually we usually sell out all specified products will not mean that we will be able to get more of it.

The system update: Allergen information shouldn't really be deleted or archived, so no changes from the text above.

2.4 Concurrency viewpoint

2.4.1 Task Structure

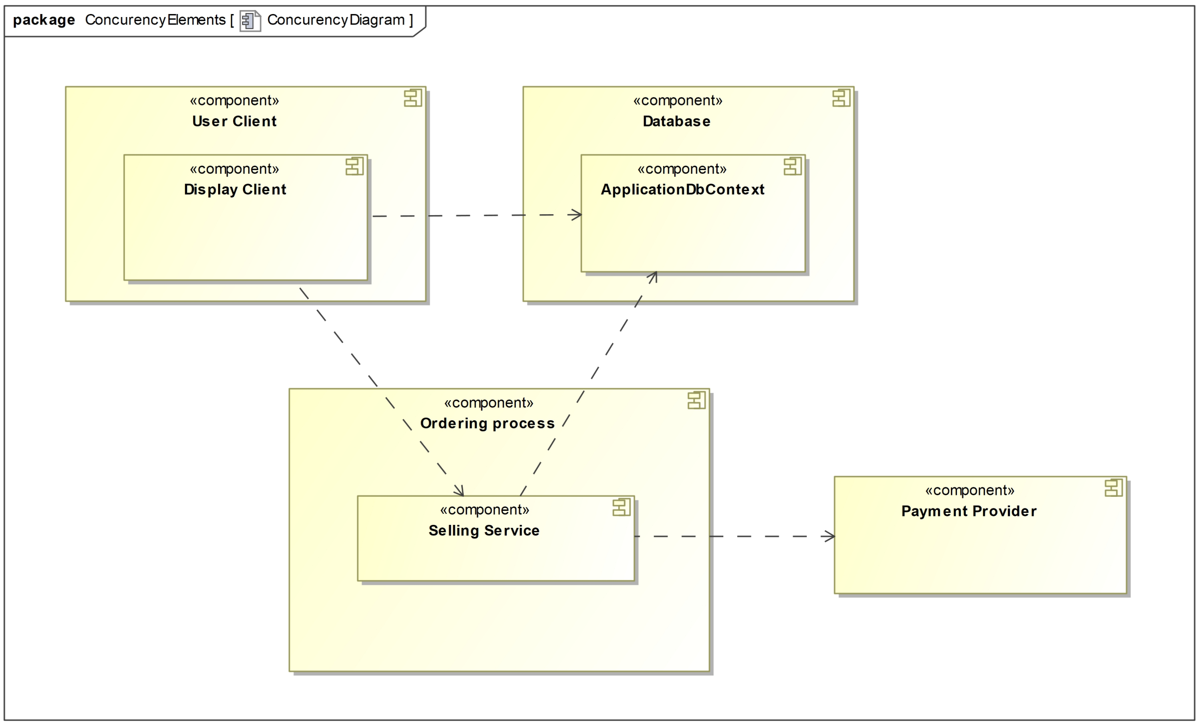


Figure 2: Concurency Diagram

As an online shop we need to be able to handle several orders at once.

2.4.2 Interprocess Communication

Since we need to handle several users at once and our website is running on one machine (server) we can make use of processor threads and let the machine work asynchronously

2.4.3 State Management

Instead of holding current order states we are going to capture events that change the state. It will make the transaction process a bit slower but in case of emergency or data loss we'll be able to easily recover. Currently slower transactions won't cause problems because transactions in most cases are pretty short and several threads will handle them fast enough.

2.4.4 Synchronization and Integrity

Each order event will have to be immediately updated to database to prevent people from ordering the same product if it's out of stock.

2.4.5 Supporting Scalability

Currently our system in on a really small scale, therefore scalability is not an issue. Majority of functions can be easily reused and extended with need.

2.4.6 Startup and Shutdown Procedures

Since we are using event based state management, there are no specific precautions when shutting down the system. Starting the system is as simple as loading up the data from the database

2.4.7 Task Failure Modes

Again, event based state management lets us track any events that happend or may have caused issues making it easy to solve problems and quickly restore previous states.

2.4.8 Reentrancy

Ordering process should be re-entrant if the system breaks during payment. Our event help with this functionality

2.4.9 Requirement specifics

Not applicable. Allergens are properties, not processes that could happen concurrently.

2.5 Development viewpoint

Because there are no future plans of expanding or working on the system in any ways, this viewpoint was only minimally considered.

The system update: Allergen in FoodWaste system is not a major architectural component. Allergen lists are attached to major components such as product and user, however, they do not interfere with current usage of these entities and their functionality (except for the alert if trying to reserve allergic foods). In the future, it should be easy to implement allergen filtering on main page (for better user experience).

2.6 Deployment viewpoint

2.6.1 Runtime Platform Required

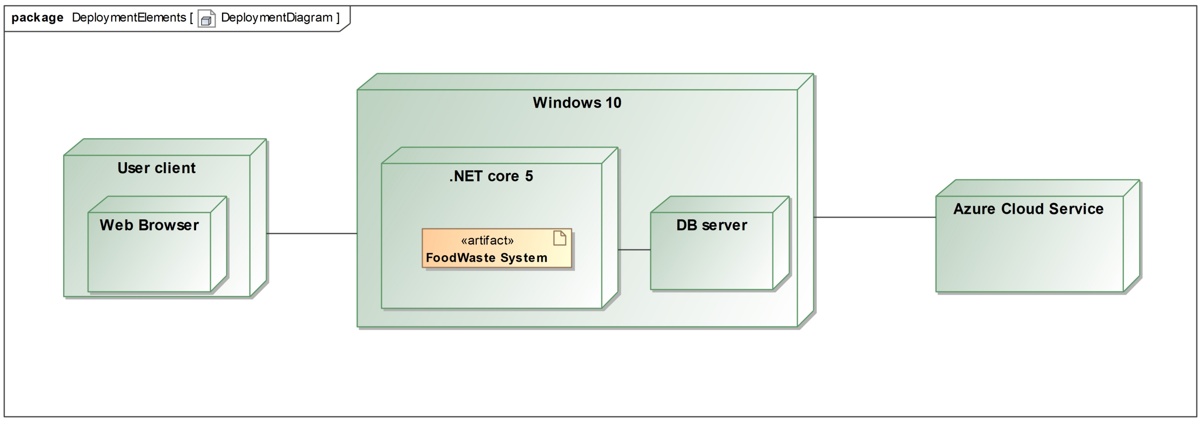


Figure 3: Deployment diagram

There are no system requirements for end users, because we are developing a web application. Since we used .NET Razor Pages framework, it is highly recommended to use Windows operating system in execution environment.

2.6.2 Specifications and Quantity of Hardware or Hosting Required

Due to our type of service we are going to need a server to host the website with sufficient amount of storage to store all the data locally. Server specifications are currently unknown we need a consultation from more knowledgeable hardware specialist. We are also going to leave some headroom for the hardware specifications to deal with heavy loads and future growth. We are getting Azure cloud service to backup our local data to protect against possible physical damage to server or in case of power outage

- Server components - TBD
- Server Memory - TBD

2.6.3 Third-Party Software Requirements

Since we are using .NET core we are going to use available libraries to make our work easier

- Xunit 2.4.1
- PostgreSQL 5.0.0-rc2
- MatBlazor 2.8.0
- Coverlet.Collector 3.0.2

2.6.4 Technology Compatibility

Due to .NET core used libraries should be backwards compatible but in case there are runtime error, we have provided versions in previous segment

2.6.5 Network Requirements

We are primarily going to use our local DB server to store data but we will want to backup the said data to Azure cloud, regular ISP speeds of 1 gigabit should be sufficient for the startup but we will want to upgrade down the road to support heavier client traffic.

2.6.6 Network Capacity Required

Based on the recommendations for our local database server Azure cloud should be capable to handle the same amount of data (TBD)

2.6.7 Physical Constraints

We don’t have much physical constraints, we are going to need a small server and we are using cloud service for backup, so no need to rent remote area for backup database. We might want to expand with this later, to have a remote server in case of emergency, to keep our services always running.

2.6.8 Requirement specifics

Allergens are properties, not major components that change the program environment or its execution. Deployment will not differ from other feature deployments (A/B testing in production environment for part of the users and if there are no incidents - major release).

2.7 Operational viewpoint

2.7.1 Instalation and Upgrade

The whole system: Our system is planned to run on our customer machine, so the installation and upgrade procedures should be relatively easy, as we only need to worry about limited hardware specification. The installation and upgrades are not fully automated, no request was given for such a convenience. Our system will be installed as a fresh product, no other similar system operated fully before this one, there are no software that needs to be relieved. But any future changes will have to be upgrades to the running system rather than completely new installations, as the system will have to work with old users as if nothing changed.

The system update: The implemented changes do not radically change the system and will allow for an easy upgrade. The hardware requirements are the same, only some additional information is logged in the database, so no radical changes to existing record are needed, only an addition of an empty field to all existing food products and to the allergens of all users.

2.7.2 Functional Migration

The whole system: As the old version of the system was minimally used, our system can be migrated using the big bang approach, as no extensive use is planned in the future. Due to having a limited operational window, there is nothing to lose in case of an unsuccessful migration.

The system update: The system is used in a limited way, so there is no risk of damage from unsuccessful migration. We will use the big bang approach.

2.7.3 Data Migration

The whole system: The system did not operate with any actual users, neither before or after our work on it, so any existing data on it is fabricated and purely for testing purposes and has no need to be migrated. Even if the data is required to be migrated it can easily be done manually due to its small size.

The system update: The system data had a slight change due to addition of allergens, but just like before, no actual user has used the system, so any migration currently has no huge meaning and can be done manually or not even done at all.

2.7.4 Operational Monitoring and Control

The whole system: A system with monetary transactions would benefit from monitoring and control operations. There were monitoring functionalities in the system given to us but there were no requests by the customer to develop control operations or expand on the monitoring in the system, so currently no control operations exists in this system.

The system update: The update was the addition of allergens to the system user profile and food products. No in-depth monitoring or control operations over this update is required nor was requested, so none were developed.

2.7.5 Alerting

The whole system: No alert system is currently present in the system and no request to develop one was given. Any unforeseen events occurring are meant to be reported to the developer team.

The system update: The update was the addition of allergens to the system user profile and food products. Although this means that there are more moving parts that can fail, no alert system has been requested ant there is no base that we could easily build upon.

2.7.6 Configuration Management

The whole system: Because our database does not run on the local machine, it has a separate configuration from the system. Other than that there are no foreseen requirements for handling different configurations – currently only one is used.

The system update: No new configurations have been made with the update.

2.7.7 Performance Monitoring

The whole system: No additional performance monitoring is performed outside of viewing how the system runs on the used machine. The Visual Studio IDE can give some statistics of the systems performance during run time, no other way of collecting performance information was requested by the customer.

The system update: The update focused on allergens, no work on system monitoring was done, nor did it had a great effect on the systems performance that would warrant such work to be done.

2.7.8 Support

The whole system: We discussed how technical support could be implemented in the second laboratory work. We did not choose to implement that change into the system, but if any need arises in the future the basic idea of it is already though through.

The system update: We wanted to be as clear as possible with the allergens in the user interface itself. We believe we have made an easily comprehensive system and that no extra customer support is needed.

2.7.9 Backup and Restore

The whole system: Our system uses only a single database, so no need to prepare for a complicated recovery over multiple data stores. Making a backup would usually take place every night, when data changes the least, but with current usage of the system (which is minimal) the backup process could happen every weekend. The backup copy would be saved in a separate drive, not in the active one in the system.

The system update: No changes were made to the process of backing up data.

2.7.10 Operation in Third-Party Environments

The whole system: Our system uses a database hosted on a remote machine. Although for a small-scale product, like ours currently is, it is not the most convenient solution, our customer was rather considerate of its long-term advantage and gave the system to us in such a state. We had some difficulties because of this decision as we struggled to even run the project properly as we received little detailed support from the client about the connection process. But we had received no request to change the system to use a local database, so it has stayed the way we received it.

The system update: The update has not added any new third party environments nor removed any old ones.

3 Perspectives

3.1 Accessibility perspective

Currently FoodWaste does not support accessabillity standarts that it would enjoy giving to users. Here are the plans:

3.1.1 On Concurrency view

The whole system: Not applicable. Concurrency is internal and end user does not know anything about it.

3.1.2 On Deployment view

The whole system: Not applicable. Deployment process does not change.

3.1.3 On Context view

The whole system: Team plans to support versions that help people with different disabilities to use the FoodWaste.

3.1.4 On Functional view

The whole system: Buttons/voice commands will be used to enter accessibility mode. There the most basic features will be available with the potential to include all features.

3.1.5 On Information view

The whole system: Not applicable. Accessibility support is only front end.

3.1.6 On Operational view

The whole system: Not applicable. Processes do not change.

3.2 Availability and Resilience perspective

3.2.1 On Concurrency view

The whole system: The system should have redundant hardware in case of the used one failing. Not being able to sell our food products would mean that a lot of them would go to waste and we, as well as our partners, would lose out on profits. Because we do not have a local database we should look into having multiple providers to host our data in case one of them would fail to provide their services. Our system should also have an automated failover in case the hardware fails. Best case scenario would be that the emergency system could take over the work of the failed system.

3.2.2 On Deployment view

The whole system: The redundant hardware should always be ready for deployment and up to date. During the deployment the system should be autonomous enough to know which database is ready to use and in what state were the users left when the system failed.

3.2.3 On Context view

The whole system: Because we do not use a local database, we could have two remote data storage providers handle our data. This should be taken into consideration as one database will be the active one, meaning if it fails the other database will not be up to date. We should prepare for this by either having extremely frequent database updates for the redundant database or by having a reasonable system that could estimate how many products could be left in our inventory based on how fast they usually sell out and what was the last time the database was updated.

The system update: As allergens are important to the health and wellbeing of our users, therefore is we would lose any information that they have given us we should have notifications in place that would warn the user that they should double check the allergen tab in the product page to make sure they are buying a product that is safe for their consumption.

3.2.4 On Functional view

The whole system: Unfortunately, due to the nature of our system, having a network connection and a working system is essential if users want to use our system as it was intended. If the system would totally collapse or the user would lose internet connection for a long period of time the user would be left without any way to interact with the system. If we were to allow some accessibility to the user when they are offline to, for example, look through their profile or order history, it would be a violation of user data security and should not be allowed.

3.2.5 On Development view

The whole system: Not applicable, the system would restart fully if a failover would happen on the system hardware or it would wait until connection to the backup database would be established if database storage provider would go offline.

3.2.6 On Information view

The whole system: As was written before, we do not use a local database. We should use two database storage providers in case the provider goes offline for whatever reason. The system should be constantly communicating with the database so that it could tell when the database is inaccessible as soon as possible. In theory, transitioning to the other database should not take too much time. The backup database should be updated every night when there are minimal number of users using the system, as the process should be lengthy.

3.2.7 On Operational view

The whole system: In case of a system hardware failure or database storage supplier going down, the system should be ready to make a swift diagnostic of what services are unavailable and what course of action should be taken. The entire process must be tested beforehand.

3.3 Development Resource perspective

3.3.1 On Concurrency view

The whole system: Concurrency is hard to manage therefore we might need additional people to test and implement properly working system

The system update: Not applicable, allergens are immediately updated on users profiles, no need to concurrently handle it - no additional resources required

3.3.2 On Deployment view

The whole system: We need people that understand and can handle server hardware. We are not qualified enough to select equipment that would handle our needs meaning additional specialists are required to sort out the hardware and networking

The system update: No need for additional resources, system admin will be able to deploy the update as soon as its ready

3.3.3 On Context view

The whole system: we need time to prepare correct relationships and dependencies for the system and its environment

The system update: we need time and people to prepare new relationships

3.3.4 On Functional view

The whole system: No need to additional resources

3.3.5 On Information view

The whole system: We need a large staff of specialists to prepare sophisticated information models and maintain them

3.3.6 On Operational view

The whole system: We need people to operate the system and administere it. Install or upgrade it

3.4 Evolution perspective

3.4.1 On Concurrency view

The whole system: FoodWaste does not have any special implementations that prohibit use of more in depth use of concurrency than the default one that .NET provides.

3.4.2 On Deployment view

The whole system: Currently CI does not have all of the important features that it will in the future, however, there are no roadblocks to improve it.

3.4.3 On Context view

The whole system: Currently FoodWaste does not provide courier services. This is probably biggest feature that is going to be made if the idea proves to be successful.

The system update: Allergen functionality helps FoodWaste to become more user friendly - it is a step to increase user base.

3.4.4 On Functional view

The whole system: Many features of current FoodWaste may change in the future. Developers try to follow many of the best practices to make new and changing features easy to implement in the future.

The system update: There will be more allergen features that are planned. However, the team does not see this component needing more of the development power in the near future.

3.4.5 On Information view

The whole system: FoodWaste tries to keep everything simple, however with new features it is obvious that changes will have to be made. Being able to switch tiers/products in Azure/AWS is going to be beneficial.

3.4.6 On Operational view

The whole system: As FoodWaste will become bigger project and more users will depend on it - more rigorous systems for operations will be implemented.

3.5 Location perspective

3.5.1 On Concurrency view

The whole system: Because our database is accessed remotely, the latency can be slightly higher than normal. Knowing that the same record from it won't be accessible to multiple users combined with the latency can result in long query times. Therefore, the system should be able to determine on its own how many products it can sell to the user before noting the change in stock in the database when the needed record is not occupied. This means that the system should always have a limit on the quantity of a single product that the user can purchase at a given timeframe.

3.5.2 On Deployment view

The whole system: Because the only separate part of our system is the database, the disparity is not a big issue. We should take into consideration to have our queries and their results as short as possible, as the database is going to be used constantly.

3.5.3 On Context view

The whole system: The database storage providers should be based in Europe, as it would ensure we have minimal latency and reliable availability

3.5.4 On Information view

The whole system: The main database should be used to update the backup database. The update would take place every night when the system experiences minimal user activity. If a provider were to go offline the system should be able to swiftly diagnose the disconnect and start using the backup database. Of course, if the disconnect takes places in the evening, the data will be inaccurate. The system should have access to some statistical measurements of how fast certain food products are bought on what times of day and roughly assess how much food products should be left in stock and could be sold while keeping tabs of current transactions to update the main database when it comes back online. It is certain that separate providers will be able to synchronize the database information on their own, so during the night the system should compare the stock of main database to the stock of the backup one and issue update commands on its own.

3.5.5 On Operational view

The whole system: Because we are using a provider that specializes in database hosting, we believe that they can be trusted to deliver their best efforts to keep their services running. If any unforeseen event occurs, we trust that they will recover within a couple of hours.

3.5.6 On Functional view

The whole system: Not applicable, functional view is presented independently of real-world location concerns.

3.5.7 On Development view

The whole system: Not applicable, development in the system is planned to be done in a single location.

3.6 Regulaiton perspective

3.6.1 On Context view

The whole system: Our system should be able to generate monthly reports to an accounting company so that they could work on salary calculations, audit reports, etc.

3.6.2 On Functional view

The whole system: Because we are dealing with food that is near its expiration, we should take we partner with companies that can ensure, that their provided food products comply with “Lietuvos Respublikos maisto įstatymas”, document number VIII-1608.

3.6.3 On Information view

The whole system: Because we operate in Europe, we have to be able to give a request to our users about their collecting and managing information about the, give a report to our users about the data we have collected on them, give them the possibility to delete this data, to notify them within 72 hours of data leaks, ant not to hold onto any user information that we do not need to hold on to.

The system update: We will have to ask for a permission to collect and manage data about user allergies. Allergens will be deleted on request of the user or on user profile termination.

3.6.4 On Concurrency view

The whole system: Not applicable, this perspective has little or no impact on the Concurrency view.

3.6.5 On Development view

The whole system: Not applicable, this perspective has little or no impact on the Development view.

3.6.6 On Deployment view

The whole system: Not applicable, this perspective has little or no impact on the Deployment view, as we do not have any sort of health or safety hazards because of our used hardware.

3.6.7 On Operational view

The whole system: We need to be able to automatically give out SLAs to our users to be able to provide our services. This includes basic services, cookie usage, etc.

3.7 Security perspective

To review our systems security we will be using OWASP model

3.7.1 Injection

Injections are not concerning us due to our program structure. Data quaries are deep in the backend code hidden behind controllers and are protected by our data validation.

3.7.2 Broken Authentication

Each input field is validated before it's accepted.

3.7.3 Sensitive Data Exposure

Our system is currently on a really small scale, therefore we use almost no external APIs which reduces the risk of exploiting insecure data transmissions

3.7.4 XML External Entities

Our system does not allow to upload any files because there is no need for it, we simply do not have this functionality therefore this type of attack is irrelevant

3.7.5 Broken Access Control

Our automated UI tests did not show any access control flaws or configuration errors. Regular users have no acces to sensitive files, systems or admin settings.

3.7.6 Security Misconfiguration

All integration tests have passed therefore and currently no misconfiguration is apparent

3.7.7 Cross-Site Scripting

We use almost no external APIs, use data encryptions and validate inputs to minimize the risk of attacks

3.7.8 Insecure Deserialization

System saves files to a database with quaries, we have no need to serialize / deserialize data. Not applicable

3.7.9 Using Components with Known Vulnerabilities

We use almost no external APIs, there is always a risk that some security flaw was missed, as far as we are aware we do not have APIs with known vulnerabilities

3.7.10 Insufficient Logging and Monitoring

We use event sourcing which helps us to monitor and log everything that happens within the program

3.8 Usability perspective

3.8.1 On Concurrency view

The whole system: Not applicable. Concurrency is internal and end user does not know anything about it.

3.8.2 On Deployment view

The whole system: Not applicable. Deployment process does not change.

3.8.3 On Context view

The whole system: FoodWaste looks forward to implement system that enables users to easily achieve the functionality provided. Usability is highly strived for in the process of developing FoodWaste.

The system update: Allergen functionality will improve the usability of FoodWaste tremendously for people with allergies.

3.8.4 On Functional view

The whole system: FoodWaste will use A/B testing to determine which features are the best according to users.

3.8.5 On Information view

The whole system: Efficient use of data enables FoodWaste to operate quickly - all users are impacted by this.

3.8.6 On Operational view

The whole system: Not applicable. Processes do not change.

4 Architectural styles

4.1 Data-centred

First and foremost, it's hard to deny that our main architectural style is data-centred. Because we sell food products the main goal of our system is to manage the data about them and allow the users to conveniently look it up and for the food product providers to conveniently update information about the food products. There fore the database is the heart of our system, accessed continuously thorough the functioning of our operations. If our database is for some reason inaccessible the entire system cannot work, no matter how much processes can be executed independently by the client components. This leads us to our second

4.2 Component-based

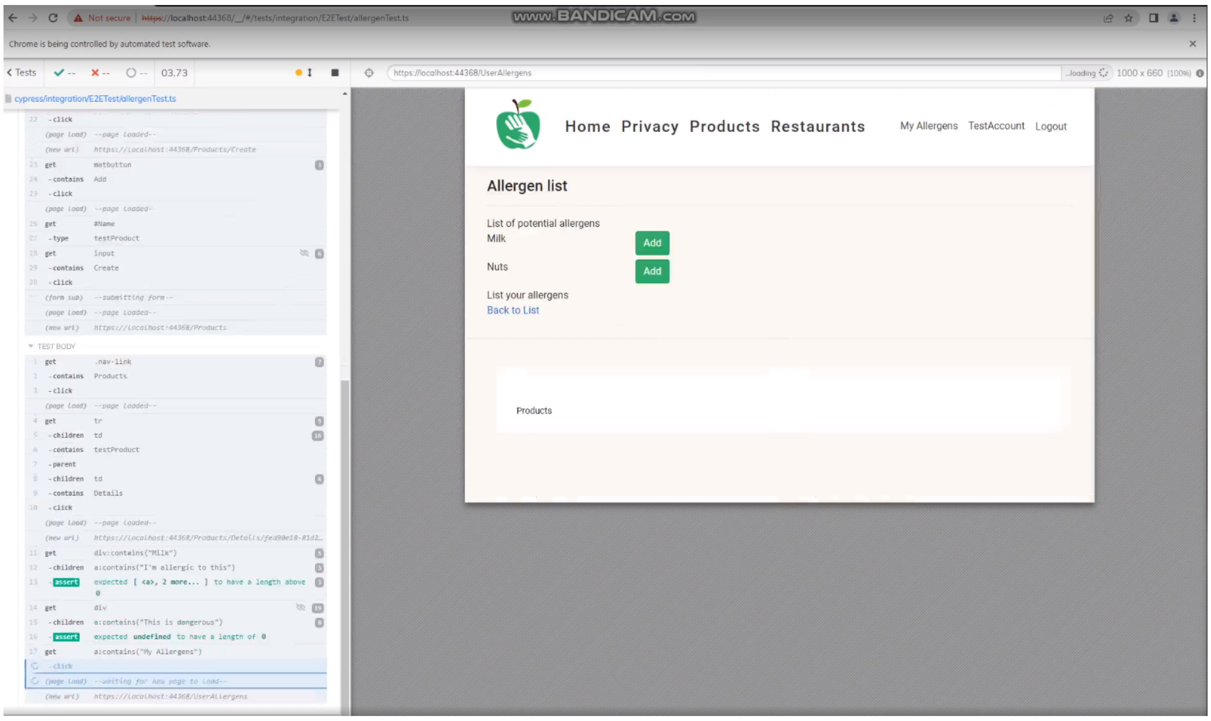
Because our system expects multiple kinds of users to use our system in different, but similar ways, our architectural style is also component-based. A lot of processes require their unique components, like payments, accounting, monitoring, food product update, profile update, etc. Therefore, we had to utilize component-based architecture to create a system that would be able to fulfil these varied requirements while allowing to reuse some code.

5 Testability

For this iteration one of the requirements to be implemented is the ability to use allergens in the system. After implementing the system changes, we will have to test it to make sure that it is working properly. The test case should cover changes in the system, both frontend and backend, as well as changes in the database. For this an UI test case written using Cypress framework in typescript will be used.

Test case steps:

- User logs in with valid credentials, creates product with new allergens.
- User checks that item details, the system shows that there is no dangerous allergens.
- User goes to the "User allergens" tab, and check that he is allergic to one or more allergens from the newly created product.
- User check the same product details, the system shows that product contains dangerous allergens.
- User deletes created product.
- User logs out.



```
Not secure | https://localhost:44368/__/#/tests/integration/E2ETest/allergenTest.ts
Chrome is being controlled by automated test software.

cypress/integration/E2ETest/allergenTest.ts
 -click
(page load)   --page loaded--
(new url)     https://localhost:44368/Products/Create
get           matbutton
 -contains    Add
 -click
(page load)   --page loaded--
get           #Name
 -type        testProduct
get           input
 -contains    Create
 -click
(form sub)    --submitting form--
(page load)   --page loaded--
(new url)     https://localhost:44368/Products
TEST BODY
get           .nav-link
 -contains    Products
 -click
(page load)   --page loaded--
get           tr
 -children    td
 -contains    testProduct
 -parent
 -children    td
 -contains    Details
 -click
(page load)   --page loaded--
(new url)     https://localhost:44368/Products/Details/fed90e18-81d2…
get           div:contains("Milk")
 -children    a:contains("I'm allergic to this")
 -assert      expected [ <a>, 2 more... ] to have a length above
              0
get           div
 -children    a:contains("This is dangerous")
 -assert      expected undefined to have a length of 0
get           a:contains("My Allergens")
 -click
(page load)   --waiting for new page to load--
(new url)     https://localhost:44368/UserAllergens
```

Figure 4: Cypress UI test case

5.1 Aspects tested from the frontend:

- Create product form with allergens
- Create allergens within create product form

- User allergens tab functionality
- Product details page with allergens information and warning when user is allergic to allergens

5.2 Aspects tested from the backend:

- Product creation/deletion with allergens
- User allergen adding/removing
- Warning messages when allergens are in the product.

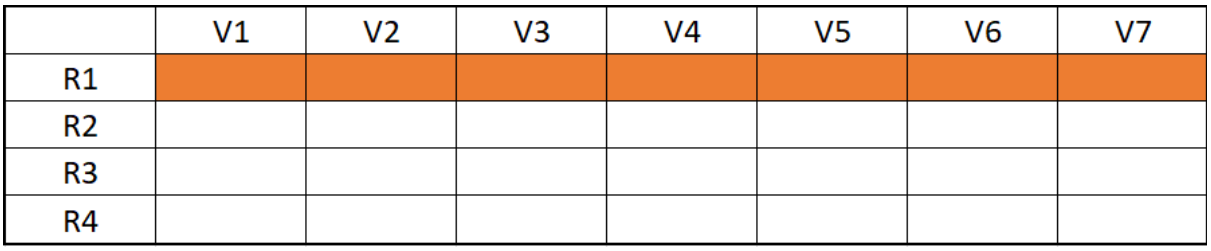
6 Traceability

6.1 Requirements

- R1 Allergens tags on food products
- R2 Remove reservation
- R3 Restaurant might have multiple addresses
- R4 Notifications for selected restaurants

6.2 Viewpoints

- V1 Context viewpoint
- V2 Functional viewpoint
- V3 Information viewpoint
- V4 Concurrency viewpoint
- V5 Development viewpoint
- V6 Deployment viewpoint
- V7 Operational viewpoint



	V1	V2	V3	V4	V5	V6	V7
R1	🟧	🟧	🟧	🟧	🟧	🟧	🟧
R2							
R3							
R4							

Figure 5: Traceability matrix