Group assignment spec.: T-Low food delivery service


Figure: Pixabay

Table of Content

1 INTRODUCTION

Food delivery services have appeared profitable business in the recent years. The consumers want tasty food but live a busy life, therefore they value the opportunity not to leave home or workplace to get something delicious to eat. Besides, smartphones and communication infrastructures have enabled easy yet powerful user interfaces to not only customers but also restaurants. Therefore, the business model of food delivery has evolved from pizza taxis to services that enable customers to conveniently order food from a selection of restaurants. For restaurants, this also opens a new channel for sales.

In this assignment, you design T-Low, a food delivery system powered by recent technology. You get to try to design a large-scale system even with some IoT functionality.

1.1 TERMS AND DEFINITIONS

Courier: A person delivering food items.

Food bag: A bag utilized to deliver food items from restaurants to customers. The bag is generic and provided to each courier by the operator.

Food item: An item that can be ordered. This can be a single piece of food (e.g., hamburger) or a meal of multiple pieces (e.g., hamburger, fries and a drink), predefined by the restaurant.

Operator: the company operating the system.

1.2 REFERENCES

Haikala, I. & Märijärvi, J. (2006). *Ohjelmistotuotanto* (11. painos). Talentum.

Kruchten, P. (1995). Architectural Blueprints — The "4+1" View Model of Software Architecture. *IEEE Software* 12(6), 42-50.

1.3 VERSION HISTORY

Ver	Date	Editor	Description
1	2024-03-13	DH,MF,PN,PK	First version.

2 SYSTEM REQUIREMENTS

This section describes the high-level functional and non-functional requirements expected from the system. The functional requirements are grouped after each user type.

2.1 RESTAURANTS

Update food item menu. The list of available food items changes repeatedly. The restaurant must be able to update this menu whenever they find it appropriate.

User interface. Any interactions take place using a) the T-Low mobile app for restaurants, b) browser interface or c) an order management system used internally by the restaurant.

Notification from orders. Whenever a customer orders something from a restaurant, there must be a notification to tell the restaurant what has been ordered.

Order received acknowledgement. When a restaurant receives an order, it will acknowledge this. The information will be visible to the customer.

Estimate of finishing. The restaurant needs a feature to provide an estimate when the food will be ready for pickup. The estimate will be delivered to both the customer and the courier that accepted the order. The restaurant may or may not use this feature.

Order ready for delivery. The courier receives a notification whenever there is a delivery for pickup, announced by the restaurant. This applies only to the deliveries accepted by the courier.

****Excluded** from requirements:**

(1) The creation of user accounts
(2) Addition or deletion of restaurants in the system, editing of restaurant information (except food menu)
(3) Exceptional situations where restaurant faces an issue that prevents food preparation, such as a broken pizza oven
    - In these cases, the customer service would refund the customer and not via this system

2.2 CUSTOMERS

User interface. There must be both a browser interface and a phone app for the customers.

Food ordering. There must be a user interface to order food items from the nearby restaurants registered to the service.

Payment. The payment occurs via an external service provider, such as PayPal or MobilePay.

Location tracking of courier. During delivery, the customers want to see where their food is going.

Expected time of delivery. After ordering, the customer receives an estimate when the food will be delivered. The estimate might be updated when the food is ready or when it has been picked up and is on its way.

****Excluded** from requirements:** the creation of user accounts.

2.3 COURIERS

User interface. The couriers must have a phone app to access the system.

Start and end of shift. When a courier starts or ends a shift, they report this.

See available deliveries. The couriers can view which deliveries are available in the area.

Accept delivery. The couriers select by themselves which orders they accept for delivery.

Order pickup. When a courier starts a delivery, they report this using the app. The report must enable the system to track which food items are being delivered by which courier and when the delivery has started.

Delivery completed. The couriers must acknowledge the delivery of an item to the customer.

Statistics. This enables the courier to view how they have performed compared to others. This must include the distance travelled, the number of food items delivered and the number of separate deliveries to separate addresses. The user must be able to view this for a particular shift or a particular week.

Location of couriers.

****Excluded** from requirements:**

(1) The creation of user accounts
(2) Exceptional situations where the courier fails to deliver the ordered food items, e.g., due to an accident
    - In these cases, the customer service would refund the customer and not via this system

2.4 OPERATOR

Reporting data storage. The operator needs functionality to monitor how the business is doing. For this, there must be a scalable data storage that enables multiple data types to be collected to enable analysis. Regarding this, the more detailed requirements are explained in the following paragraphs.

Restaurant sales. The reporting data storage must cover the sales volume of each restaurant. This includes at least the number of food items sold, the revenue generated for the restaurant and the revenue for the operator, reported for a time period.

Throughput of couriers. The reporting data storage must include data to indicate courier efficiency. This covers, for each courier, the number of delivered food items, the number of deliveries (separate deliveries to separate addresses), the total distance travelled and the total working hours during a certain time period.

Careful food bag handling. The customers prefer their food intact without all ingredients messed up. The system should be able to collect acceleration data for future features to monitor proper handling of food bags. The food bags contain an acceleration sensor that collects acceleration measurements using Bluetooth LE. The reporting occurs via the phone app of each courier.

****Excluded** from requirements:**

- The APIs (i.e., interfaces) and clients the operator utilizes for data analysis
    - You should still include the required backend and the logic of data collection

- The functionality inside of the acceleration sensor devices, as these are built externally.
    - You should still include the logic of the data collection from the sensors.

2.5 NON-FUNCTIONAL REQUIREMENTS

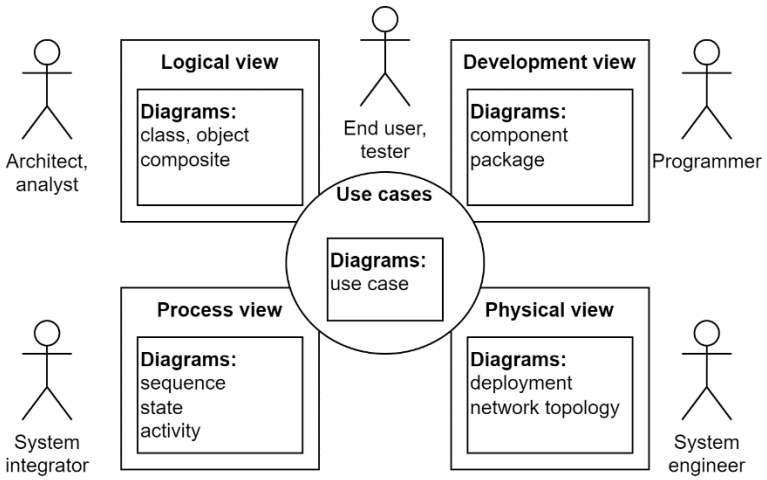
- Restaurant- and food-item-related information is stored in the cloud instead of some system operated in the restaurant.
- The backend must be resilient and contain no single point of failure.
    - I.e., “backend” excludes the apps and devices used by either customers, restaurants or couriers.
- Scalability/load balancing: the number of simultaneous users varies a lot and can be 100.000 at maximum while there are only a few thousand users during the quiet hours.
- Strong authentication must be supported, using existing authentication providers such as Facebook, Google etc.
- The design should enable the system to evolve.

3 ARCHITECTURAL DESCRIPTIONS

This section describes what is expected from the architectural descriptions.

3.1 4+1 MODEL

The representation will follow the 4+1 model (see the figure below; introduced by Kruchten (1995)). To apply 4+1, we start by introducing use cases that we exploit to derive the requirements. Based on these, the logical, process, deployment, and physical view are considered, each with appropriate UML diagrams.



Adapted & re-drawn from Kruchten (1995)

There are other views about the 4+1, such as this video modified from Kruchten by Birgit Penzenstadler (Chalmers University): https://www.youtube.com/watch?v=r8ucofiI8vY

For further reading, see the book “UML2 and the unified process”.

3.2 DOCUMENTATION STRUCTURE AND CONTENT

The *final* documentation must contain at least the following sections, identified with an appropriate heading:

1. (Brief) introduction
    - Objectives of the document
    - Working environment, i.e., what tools you have used
    - Total count of words in the document (with resolution of 100 words)
2. Requirements and use cases
    - Explained later in this doc
    - Use case diagram expected!
3. The four views of 4+1 model
    - Explained later in this doc
    - For each view, at least one UML diagram expected!
    - Explain each diagram as text

Furthermore, the following items must be included:

- Identification and context
  - Which course
  - Authors
- **Justifications, i.e., the most important architectural decisions and their rationale**
  - Must be highlighted in *italics*!

In the assessment phase, the course personnel will review the justifications in particular. Please remember also to check your architectural decisions against any quality-related or other non-functional requirements.

Please emphasize clear expression and quality over length – as long as the documentation explains what is necessary. That is, length is less important than the actual content, and too lengthy and ill-structured documentation may even decrease your points. **The absolute maximum length of the *final* documentation is 4000 words,** which is not much. **The final documentation must contain everything, including what is required for the intermediate version!**

3.3 REQUIREMENTS AND USE CASES

3.3.1 Use cases in general

Use cases should represent the user's interactions with the system. In addition, there might be a few important internal processes that can be auto-triggered. Use cases describe what the user needs to do, what they are trying to accomplish, and how the system responds when they use the software.

For more information about use cases, you can search the web. For example, the following websites seem informative:

- https://www.usability.gov/how-to-and-tools/methods/use-cases.html
- https://www.ecs.csun.edu/~rlingard/COMP682/use_case_template.htm

You should write use cases in a tabular format. Below, you can find an example structure, adapted from Haikala & Märijärvi (2006). The format is not strict, but you should include at least the items currently included.

Identifier and name	UC-1 Some descriptive name For name, you can optionally put each use case in a (sub)section and indicate the name in the related heading.
Actors	List here whichever are the external actors that participate in the use case. The actors can be users and/or *external* systems or software.
Preconditions	Describe the preconditions required for the use case to happen.
Main success scenario	Explain here what happens when the use case succeeds. Often, there is a single main actor that determines the viewpoint to be applied. Usually, the main success scenario is straightforward and contains almost no conditions. Therefore, do not try to give an exhaustive description of all possible combinations of what the main actor could decide to do, but rather give a single success scenario. Still, you should use exceptions for any exceptional situations.
Exceptions	Give a list of exceptions that may affect the main success scenario. Usually, an exception indicates that something is wrong or in an unexpected state. Explain what causes the exception and what happens then. Give each exception an identifier and refer to it in main success scenario to indicate when the exception may occur.
Expected result	This is the result you expect when the main success scenario finishes.

3.3.2 Use case diagram

The set of use cases must be visualized with a use case diagram. The diagram shows the relationship between the actors and the use cases but not the structure of the use cases. **Draw exactly one (1) use case diagram.**

For guidelines, please search the web. For example, the following page provides an introduction: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

3.3.3 Required use cases

To reduce your amount of work, you only write a subset of the actual use cases. This includes the following (primary actor in parentheses):

- Updating food item menu (restaurant)
- Order received acknowledgement (restaurant)
- Order ready for delivery (restaurant)
- Food ordering (customer)
- Location tracking of courier (customer)
- See available deliveries (courier)
- Accept delivery (courier)
- Order pickup (courier)
- View courier statistics (courier)

!!! NOTE: The list above only applies to use case text and use case diagram! Regardless of the list, you must consider all the requirements in this document in other aspects of the 4+1.

3.3.4 Other requirements

In this assignment, use cases are the primary method for the documentation of requirements, but not everything can be documented in use cases. Therefore:

- As this specification contains a set of requirements, refer to the specification (i.e., this doc) where appropriate
    - Avoid repeating the requirements as such to shorten your document
- Complement the use cases with additional requirements, written by you, as you see appropriate
    - Aim for a consistent design document that contains what is essential for the architecture

3.4 THE FOUR VIEWS

3.4.1 Logical view

Logical view supports the description of *functionality*. In the end, UML class diagrams are expected here.

In 4+1, the logical view is modeled early, which means the designers do not yet know accurately what physical components the system will contain. Therefore, use the class diagram as a tool to analyze what concepts will appear in the system and how these are related. The class diagram can remain abstract. That is, there is no need to consider which physical component holds each item of the class diagram or how the classes (or concepts) will interact via the communication network.

For an actual system, the class diagram would be updated to match the component structure and the internal modules of each component. However, this assignment focuses on system-level concepts and therefore skips the detailed application of class diagrams.

3.4.2 Process view

The process view describes the communications between components and processes of the system. The focus is on the runtime behavior of the system. Most often used diagrams in the process view include:

- sequence diagrams: https://circle.visual-paradigm.com/category/uml-diagrams/sequence-diagram/
- state diagrams: https://circle.visual-paradigm.com/category/uml-diagrams/state-machine/
- communication diagrams: https://circle.visual-paradigm.com/category/uml-diagrams/communication-diagram/
- activity diagrams: https://circle.visual-paradigm.com/category/uml-diagrams/activity-diagram/

Please focus on the interaction of the components from the system-wide viewpoint. Please also describe some of the internal logic if this is essential to understand the overall functionality.

3.4.3 Development view

The development view is described with the UML component diagram. You can find information about the diagram type here:
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

The component diagram divides a system into components. Sometimes, it is difficult to make the distinction whether an entity should be described as a class or a component:

- With a class, one can demonstrate both the internal functionality as well as required/provided interfaces.

- On the other hand, the components of UML component diagrams describe the required and provided interfaces and how these enable interaction. In other words, the emphasis is on the required and provided interfaces and interactions between the components.

In the architecture documentation, we thus expect the UML component diagram to be less detailed compared with the class diagram. The component diagram should document both client and server functionality as well as the required and provided interfaces regarding external services if any. Please consider quality requirements and provide justifications, too.

The component diagram should not yet refer to any cloud products or similar, because these come only in physical view. That is, the component diagram focuses on the logical structure of the system.

Here are some additional tips and guidelines to help you draw your diagram:

- Avoid using monolithic “system” or “backend” component that hides inside other components. Try to visualize these larger components as smaller, connected components.
- Be careful you do not include components that are more fitting for the physical view in the component diagram, such as an API gateway.
- You can also add service components for the payment and authorization services in the diagram, though connectors from appropriate components to these services are sufficient.

3.4.4 Physical view

The physical view is described with deployment diagrams that visualize the hardware nodes or devices of the system, the links of communication between them and the placement of software files in the hardware. That is, the deployment diagram describes:

- What cloud platforms or hardware components (e.g., mobile phones or servers) there are; also called “nodes”
- What software components ("artifacts") run on each node (e.g., web application, web service, message broker, database).
- How the components are connected.
- For more guidelines, visit: https://creately.com/blog/diagrams/deployment-diagram-tutorial/#:~:text=How%20to%20Draw%20a%20Deployment%20Diagram

Here are some additional tips and guidelines to help you draw your diagram:

- You do not need to separate devices into separate nodes based on the technology used. For example, if you have a “Customer Device” node, you don’t need to make separate ones for PC, iOS and Android.
- We encourage you to explore pre-existing cloud technologies that you could base your implementation on, such as AWS or Microsoft Azure. This is not required, but they might help you realize what artifacts are available.

3.5 UML TOOL: ECLIPSE PAPYRUS

To draw the UML diagrams, you must use Eclipse Papyrus.

Eclipse Papyrus is a powerful UML modeling tool that creates an actual metamodel for the diagrams. The metamodel enables you to re-use the concepts across multiple diagrams. Compared to simple diagramming tools, the learning curve of Papyrus is steeper but pays back when the size of the system grows.

The university staff preserves the right to use your models and diagrams as research material.

Get Papyrus here: https://www.eclipse.org/papyrus/

3.5.1 Deployment diagrams

Papyrus is somewhat strict in the modeling of deployment diagrams. In particular, it doesn't let you put certain elements inside each other or create too many nested elements.

To indicate a deployment, you can use the "deploy" relationship. You would create such an arrow from "artifacts", which represent deployable pieces of software, to execution environments or nodes. The diagram below shows an example where some legacy software is deployed to a virtual machine in the cloud. ***Please note*** that this is merely an example and you should consider what is the best approach for T-Low – not necessarily virtual machine hosting.