Requirements Document

Context

Products

The following products are modelled:

Foorder:

Relations	Destinations
owns	Goal Project goal Elicitation Requirements elicitation Specification Specification Prioritisation Prioritisation Validation Requirements validation Analysis Stakeholder analysis
requires	ContextDiagram Context diagram Feature Admin UI Feature Customer App Feature Server Quality SSL communication Class User

Stakeholders

The following stakeholders have interest in the requirements:

Company D: *Foorder developers*

Attributes	Values
Rationale:	Making it easier to find restaurants that fits the customer's needs. To help both the customers and the employees of a restaurant by improving the communication between them and by improving the experience.
Problem:	People with different allergies/food preferences/diets may have difficulties finding a restaurant that fits them.

Relations	Destinations
helps	Analysis Stakeholder analysis

Restaurant customer: *Customer at a restaurant*

Attributes	Values
Rationale:	This system helps the customer in finding food.
Problem:	The customer have trouble knowing where he or she can find food according to his or her food preferences.

Relations	Destinations
helps	Analysis Stakeholder analysis

Restaurant owner: *Owns food restaurant*

Attributes	Values
Rationale:	Wants to make it simple for the customer to find them and to try their food.
Problem:	Needs new customers to find them.

Relations	Destinations
helps	Analysis Stakeholder analysis

Goals

The following goals are set:

Project goal:

Attributes	Values
	Company D wants to develop a smartphone application that you can download from Google Play (for Android phones) that will be called Foorder. This requirements document describes the requirements for the application. The app will be used by customers at restaurants as well as the employees at said restaurant. The mission of the app is to help both the customers and the employees of a restaurant by improving the communication between them and by improving the experience.
Specification:	Company D's biggest competitors will be similar systems like Onlinepizza and Mat24 therefore their main competitive advantage will be that they will focus on users being able to have a specific user account where they can save their food preferences and allergies. The project is of the type 'sub-contracting' between us and Company D, we will only be making the requirements for the application. Company D will later develop this application and our goal is to help them do this with these requirements.

Elicitation

Requirements elicitation:

Attributes	Values
Specification:	**Elicitation methods and results** Before we could start with the specification techniques and before we were able to write requirements for Foorder we made three elicitations, to help us see what product we are writing requirements for and to see which problems that could be solved with this application. The elicitation methods we choose to use was 'Group interview', 'Stakeholder analysis' and 'Prototyping'. **Group interview** This technique was used to elicit what the customers of the project had in mind when they authored their project mission. This technique was used because it was the easiest way to quickly get a grip of what the customers wanted for their product and what they needed for getting the project up and running. **Stakeholder analysis** At the time of the stakeholder analysis there were mainly four different stakeholders to this project; company D, restaurant owners of fancy restaurants, waiters and restaurant visitors. The most important part of the analysis was to ask the people with domain knowledge, restaurant owners and waiters, what goals, risks, solutions they see with Foorder. We also asked a couple of restaurant visitors a few questions to see what they wanted and thought about the idea. After receiving this information we had smaller meetings with all the stakeholders and the information we got out of this is available in Appendix 1. **Prototyping** To be able to get idéas how to get a good flow in Foorder we made a prototype in paper and then also one with a program called Fluid UI. We wanted to do this so we later on in the process could make design-level requirements. To make sure we were on the same page as our customers we showed them the prototypes and gave them the opportunity to ask questions and give us feedback. **Results from elicitation** After the first group interview with company D we reached the conclusion that the system was intended for fancy restaurants and the main stakeholder was the restaurant manager. They

wanted to make a application where the customers for the restaurants could check in to a table. They also wanted the customers to be able to order food through the app so the waiter would not have to take the order. If the waiter was needed the customer should be able to all for him/her with the app. The most important part was that the owner should have a good overview of the restaurant and, to give an example, be able to see how long customers had waited for their food and get statistics.

Then we did a stakeholder analysis with people working at restaurants and their guests. The result of this analysis was that the part of the system that the restaurant owner could get an overview of the restaurant was already in place and working without any flaws. The people working at the restaurant was not at all interested of the reduced interaction between customer and waiter and said that these features would make their business less gainful. However there were some features that they thought were really interesting and these features was those that we presented to company D. We asked if they wanted to change the scope of the mission to focus on those features.

After some compromising we, together with company D, reached the conclusion that the application should not direct to fancier restaurants but more casual and the main stakeholder is now the customer of the restaurant in contrast to the restaurant manager. The product we came up with together should focus on helping people with food preferences to find and order take out food easier.

After the prototype elicitation we come up with a good flow for the new application and we have a good start for starting to make a more realistic prototype to view the design-level requirements.

Specification

Specification:

Attributes	Values

Specification techniques

The specification techniques we have chosen to use is "Context diagram", "Feature requirements", "E/R-model", "Task descriptions" and "Screens & prototypes". And for our quality requirements we choose "Usability requirements","Performance quality" and "Security requirements". These different techniques will be explained below.

Functional requirements

Context diagram

A context diagram is a diagram over the product and its surroundings showing the scope of the project. It makes it easier to discuss it with a customer what is to be delivered and see the product surroundings. The diagram also makes it easier to verify that there is requirements written to all the interfaces that will be developed.

Feature requirements

Feature requirements is the most common way to specify requirements. These requirements can often be directly translated into functions and is often written in the text form 'The product shall ...'. It is easy to verify feature requirements but it can be quite time-consuming if there are many requirements.

E/R-model

An E/R-model is a block diagram that visualizes the data in the

system and the relationships between them. It's very useful for developers of the system but can be difficult to understand for users of the system.

Specification:

Task description

A task description is a structured text that describes a specific user task. It should be easy to understand for both developers and users. A task description is good for specifying different variations of a task.

Screens & prototypes

Screens and prototypes is excellent as design-level requirements, but only if they are well tested. Therefore it is important to test the prototype with potential users and see if they can perform all the tasks they are given without any help or assistance. The screens and prototypes technique is also very good to use if, as in our case, the entire system should be developed from scratch.

Quality requirements

You shall also explicitly specify quality requirements, including the benefit of chosen quality level from a user point of view.

Usability requirements

These requirements specify how easy the system should be to use.

Performance quality

These requirements specify fast the product shall be, e.g. the response time for the different functions in the system.

Security requirements

These requirements specify protection against abuse and unforseen disasters.

Prioritisation

Prioritisation:

Attributes	Values
Specification:	**Prioritisation method and result** Prioritisation is necessary to filter out what requirements that may not be required in the system and how to divide the requirements into different releases. The methods we used for prioritisation is described below. After the initial elicitation with company D all the different features of Foorder were written on small pieces of paper and then sorted into different groups. There were four groups created: “Must have”, “Nice to have”, “Not important” and “Discarded”. After the stakeholder analysis it was concluded that many of the earlier functions could be thrown away, so that is why the “Discarded” group was created. After some further elicitation the scope of the project changed and the prioritisation had to be redone. This time every feature was ranked for importance, both by company D and us. This was weighed together with the estimates of how long the features would take to implement and the result was the release plan.

Validation

Requirements validation:

Attributes	Values
Specification:	We choose to use the IEEE standard checklist with a few additions. This checklist is used both internally and by Company D to validate the document. Every requirement should fulfill the following points. **Correct** Incorrect requirements are useless and potentially dangerous! If the requirements are not correct, we risk spreading mis-information within project and to customers. **Complete** Specification covers all necessary reqs to describe the full scope including exceptions, error handling etc. **Unambiguous** Everyone understands it the same way. Can everyone read, discuss and agree on what it means? **Clear and Concise** Simply and clearly stated. Makes it easier for others (including pure readers) to understand. **Consistent** Are there requirements that contradict each other? **Modifiable** Modifications are easy to make, maintaining consistency of the whole specification. **Verifiable** If a requirement is not verifiable, determining whether it was correctly implemented is a matter of opinion. **Design and stability** Info needed to handle changes; why is requirement important (requirement motivation/ prioritisation / stakeholder), likely to change? **Traceable** What motivates this requirement? Indicates if it is needed. Useful when discussing scope and/or requirement changes.

Releases

The following releases are planned:

R1: *version 0.1*

Relations	Destinations
requires	Feature Add menu items into shopping cart Feature Login to the app Feature Order items in shopping cart Feature Register in the app Feature Register new restaurant Feature Search for restaurants according to location Feature See menu Feature See orders

R2: *version 1.0*

Relations	Destinations
requires	Feature Accept orders Feature Change menu Feature Create new food profile Feature Decline orders Feature Delete restaurant Feature Edit info about restaurant Feature Filter restaurants on food type Feature Notification when order is accepted Feature Notification when order is declined Feature Pay through app Feature See menu according to saved profile

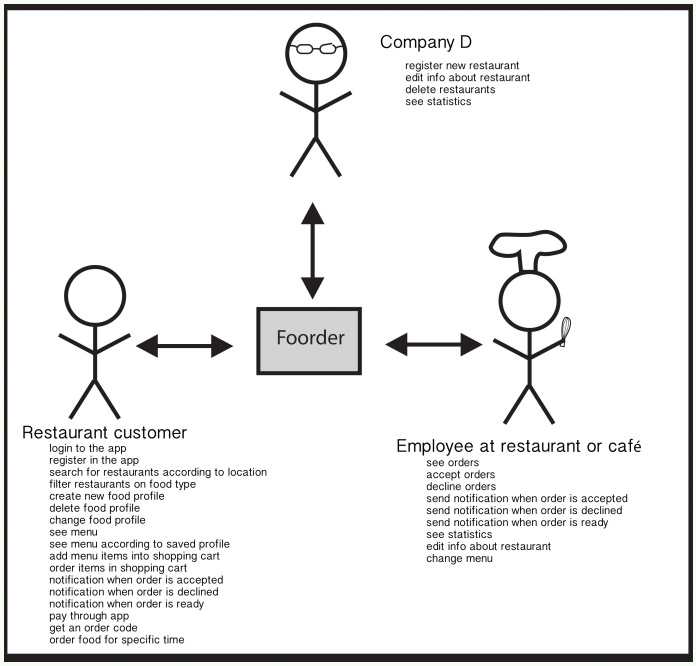
R3: *version 2.0*

Relations	Destinations
requires	Feature Change food profile Feature Delete food profile Feature Edit info about restaurant Feature Get an order code Feature Notification when order is ready Feature Order food for specific time Feature See statistics

Context Diagram

The following context diagrams have been used:

Context diagram:



Attributes	Values
Image:	context_diagram.png

Features

A feature is a releasable characteristic of a Product.

Accept orders:

Attributes	Values
Effort:	15

Relations	Destinations
owns	Function Accept order
requires	Class Order Class Restaurant

Add menu items into shopping cart:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Add dish to order Function Remove dish from order

Admin UI:

Relations	Destinations
requires	Feature Accept orders Feature Change menu Feature Decline orders Feature Edit info about restaurant Feature Notification when order is accepted Feature Notification when order is declined Feature Notification when order is ready Feature See orders Feature See statistics

Change food profile:

Attributes	Values
Effort:	30

Relations	Destinations
owns	Function Change food profile
requires	Class Profile Class User

Change menu:

Attributes	Values
Effort:	60

Relations	Destinations
owns	Function Edit restaurant menu
requires	Class Restaurant

Create new food profile:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Add food profile
requires	Class Profile Class User

Customer App:

Relations	Destinations
requires	Feature Add menu items into shopping cart Feature Change food profile Feature Create new food profile Feature Delete food profile Feature Filter restaurants on food type Feature Get an order code Feature Login to the app Feature Notification when order is accepted Feature Notification when order is declined Feature Notification when order is ready Feature Order food for specific time Feature Order items in shopping cart Feature Pay through app Feature Register in the app Feature Search for restaurants according to location Feature See menu according to saved profile Feature See menu Quality Ease of understanding Quality Time to start

Decline orders:

Attributes	Values
Effort:	15

Relations	Destinations
owns	Function Decline order
requires	Class Order Class Restaurant

Delete food profile:

Attributes	Values
Effort:	10

Relations	Destinations
owns	Function Delete food profile
requires	Class Profile Class User

Delete restaurant:

Attributes	Values
Effort:	10

Relations	Destinations
owns	Function Delete restaurant
requires	Class Restaurant

Edit info about restaurant:

Attributes	Values
Effort:	90

Relations	Destinations
owns	Function Edit restaurant info
requires	Class Restaurant

Filter restaurants on food type:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Filter search results by food type Function Filter search results by profile

Get an order code:

Attributes	Values
Effort:	15

Relations	Destinations
owns	Function Directions Function Order list Function Pickup code
requires	Class Order Class User

Login to the app:

Attributes	Values
Effort:	40

Relations	Destinations
owns	Function Guest account Function Login Quality Hashed login
requires	Class User

Notification when order is accepted:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Get notification when order is accepted Function Send notification when an order is accepted
requires	Class Order Class Restaurant

Notification when order is declined:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Get notification when order is declined Function Send notification when an order is declined
requires	Class Order Class Restaurant

Notification when order is ready:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Get notification when order is ready Function Send notification when an order is ready
requires	Class Order Class Restaurant

Order food for specific time:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Order food for specific time

Order items in shopping cart:

Attributes	Values
Effort:	80

Relations	Destinations
owns	Function One order per restaurant Function Place order

Pay through app:

Attributes	Values
Effort:	80

Relations	Destinations
owns	Function Cancel order money reservation Function Charge order money Function Reserve order money

Register in the app:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Create account Function Sync profiles

Register new restaurant:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function Register new restaurant
requires	Class Restaurant

Search for restaurants according to location:

Attributes	Values
Effort:	80

Relations	Destinations
owns	Function City search Function Location search Function Postal code search
requires	Class Restaurant search result

See menu according to saved profile:

Attributes	Values
Effort:	20

Relations	Destinations
owns	Function See menu according to a chosen profile
requires	Class Menu Class Restaurant

See menu:

Attributes	Values
Effort:	50

Relations	Destinations
owns	Function See menu according to food types Function See menu
requires	Class Menu Class Restaurant

See orders:

Attributes	Values

Effort:	30

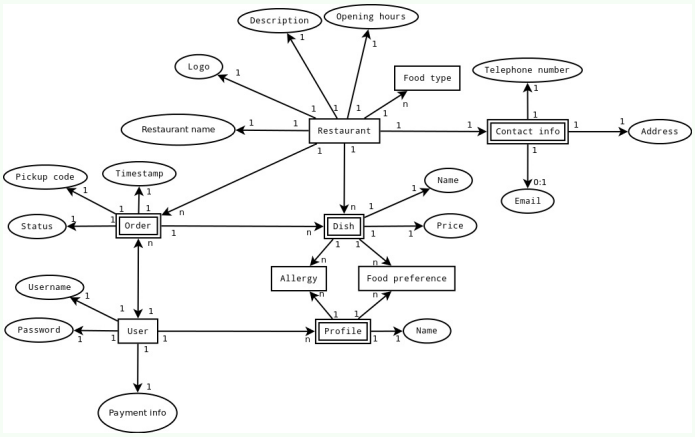
Relations	Destinations
owns	Function See order
requires	Class Order Class Restaurant

See statistics:

Attributes	Values
Effort:	45

Relations	Destinations
owns	Function Graphable Function Restricted statistics Function Sold statistics

Server:



Attributes	Values
Specification:	The back-end server that both the Admin UI and Customer App communicates with.
Image:	er_diagram.png

Relations	Destinations
requires	Feature Delete restaurant Feature Edit info about restaurant Feature Register new restaurant Feature See statistics Quality Backup database Quality Bruteforce blocking Quality Log errors Quality Order time Quality Restart Quality Simultaneous users Quality Uptime

Task

These are tasks.

1 Order food:

Attributes	Values
Rationale:	Give a customer the opportunity to make an order
Trigger:	Customer logged in to his/her account or as guest and selected an area where he/she wants to pick up the order
Critical:	Order with more than 20 items

Relations	Destinations
owns	Task 1.1 Search for restaurant Task 1.2 Browse menu Task 1.3 Select food to order Task 1.4 Pay Feature Add menu items into shopping cart Feature Filter restaurants on food type Feature Order food for specific time

requires	Feature Order items in shopping cart Feature Pay through app Feature Search for restaurants according to location Feature See menu according to saved profile Feature See menu Class Order Class Restaurant

1.1 Search for restaurant:

Attributes	Values
Specification:	variants: a) Show all restaurants, b) Show restaurants according to food type, c) Show restaurants according to food profile, d) b and c combined

1.2 Browse menu:

Attributes	Values
Specification:	variants: a) See whole menu, b) See menu according to food profile

1.3 Select food to order:

Attributes	Values
Specification:	variants: a) Add dish to order, b) Remove dish from order c) Order food for specific time.

1.4 Pay:

Attributes	Values
Specification:	Submit payment information.

2 Login to the app:

Relations	Destinations
owns	Task 2.1 Login is approved Task 2.2 Login is declined
requires	Feature Login to the app

2.1 Login is approved:

Attributes	Values
Rationale:	User wants to log in to his or hers account. 1. The user writes his or hers username. 2. The user writes his or hers password. 3. The application checks with the server and sees that the password and username is correct and matches. 4. The user is logged in to the account and the app.
Trigger:	A user tries to login
Critical:	Multiple logins at the same time/ server busy.

2.2 Login is declined:

Relations	Destinations
owns	Task 2.2.1 Login is declined, wrong password Task 2.2.2 Login is declined, no such username

2.2.1 Login is declined, wrong password:

Attributes	Values
	User wants to log in to his or hers account. 1. The user writes his or hers username. 2. The user writes his or hers password. 3.

Rationale:	The application checks with the server and sees that the password and username does not match. 4. The user is declined to login to the account and the app.
Trigger:	A user tries to login
Critical:	Multiple logins at the same time/ server busy.

2.2.2 Login is declined, no such username:

Attributes	Values
Rationale:	User wants to log in to his or hers account. 1. The user writes his or hers username. 2. The user writes his or hers password. 3. The application checks with the server and sees that the username does not exist. 4. The user is declined to login to the account and the app.
Trigger:	A user tries to login
Critical:	Multiple logins at the same time/ server busy.

3 Search for restaurants according to location:

Relations	Destinations
owns	Task 3.1 Search for restaurants according to city Task 3.2 Search for restaurants according to postal code Task 3.3 Search for restaurants according to current location
requires	Feature Search for restaurants according to location

3.1 Search for restaurants according to city:

Attributes	Values
Rationale:	1. User chooses search by city. 2. User chooses city. 3. Restaurants in that city is shown.
Trigger:	When user wants to search restaurant.
Critical:	No restaurant in the chosen city.

3.2 Search for restaurants according to postal code:

Attributes	Values
Rationale:	1. User choses search by postal code. 2. User writes the wanted postal code. 3. Restaurants in that postal code is shown.
Trigger:	When user wants to search restaurant.
Critical:	No restaurant in the chosen postal code.

3.3 Search for restaurants according to current location:

Attributes	Values
Rationale:	1. User choses search by current location. 2. Restaurants in reasonable distance to current location (given by phones location) is shown.
Trigger:	When user wants to search restaurant.
Critical:	No restaurant in the chosen current location.

4 Change food profile:

Relations	Destinations
owns	Task 4.1 Create new food profile Task 4.2 Change food profile Task 4.3 Delete food profile
requires	Feature Change food profile Feature Create new food profile Feature Delete food profile Class Profile Class User

4.1 Create new food profile:

Attributes	Values
Rationale:	1. User chooses to create food profile. 2. User gives the profile a name. 3. User marks food preferences and allergies.
Trigger:	When user wants to add a new food profile.

4.2 Change food profile:

Attributes	Values
Rationale:	1. User chooses an existing food profile. 2. User changes marked food preferences and allergies.
Trigger:	When user wants to change an existing food profile.

4.3 Delete food profile:

Attributes	Values
Rationale:	1. User chooses an existing food profile. 2. User deletes that food profile.
Trigger:	When user wants to delete an existing food profile.

5 Edit info about restaurant:

Attributes	Values
Specification:	variants: a) Change restaurant name, b) Change restaurant logo, c) Change restaurant description, d) Change restaurant opening hours, e) Change restaurant's food types f) Change restaurant's contact info g) Change restaurant's menu

Relations	Destinations
owns	Task 5.1 Change restaurant's menu Task 5.2 Change restaurant's contact info
requires	Feature Edit info about restaurant Class Restaurant

5.1 Change restaurant's menu:

Attributes	Values
Specification:	variants: a) Change prices, b) Add dishes, c) Delete dishes, d) Change a specific dish's name, e) Changes which allergies that are connected to a dish, f) Changes which food preferences that are connected to a dish

5.2 Change restaurant's contact info:

Attributes	Values
Specification:	variants: a) Change address, b) Change email, c) Change telephone number

Function

These are product level functions.

Accept order: *Accept a customer's order*

Attributes	Values
Specification:	A restaurant employee must be able to accept a customer's order.
Rationale:	The restaurant must be able to accept an order.

Relations	Destinations
requires	Class Order

Add dish to order: *Adding dishes*

Attributes	Values
Specification:	Food dishes can be added to the order.
Rationale:	For the order to mean anything, it must be possible to add food to it.

Relations	Destinations
requires	Class Dish

Add food profile: *Create a new profile*

Attributes	Values
Specification:	After invoking the new profile function the user must add a name to the profile and save it.
Rationale:	The user should be able to create new food profiles.

Relations	Destinations
requires	Class Profile

Cancel order money reservation: *No food = no money*

Attributes	Values
Specification:	Cancel the money reservation when the order is declined.
Rationale:	If a order is declined the customer won't get any food so no money should be charged.

Relations	Destinations
requires	Function Decline order Function Reserve order money

Change food profile: *Change existing food profile*

Attributes	Values
Specification:	The user can edit their profile, picking food preferences and allergies from a predetermined list.
Rationale:	The user should be able to change food profiles according to their preferences.

Charge order money: *Earning money*

Attributes	Values
Specification:	After an order is accepted the money reserved for that order is charged.
Rationale:	The restaurant needs money to operate.

Relations	Destinations
requires	Function Accept order Function Reserve order money

City search: *Restaurants in specific city*

Attributes	Values
Specification:	The user must be able to choose from a predetermined list of cities.

Rationale:	If the user wants to find a restaurant in a specific city.

Create account: *Create account for new users*

Attributes	**Values**
Specification:	The user must be able to create a account in the app.
Rationale:	In order to login you must first create an account.

Decline order: *Decline a customer's order*

Attributes	**Values**
Specification:	A restaurant employee must be able to decline a customer's order.
Rationale:	If the restaurant employee sees a reason not to accept an order (if for example an ingredient is recently out of stock) he or she must be able to decline an order.

Delete food profile: *Deleting unused or unwanted food profiles*

Attributes	**Values**
Specification:	It must be possible to remove a profile inside the application.
Rationale:	The user should be able to delete profiles that are not used anymore.

Delete restaurant: *Delete a restaurant*

Attributes	**Values**
Specification:	Company D must be able to delete a restaurant from the server.
Rationale:	If a restaurant do not want to be a part of Foorder any more Company D should have a way of deleting that restaurant from the server.

Deletion confirmation: *Food profile deletion confirmation*

Attributes	**Values**
Specification:	The user must confirm a deletion of a food profile before the profile is deleted.
Rationale:	To prevent unwanted deletion of a food profile.

Relations	**Destinations**
requires	Function Delete food profile

Directions: *Directions to a restaurant*

Attributes	**Values**
Specification:	A step-by-step direction to a restaurant from the users location should be availabe through the app.
Rationale:	The user might not know where the restaurant is or how to get there.

Edit restaurant info: *Edit a restaurant's information*

Attributes	**Values**
Specification:	A restaurant employee or Company D must be able to edit the restaurant's information.
Rationale:	If a restaurant changes something about their restaurant or changes address for example, the restaurant or Company D

	then wants to be able to change the information.

Edit restaurant menu: *Edit a restaurant's menu*

Attributes	Values
Specification:	A restaurant employee or Company D must be able to edit the restaurant's menu.
Rationale:	If a restaurant changes something in their menu the restaurant then want to be able to change their menu visible in the user application.

Filter search results by food type: *Food you like*

Attributes	Values
Specification:	The user can select from a collection of available pre-defined food categories and filter the restaurants according to the filters. Only restaurants matching the selected categories will be shown.
Rationale:	Sometimes the user want just a specific type of food.

Relations	Destinations
requires	Class Food type

Filter search results by profile: *Relevant restaurants*

Attributes	Values
Specification:	The user can select from a collection of available user-defined profiles and filter the restaurants according to the profile. Only restaurants with at least one dish suitable to the profile will be shown.
Rationale:	Restaurants without any dishes according to the user's profile are irrelephant.

Relations	Destinations
requires	Class Profile

Get notification when order is accepted: *Get a notification when an order has been accepted*

Attributes	Values
Specification:	A user must be able get a notification when an order is accepted.
Rationale:	A customer should have a way of knowing if his or her order can be made and if it was accepted. Because otherwise they will have to make a new order.

Relations	Destinations
requires	Function Send notification when an order is accepted

Get notification when order is declined: *Get a notification when an order has been declined*

Attributes	Values
Specification:	A user must be able get a notification when an order is declined.
Rationale:	A customer should have a way of knowing if his or her order can be made and if it was not accepted. Because then they will have to make a new order.

Relations	Destinations
requires	Function Send notification when an order is declined

Get notification when order is ready: *Get a notification when an order is ready*

Attributes	Values
Specification:	A user must be able get a notification when an order is ready.
Rationale:	A customer should not have to wait at the restaurant for an order to be ready.

Relations	Destinations
requires	Function Send notification when order is ready

Graphable: *Visualise the statistics*

Attributes	Values
Specification:	Statistics over sold dishes and placed orders should be exportable to a graphical format.
Rationale:	By providing visual feedback such as graphs for the restaurant and its employees it will be faster and easier for them to understand the meaning of the numbers.

Guest account: *For one-time users*

Attributes	Values
Specification:	The app can be accessed without having an account. All features are available except for user profiles and placing orders
Rationale:	Not everyone wants or needs an account.

Location search: *Nearest restaurants*

Attributes	Values
Specification:	The user must be able to find restaurants close by. That are then listed and sorted by distance in ascending order.
Rationale:	Using a users phone location a list of restaurants should be presented.

Login: *Users accounts*

Attributes	Values
Specification:	The user can login to Foorder using a username and password.
Rationale:	TO be able to save information about the user, such as food profiles or payment information.

One order per restaurant: *Food from only one place*

Attributes	Values
Specification:	When a user leaves a restaurant's menu, the cart is emptied.
Rationale:	It shouldn't be possible to order food from multiple places at once. Keeping the orders seperate will prevent this.

Order food for specific time: *Order food for later*

Attributes	Values
Specification:	When placing a order there should be an option to order for specific time. A time later the same day may be entered, provided it is before the restaurant closes
Rationale:	To avoid having to wait for food, the user might want to order in advance. Then the restaurant will (might) have the food ready

	exactly when the user wants

Relations	Destinations
requires	Function Place order

Order list: *View previous orders*

Attributes	Values
Specification:	A user must be able to get a list of all his or hers placed orders.
Rationale:	So the user can keep track of previous orders.

Pickup code: *Anti-theft system*

Attributes	Values
Specification:	After a successful order payment the user receives a code which is used for picking up the food. The restaurant receives the same code.
Rationale:	To prevent someone from picking up a food order they didn't pay for.

Relations	Destinations
requires	Function Get notification when order is accepted

Place order: *Submit the order*

Attributes	Values
Specification:	If money is successfully reserved from the user the order will be transferred to the restaurant.
Rationale:	So that the user can order the food they want and so that the restaurant can see what food to make.

Relations	Destinations
requires	Function Reserve order money

Postal code search: *Search a specific postal code*

Attributes	Values
Specification:	The user can enter a postal code and search for restaurants located in the postal code.
Rationale:	For big cities a city search covers a too large area and the search needs to be more specific.

Register new restaurant: *Register a new restaurant*

Attributes	Values
Specification:	Company D must be able to add a new restaurant to the server.
Rationale:	Company D should have a way of adding a new restaurant to the server if that restaurant wants to be a part of Foorder.

Remove dish from order: *Removing dishes*

Attributes	Values
Specification:	Food dishes can be removed from the order.
Rationale:	The user might change his or her mind after picking a dish.

Reserve order money: *Making sure payment will come*

Attributes	Values
Specification:	Reserve a sum equal to the cost of the order when the order is placed.
Rationale:	Since orders don't have to be accepted by the restaurant, money can't be charged until the order is accepted. But the restaurant still want reassurance that they will get money for their food. This way they know that they will get their money and the customer knows he/she will only be charged if the order is accepted.

Restricted statistics: *Protecting sales data*

Attributes	Values
Specification:	A restaurant can only see their own sales. Company D can see all restaurants sales.
Rationale:	Restaurants don't want to share their sales information to everyone.

See menu according to a chosen profile: *See a restaurant's menu according to a chosen profile*

Attributes	Values
Specification:	The user must be able to filter the menu according to his or hers food profile of choice.
Rationale:	The user doesn't want to eat food that isn't in his or hers food profile.

Relations	Destinations
requires	Class Profile

See menu according to food types: *See a restaurant's menu according to a chosen profile*

Attributes	Values
Specification:	A user must be able to choose if certain food types is shown or not in the menu.
Rationale:	A user may want to be able to see just a certain food type in the menu.
Deprecated:	You should only be able to filter restaurants on food types. Not dishes.

See menu: *See a restaurant's menu*

Attributes	Values
Specification:	A user must be able to get the menus from listed restaurants.
Rationale:	To be able to choose which food to order, the user must be able to see what food that restaurant has to offer.

See order: *See a customer's order*

Attributes	Values
Specification:	A restaurant employee must be able to see the customers orders.
Rationale:	If a restaurant employee is going to make and to accept orders he or she has to be able to see the orders.

Send notification when an order is accepted: *Send a notification when an order has been accepted*

Attributes	Values
Specification:	A restaurant employee must be able to notify a customer if an order is accepted.
Rationale:	A restaurant employee should be able to inform the customer if the order will be made.

Relations	Destinations
requires	Class Restaurant

Send notification when an order is declined: *Send a notification when an order has been declined*

Attributes	Values
Specification:	A restaurant employee must be able to notify a customer if an order is declined.
Rationale:	A restaurant employee should be able to inform the customer if the order can not be made for any reason.

Relations	Destinations
requires	Class Restaurant

Send notification when an order is ready: *Send a notification when an order is ready*

Attributes	Values
Specification:	A restaurant employee must be able to notify a customer if an order is ready.
Rationale:	A customer should not have to wait at the restaurant for an order to be ready the restaurant should therefore be able to inform the customer when the order is ready.

Send notification when order is ready:

Relations	Destinations
requires	Class Restaurant

Sold statistics: *See statistics over sold food*

Attributes	Values
Specification:	The restaurant must be able to see statistics over their sold food.
Rationale:	If the restaurant can see how much they are selling of a certain dish, on a certain day or at what hour it can help them to easier predict how much they will need to prepare in the future.

Sync profiles: *Profile downloading*

Attributes	Values
Specification:	Profiles must be stored on the server and the customer app must make sure that the profiles the app has stored are the same that the server has stored.
Rationale:	If you log on to another device or delete and then restore the app you want your profile information there.

Relations	Destinations
requires	Class Profile

Quality

These are quality requirements.

Backup database: *How often to backup the database*

Attributes	Values
Specification:	The database should be backed up at least once every 6 hour.
Rationale:	If something goes wrong the system should be restorable from a recent backup.

Bruteforce blocking: *Attempts to stop bruteforcing of an account*

Attributes	Values
Specification:	If the server recieves 5 failed logins to a user account in less than 5 minutes it should limit the user account to a maximum of 1 login per minute until either a successful login or 5 minutes have passed since last login attempt. Likewise if the server recieves 5 failed logins from an IP adress in less than 5 minutes it should limit the IP adress to a maximum of 1 login per minute until either a successful login or 5 minutes have passed since last login attempt.
Rationale:	For safety concerns.

Ease of understanding: *How easy it is for new users to register and order food*

Attributes	Values
Specification:	90% of first time users should be able to start the app, register and order food from a restaurant in less than 10 minutes.
Rationale:	It should be very simple for a new user to understand the app.

Hashed login: *Secure user accounts*

Attributes	Values
Specification:	User passwords must be stored as a hashed value in the server database.
Rationale:	Payment information tied to the account makes it attractive to hackers.

Log errors: *To track errors*

Attributes	Values
Specification:	Orders, logins and menu updates that fail should be logged to a local file on the server.
Rationale:	If something goes wrong it is good to be able to track it.

Network error handling: *When network traffic is used*

Attributes	Values
Specification:	Network traffic calls must have a timeout and notify the user if the timeout has been reached.
Rationale:	The user needs to be informed if the action wasn't successful.

Order time: *How long it takes to put an order*

Attributes	Values
Specification:	The time from order arrival on the server until it is registered in the system must take less than 1 second in 95% of all cases.
Rationale:	The system should appear fast to the end user.

Restart: *Be able to self-restart*

Attributes	Values
Specification:	When an irrecoverable error occurs the server software shall shutdown and restart within 5 minutes.
Rationale:	If something goes wrong the server should not require human interaction.

SSL communication: *Encrypted communication*

Attributes	Values
Specification:	Communication between the application and server must be encrypted via SSL.
Rationale:	To prevent sensitive information from leaking out.

Relations	Destinations
requires	Feature Server

Simultaneous users: *How many can use the system at the same time*

Attributes	Values
Specification:	The system must be able to handle at least 2000 logins or orders per minute.
Rationale:	The system must be able to handle multiple users at the same time.

Time to start: *Time until login screen is visible*

Attributes	Values
Specification:	In 9/10 cases the app should take less than 2 seconds from startup until it is showing the login screen.
Rationale:	The user should not have to wait for too long.

Uptime: *How long the server should be responsive on average*

Attributes	Values
Specification:	the server should have an uptime of at least 99.8% per month.
Rationale:	The restaurants depend on the server to be able to recieve orders.

Class

African:

Attributes	Values
Specification:	Tag for african food.

Relations	Destinations
inherits	Class Food type

Allergy:

Attributes	Values
Specification:	Abstract type for different allergies.

Atkins:

Attributes	Values
Specification:	Tag for Atkins food preference.

Relations	Destinations
inherits	Class Food preference

BBQ:

Attributes	Values
Specification:	Tag for BBQ food.

Relations	Destinations
inherits	Class Food type

Beef:

Attributes	Values
Specification:	Tag for beef allergy.

Relations	Destinations
inherits	Class Allergy

Chicken:

Attributes	Values
Specification:	Tag for chicken allergy.

Relations	Destinations
inherits	Class Allergy

Chinese:

Attributes	Values
Specification:	Tag for chinese food.

Relations	Destinations
inherits	Class Food type

Contact info:

Relations	Destinations
requires	Member Restaurant address Member Restaurant email Member Restaurant telephone number

Corn:

Attributes	Values
Specification:	Tag for corn allergy.

Relations	Destinations
inherits	Class Allergy

Dish:

Relations	Destinations
requires	Class Allergy Class Food preference Member Food name Member Price

Eggs:

Attributes	Values
Specification:	Tag for egg allergy.

Relations	Destinations
inherits	Class Allergy

Fast food:

Attributes	Values
Specification:	Tag for fast food.

Relations	Destinations
inherits	Class Food type

Fish:

Attributes	Values
Specification:	Tag for fish allergy.

Relations	Destinations
inherits	Class Allergy

Food preference:

Attributes	Values
Specification:	Abstract type for different food preferences.

Food type:

Attributes	Values
Specification:	Abstract type for food types.

Fruitarian:

Attributes	Values
Specification:	Tag for fruitarian food preference.

Relations	Destinations
inherits	Class Food preference

Garlic:

Attributes	Values
Specification:	Tag for garlic allergy.

Relations	Destinations
inherits	Class Allergy

Gluten:

Attributes	Values
Specification:	Tag for gluten allergy.

Relations	Destinations
inherits	Class Allergy

Greek:

Attributes	Values
Specification:	Tag for greek food.

Relations	Destinations
inherits	Class Food type

Indian:

Attributes	Values
Specification:	Tag for indian food.

Relations	Destinations
inherits	Class Food type

Italian:

Attributes	Values
Specification:	Tag for italian food.

Relations	Destinations
inherits	Class Food type

Kosher:

Attributes	Values
Specification:	Tag for kosher food preference.

Relations	Destinations
inherits	Class Food preference

LCHF:

Attributes	Values
Specification:	Tag for LCHF food preference.

Relations	Destinations
inherits	Class Food preference

Lactose:

Attributes	Values
Specification:	Tag for lactose allergy.

Relations	Destinations
inherits	Class Allergy

Mayonaise:

Attributes	Values
Specification:	Tag for mayonaise allergy.

Relations	Destinations
inherits	Class Allergy

Menu:

Relations	Destinations
requires	Class Dish

Mexican:

Attributes	Values
Specification:	Tag for mexican food.

Relations	Destinations
inherits	Class Food type

Onion:

Attributes	Values
Specification:	Tag for onion allergy.

Relations	Destinations
inherits	Class Allergy

Oranges:

Attributes	Values
Specification:	Tag for orange allergy.

Relations	Destinations
inherits	Class Allergy

Order:

Relations	Destinations
requires	Class Dish Class Restaurant Member Pickup code Member Status Member Timestamp

Other:

Attributes	Values
Specification:	Tag for other food.

Relations	Destinations
inherits	Class Food type

Peanuts:

Attributes	Values
Specification:	Tag for peanut allergy.

Relations	Destinations
inherits	Class Allergy

Pizza:

Attributes	Values
Specification:	Tag for pizza food.

Relations	Destinations
inherits	Class Food type

Pork:

Attributes	Values
Specification:	Tag for pork allergy.

Relations	Destinations
inherits	Class Allergy

Profile:

Attributes	Values
Specification:	A food profile with allergies and food preferences.

Relations	Destinations
requires	Class Allergy Class Food preference

Restaurant search result:

Attributes	Values
Specification:	The information displayed for each restaurant in the search result list.

Relations	Destinations
owns	Member Restaurant distance Member Restaurant logo Member Restaurant name Member Restaurant opening hours

Restaurant:

Relations	Destinations
requires	Class Contact info Class Menu Member Restaurant description Member Restaurant distance Member Restaurant logo Member Restaurant name Member Restaurant opening hours

Sesame seeds:

Attributes	Values
Specification:	Tag for sesame seed allergy.

Relations	Destinations
inherits	Class Allergy

Shellfish:

Attributes	Values

Specification:	Tag for shellfish allergy.

Relations	Destinations
inherits	Class Allergy

Soybeans:

Attributes	Values
Specification:	Tag for soybeans allergy.

Relations	Destinations
inherits	Class Allergy

Spinach:

Attributes	Values
Specification:	Tag for spinach allergy.

Relations	Destinations
inherits	Class Allergy

Strawberries:

Attributes	Values
Specification:	Tag for strawberry allergy.

Relations	Destinations
inherits	Class Allergy

Sunflower seeds:

Attributes	Values
Specification:	Tag for sunflower seed allergy.

Relations	Destinations
inherits	Class Allergy

Sushi:

Attributes	Values
Specification:	Tag for sushi food.

Relations	Destinations
inherits	Class Food type

Thai:

Attributes	Values
Specification:	Tag for thai food.

Relations	Destinations
inherits	Class Food type

Tomatoes:

Attributes	Values
Specification:	Tag for tomatoe allergy.

Relations	Destinations
inherits	Class Allergy

Tree nuts:

Attributes	Values
Specification:	Tag for tree nut allergy.

Relations	Destinations
inherits	Class Allergy

User: *Restaurant customer*

Relations	Destinations
owns	Class Profile Member Password Member Username

Vegan:

Attributes	Values
Specification:	Tag for vegan food preference.

Relations	Destinations
inherits	Class Food preference

Vegetarian:

Attributes	Values
Specification:	Tag for vegetarian food preference.

Relations	Destinations
inherits	Class Food preference

Vietnamese:

Attributes	Values
Specification:	Tag for vietnamese food.

Relations	Destinations
inherits	Class Food type

Wheat:

Attributes	Values
Specification:	Tag for wheat allergy.

Relations	Destinations
inherits	Class Allergy

Yeast:

Attributes	Values
Specification:	Tag for yeast allergy.

Relations	Destinations
inherits	Class Allergy

Member

Food name:

Attributes	Values
Specification:	The name of the food/dish, max 40 characters.

Password:

Attributes	Values
Specification:	Text, min 5 characters max 20 characters. Can only contain alphanumerics.

Pickup code:

Attributes	Values
Specification:	Text, 5 characters. Can only contain alphanumerics

Price:

Attributes	Values
Specification:	The price of the food/dish, max 10 characters.

Restaurant address:

Attributes	Values
Specification:	The restaurant's address, max 500 characters

Restaurant description:

Attributes	Values
Specification:	The description about the restaurant, max 700 characters.

Restaurant distance:

Attributes	Values
Specification:	The user's distance to the restaurant in kilometers with one decimal point precision, max 10 characters.

Restaurant email:

Attributes	Values
Specification:	The restaurant's email address, max 40 characters.

Restaurant logo:

Attributes	Values
Specification:	The restaurant logo as an image, pixels 480x320 png.

Restaurant name:

Attributes	Values
Specification:	The restaurant name, max 40 characters.

Restaurant opening hours:

Attributes	Values
Specification:	The restaurant's opening hours for today or 'closed' if not opened, as text and with max 30 characters.

Restaurant telephone number:

Attributes	Values
Specification:	The restaurant's telephone number, max 20 characters.

Status:

Attributes	Values
Specification:	The status of the order. Can be 'Cooking', 'Waiting for pickup' or 'Done', max 20 characters.

Timestamp:

Attributes	Values
Specification:	A timestamp for the order, when the order was made, max 20 characters.

Username:

Attributes	Values
Specification:	Text, min 3 characters max 20 characters. Can only contain alphanumerics.

Analysis

These are analysis that have been done

Stakeholder analysis:

Attributes	Values

Company D

Company D wants the system to be attractive enough to restaurant owners so that they can sell it to the system to them and make money. To ensure that the project is a success and that the system is both what they and their customers want they need to share their input with the people writing the requirements.

Company D pays for the whole development and specification of the system for a great cost. The risk is that restaurant owners don't want to buy the system and the income of the project is smaller than the cost of the development. Restaurants already have systems similar to Foorder but company D hopes that Foorder will be able to act as an extension to these older system rather than replace them.

Specification:

Restaurant owner

The restaurant owner who got interviewed emphasized from the start that if there should be an app it should be a common one for all the restaurants and not a custom made for each. He says it would be good if a customer could search for different categories of restaurants in the app. In the application, he wants to have similar information available on the website such as the menu, how to get there and photos of the restaurant.

The restaurant owner says he is absolutely not interested in an application where customers can place an order which is then sent to the restaurant system. He says he would lose on this because then the restaurant would not be able to recommend the food to the customers and he believes that they would more often choose the cheapest dish on the menu.

When it comes to the booking of tables he was at first not at all interested in this. But after he had talked a while he changed his mind and could see the benefits with a booking system IF it lets the restaurant approve it or write a comment in response. He believes that if a large party books a table at the restaurant at 19.00 and then another large party books at 19:15 he wants to be able to tell them that they may come at 20.00. So the app would run a kind of reservation system that can easily and seamlessly first send how many are in your party and suggest what time they will come and then the restaurant will be able to propose a new time, refuse or accept this. The customer should not be able to chose table as it may not be optimal for the restaurant and also the layout can change.

As for the booking he would like to see that there was a way that allows the customer to understand the consequences that they are not so many in the party as they said when they booked or not even show up.

In terms of an overview picture of how long customers have been waiting for, what has been ordered and the like, there is already a cash system which solves this. He sees no reason to replace this system with an app because it solved all the needs that already exists. In this system, you can also see statistics about what is being sold. He is not interested in ratings of the restaurant or the food in an application because there are very many pages that have this already and it's not all that sets the rating therefore there witll be no meaningful statistics in the end.

He says that he would like to be part of an application that allows customers to order takeaway food from them. Before arriving at the restaurant the customer could order their food and the time that the customer intends to come and get it (he has heard of online pizza but made no use of it). He wishes for some kind of protection if the customer does not show up.

He also mentions an application that automatically updates his inventory system would be good. However, that is more for the suppliers and the app will completely change focus from the "project mission".

He could imagine that it might be good to be able to put offers in the application if he recognises it as a revenue stream. He points out that the app must be well marketed and there must be many (many!) restaurants interested in the start. He has previously been involved in an application where restaurants could put offers customers could receive. There were very few restaurants that participated in the beginning, and he immediately saw that it would not turn into something and therefore never put up any offers which led to those who developed the app never receiving any money. He would definitely spend money on an application that is well developed, features many restaurants and puts a lot of focus on marketing.

Restaurant staff

After an interview with two waitresses we found out that they themselves couldn't think of any system/app that would make their work easier.

They mention that they already have a POS system that keeps track of how long the visitors have been waiting and what they have ordered. They did not think that an application where the customers could order through the app would help. Because that would take away a large part of their work and the possibility for customer contact. They also believed that everything was going to be more confused if the customer had ordered what food they wanted before they got to the restaurant.

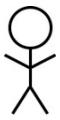
They state that they really don't want the customer contact and their ability to provide services to be lost with this application.

They feel like they might think it is a good idea to be able to write down orders through the app and then have the order sent to the kitchen. But only if really works. They fear that if would only makes things more complicated if it's not done properly.

After some given suggestions and a bit of discussion, they came to the conclusion that a feature that customers could use to order refills on drinks or snacks or just for "calling out" to a waiters would be a great feature. They tell us that when the customer received their food and after their waitress have checked with them that everything tastes good, they then leave the customers alone to enjoy the food. They say that it is often difficult to obtain when they are again needed at the table.

They also said that it would be really great if the customers could give tip with the application after the visit.

After a discussion we also concluded that it would be good if the customer with the app could be able to get answers to which dishes contain lactose or gluten (or other diet preferences). But what they didn't want was the app to have recommendations on drinks to fit certain food and likewise, because that would take away a big part of their job.



Restaurant visitors

The potential customers and restaurant visitors we spoke to said they would love to have the opportunity to pre-order food and they also said that they wanted to be able to specify the time of the visit if that was possible. They also thought that it would be a good thing to be able to see the ingredients of the different meals in the menu and to sort foods according to different categories such as "chicken", "LCHF" or "lactose-free".

They also liked to be able to reserve a table and be able to see a map of the restaurant in the app. They preferred that the app shouldn't be designed for a particular restaurant, that it instead should be possible to see a variety of restaurants.

The customers thought that it would have been nice to be able to get recommendations on which drinks is best suitable for a meal and would like to be able to see reviews of the various dishes, and also be able to grade the meal after the visit. And to be able to give comments on the experience.They would in this case also like to be able to see comments and ratings that other customers have set.

They thought that the customer itself should be able to decide whether they want to have an user account on the app or not. If they had an account they thought it would be nice to be able to save their allergies and such so that the menus could be adapted accordingly.

They thought that the feature to be able to "call out" to a waiter/waitress through the app would be useful but only as a complement to the "normal" way.

The customers also thought that it would be fun to be able to

save their receipts so that they could remember what they ate. At last they said that a feature to split the order would be very useful.