Technical Requirements

In this document we analyse the requirements for each person that interacts with our system.

Based on these requirements we will decide what technologies to use for developing the application

Continue

Click on an actor to get more detailed information

Customer

- Access application via any mobile device
- Access application easily if not installed already
- View the menu of a corresponding event
- Place orders via the application or via a waiter
- Add comments to a placed order
- Order extra items as part of the same bill
- Pay the bill via the application or via a waiter
- Possibility of splitting the bill

Employee

- Access the system via a mobile device
- Access the system from a web browser on a PC
- View the menu of an event
- Order items for customers
- Accept payment from clients
- See an overview of the currently placed orders
- Modify the status of an existing order

Manager

- Access system via any web browser
- Access system securely
- Create/modify events
- Add/modify menus of these events
- Assign administrators to events
- Add payment information to system
- Be protected from misuse of the system
- Have a system that is reliable and stable

Developer

- Easy to learn and use frameworks
- Simplify re-using existing code
- Easily change structure of database
- Quick prototyping and compilation
- Use technologies with good documentation
- Effortless Multiplatform support
- System that supports many simultaneous users

Technology Choices

Developer Requirements

- *Easy to learn and use frameworks*

  -> Because we have a limited amount of time to work on this project, we want to use frameworks which are easy to learn and use.

- *Simplify re-using existing code*

  -> The technologies we utilize should make it easy to re-use existing code that we already wrote before. This allows for better modularization of the code and saves time.

- *Easily change structure of database*

  -> During the project we might want to change certain properties of database objects. The database system should be flexible enough to support adding or removing certain fields without having to start over.

- *Quick prototyping and compilation*

  -> A lightweight framework with just the features we need is preferable. This keeps the duration of the modification-deployment cycle short. This makes debugging and rebuilding the project a faster experience.

Back to Overview Next Page

Developer Requirements

- *Use technologies with good documentation*
  -> We do not have any prior experience with java frameworks. It would work in our advantage to use frameworks that are already commonly used. The documentation and help we can find while working with such a framework will be much larger than with a relatively uncommon framework.

- *Effortless Multiplatform support*
  -> Making a seperate application for each platform is very time consuming and requires knowledge of specific features of each platform. It would be easier to use a framework that can generate platform specific apps based on a general application design.

- *System that supports many simultaneous users*
  -> Having a framework that can scale well for a large amount of users is a plus. This allows us to support many simultaneous users without having to write special code for this.

Previous Page Back to Overview Next Page

Developer equivalent technology properties

Based on the requirements of the customer we can define certain properties that should guide our decision of technology and frameworks

- Good documentation
- Ease of use
- Flexible database structure
- Quick prototyping and compilation
- Effortless multiplatform

Previous Page Back to Overview

Customer Requirements

- *Access application via any mobile device*
  - -> To allow users to access the application from any device we should use a framework which supports **multi-Platform** development.
  - -> This frameworks should be focused on the creation of **mobile interfaces,** since the primary platform for customers will be a smartphone or tablet.

- *Access application easily if not installed already*
  - -> We cannot assume that every customer at an event will have the application pre-installed. This means that we need to provide an alternative way to access the application. This could be done via a **web application.** The technology we use should allow for the creation of both **native and web-based interfaces.**
  - -> The system should also have a way for customers to easily find this native application or web app using existing software on their phone (E.g. Scan a QR-code and be redirected to google-play or the web app)

- *View the menu of a corresponding event*
  - -> The customers interface needs to send information to and receive information from the central application This should happen quickly and for multiple users of the application simultaneously. This requires a **scalable system** with an **efficient interface.**

- Place orders via the application or via a waiter
  - -> To have even more flexibility, we will allow users to place their own orders from within the application.

Back to Overview

Next Page

Customer Requirements

- *Add comments to a placed order*
  -> When placing orders we will need some additional fields in the database to allow for comments on placed orders (E.g. no tomatoes). There might be some other fields that might be added later on, so it's useful to have the possibility to **change the structure of the database** without having to start from the beginning.

- *Order extra items as part of the same bill*
  -> Each order should be linked to one bill, so that we can make a total pricing of the various items ordered from the same group of customers.

- *Pay the bill via the application or via a waiter*
  -> The framework we use should be able to communication with the **API of a payment system** (E.g. Paypal API) and should have built-in **security features** or the possibility to add this ourselves.

- *Possibility of splitting the bill.*
  -> Customers might want to split a bill evenly between the persons in a group.

Customer equivalent technology properties

Based on the requirements of the customer we can define certain properties that should guide our decision of technology and frameworks

- Multi-Platform
- Mobile interface
- Native and web interface
- Efficient interface
- Flexible ordering system
- Can communicate with payment APIs
- Built-in Security

Previous Page Back to Overview

Employee Requirements

- *Access the system via a mobile device*
  - -> The employees need a **reliable and performant** mobile interface which they can use to place orders and accept payment from customers. For this situation a **native application** is the best choice. If there is a **connection issue** the employees can still continue to work and submit the orders when a connection is available again.
  - -> Supporting **multiple platforms** is nice, but not a necessity, as they will likely use a device from the company.

- *Access the system from a web browser on a PC*
  - -> In a restaurant it might be desirable to have a stationary computer which connects to the central application via a **web interface**. A tablet computer is also an alternative, this could then use a similar interface as the interface for mobile devices.

- *View the menu of an event & order items for customers*
  - -> Like a customer, an employee also has the capability to display the menu of an event, They can place orders for customers who don’t have or don’t want to use their own device.

Back to Overview

Next Page

Employee Requirements

- *Accept payment from customers*
  - -> Employees can accept payment from customers. This should be done using a **secure** system that can connect to existing **payment APIs.**
- *See an overview of the currently placed orders*
  - -> Every employee can see an overview of placed orders. When a order is finished, employees should be **notified immediately** so they can bring the order to the respective customer. To make this possible the interface of the employee should **refresh automatically** to show the updated order overview at any moment in time.
- *Modify the status of an existing order*
  - -> If a customer makes a mistake, the employee can change the order by modifying it with his interface. This could be done by removing the previous order and creating a new one. Or by actually modifying the existing order. We need a certain **flexibility** from the database for this.

Previous Page Back to Overview Next Page

Employee equivalent technology properties

Based on the requirements of the employee we can define certain properties that should guide our decision of technology and frameworks

- Reliable
- Performant interface
- Native interface
- Persistent – continues to work with connection issues
- Multi-platform
- Flexible database structure
- Can communicate with payment APIs
- Built-in Security
- Automatic Refresh of data

Previous Page Back to Overview

Manager Requirements

- *Access system via any web browser*
  - -> The manager will typically access the application from their computer. It makes sense to make the application **cross-browser compatible**, this will make sure that it works on common browsers like chrome, firefox, safari, opera and edge.
- *Access system securely*
  - -> As our system contains sensitive information, it is logical to try and make it as **secure** as possible.
- *Create/modify events & add/modify menus of these events*
  - -> Each manager will be able to create and maintain multiple events and create menus for each such event. It's important that the employees can continue working with the system while the manager is making modifications.(**zero downtime deployment**) To serve a large amount of potential managers the central application should be **scalable**.
- *Assign administrators to events*
  - -> Managers want to assign designated people as administrator for a specific event or restaurant. Our system should allow multiple tiers of access in a **secure** way

Back to Overview

Next Page

Manager Requirements

- Add payment information to system
  - -> To accept payment from customers, the manager will link the central application to the desired payment system that he wants to utilize. This requires a **secure** system that can **interface with existing payment APIs**

- Be protected from misuse of the system
  - -> Not only security is important but also basic **tampering protection.** It should be impossible for customers to make orders that make no sense, e.g. ordering immense amounts of items or ordering for another table.

- Have a system that is reliable and stable
  - -> The application will need to be tested for for uptime and reliability. We should be able to **stress test** the system with sample data and loads.

Previous Page Back to Overview Next Page

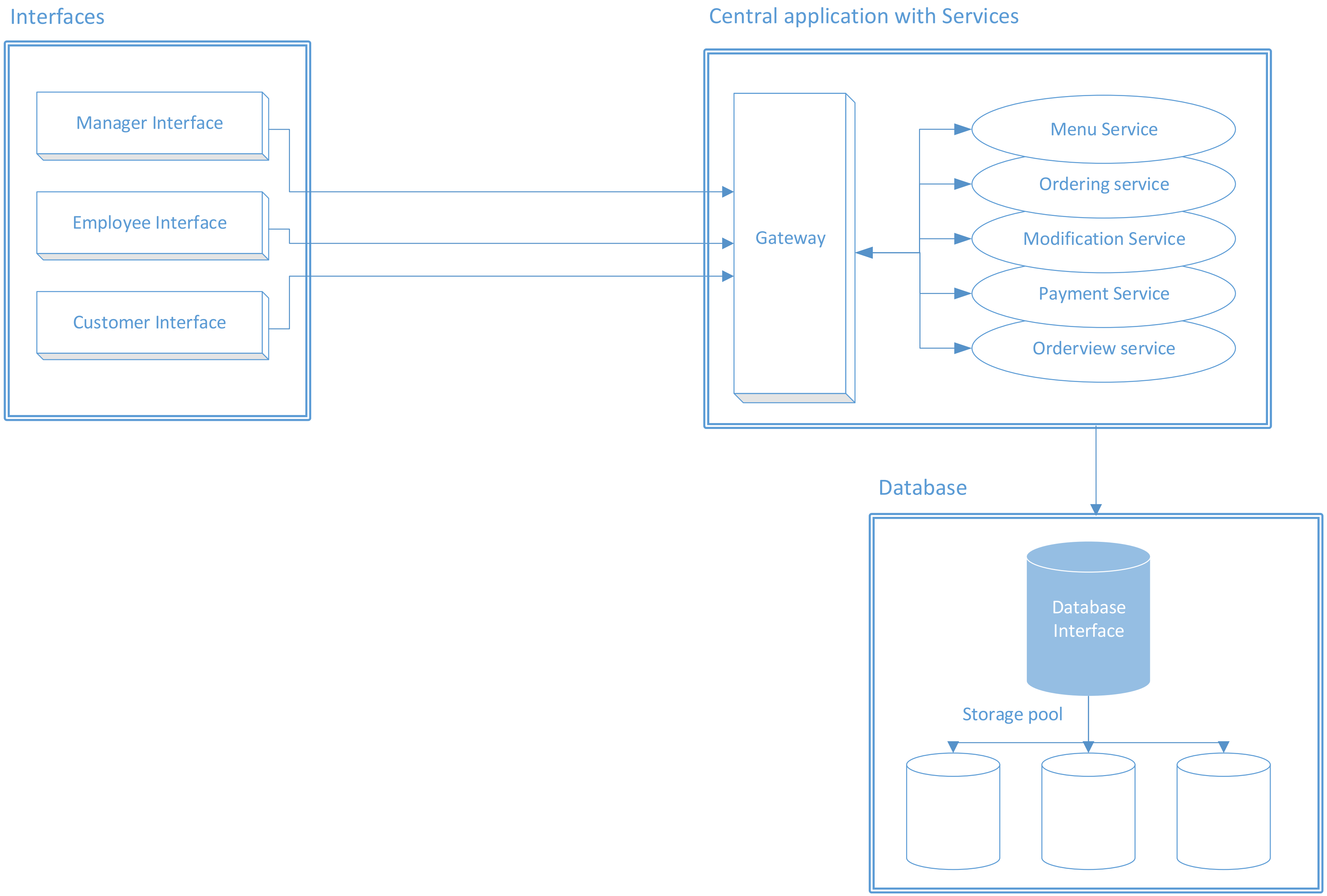
Manager equivalent technology properties

Based on the requirements of the manager we can define certain properties that should guide our decision of technology and frameworks

- Cross-Browser compatible
- Scalable system for multiple events/restaurants
- Reliable
- Update prices/information without downtime
- Can communicate with payment APIs
- Built-in Security
- Built-in tampering protection

Previous Page Back to Overview

Faros Online Cash Register – Technology chart



Back to requirements

Click on any part of the diagram to get more detailed information

☑ Good documentation

☑ Quick prototyping and compilation

☑ Built-in Security

☑ Ease of use

Because we will create the system with Java technologies, we will limit ourselves to frameworks based on Java. Commonly used Java web frameworks are **Spring**, **JavaServer Faces** and **Vaadin**.

We do not have any experience with any of these frameworks. This means that there is a big possibility that the choice we make is not the optimal framework for our project. We believe that **good documentation** and **accessibility** will be the deciding factors in determining the best framework for us to use.

Currently, Spring MVC is the most used Java web framework, with almost 40% of the developers using this system. Various resources online also use Spring as their example framework in online courses and tutorials. Additionally, Spring has a large amount of documentation and guides available for kickstarting development and supports the basic building blocks that we need for our application.

Based on these properties, we will start development on this project using the Spring MVC framework.

Back to Overview

APACHE CORDOVA™

PhoneGap

ionic

Xamarin

Multi-platform

The mobile interface should be accessible on as many of the existing platforms as possible.

- Android
- IOS
- Windows Phone

Accessible without installation

A web interface for mobile devices allows users to use the application without having it installed on their device

Built-in security and privacy features

The application should have basic security and privacy build-in. Customers using the application won’t use it if its not safe.

Payment from within the application

Customers should be able to pay employees for their items from within the application itself.

Back to Overview

Conclusion

APACHE CORDOVA™

PhoneGap

ionic

Xamarin

☑ Multi-platform

Apache Cordova supports multiple mobile platforms in addition to applications for ubuntu, windows and firefox

- ☑ Android
- ☑ IOS
- ☑ Windows Phone

☑ Accessible without installation

Apache cordova works from a html/css application, this code can be re-used to build an interface accesible via a web browser.

☑ Built-in security and privacy features

Cordova supports whitelisting and encrypted storage to improve the security of an application.

☑ Payment from within the application

Cordova contains an extensive 3rd party plugin system. This includes plugins to connect to popular payment systems like paypal.

Back to Overview

Conclusion

APACHE CORDOVA™

PhoneGap

ionic

Xamarin

☑ Multi-platform

Ionic supports the most popular mobile platforms. The latest release candidate of the ionic framework is capable of deploying applications to the universal windows platform

- ☑ Android
- ☑ IOS
- ☐ Windows Phone

☐ Accessible without installation

Browser support for ionic is limited. It is possible to deploy ionic code as a web app but the system is focused for native/hybrid development

☑ Built-in security and privacy features

Ionic has build-in secure storage and Oauth, which allows for basic security

☐ Payment from within the application

Ionic does not have built-in or existing 3rd party integration with common payment systems

Back to Overview

Conclusion

APACHE CORDOVA™

PhoneGap

ionic

Xamarin

✓ Multi-platform

Xamarin includes development tools for most popular mobile platforms

- ✓ Android
- ✓ IOS
- ✓ Windows Phone

☐ Accessible without installation

Xamarin does not have tools for creating a web application accessible through a browser.

✓ Built-in security and privacy features

Xamarin applies encryption and SSL/TLS connections as its primary way of implementing security.

✓ Payment from within the application

Xamarin has plugins called components that allow payment with popular mobile systems. There are also 3rd party plugins that support systems like paypal.

Back to Overview

Conclusion

All three of these mobile application development frameworks have an impressive featureset that will greatly assist us in speeding up the creation of a mobile version of our web application.

Both cordova and Xamarin support the three major mobile platforms, with Ionic supporting Ios and android out of the box.

We will choose to work with **Apache Cordova** because of its extensive documentation and various library options for features that are useful for improving the user experience of the application.

Cordova will allow us to quickly prototype a mobile version of an existing web interface that works on several platforms.

Back to Overview

To operate, our web application needs a database. From this database we can retrieve and store information for the customers, employees and the manager.

We don’t know in advance how many managers will make use of this application. Each manager can also create and maintain a variable number of restaurants/events. This means that our database technology should be **Scalable**.

If we want a system where orders can be changed on the fly and that allows employees and the manager to update prices and information without affecting the usage of the system. We need a database that is **flexible.**

Each time a user accesses the web application, their interface will dynamically request several distinct resources from the database. This process should be **Performant**.

Because of its flexibility and room for scaling **MongoDB** will be our preferred choice of database. However, depending on how we design the application a solution that works with an SQL database would also be possible.

For connecting this database with the framework we will use **spring data**, this slice of the spring framework allows the manipulation of data with NoSQL databases like MongoDB.

Back to Overview