Rony Azrak, Brent Coutts, Max Johnson, Jamie Day

Requirements Document

Functional System Requirements

The system shall:

#1 - High Priority: Keep track of staff (i.e., their schedules and roles).

#3 - High Priority: Include functionality to manage staff (add, modify, remove).

#5 - High Priority: Keep track of available equipment (grills, fryers, etc.)

#6 - High Priority: Keep track of available ingredients (meat, lettuce, cheese, etc.)

#7 - High Priority: Keep track of menu items and their popularity.

#8 - High Priority: Handle customer orders if there are enough ingredients for the selected menu items and automatically update remaining ingredients in the case of a successful order and increase the recorded popularity of items ordered.

Non-functional System Requirements

The system shall:

#9 - High Priority: Support laptop/desktop machines, mobile devices, and web browsers.

#10 - Medium Priority: Support persistence, with changes from previous sessions being stored locally.

#11 - Medium Priority: Perform operations swiftly in a manner which minimizes user frustration.

#12 - High Priority: Ensure that the system is trustworthy in such a way that no data can be lost in the case of software malfunction.

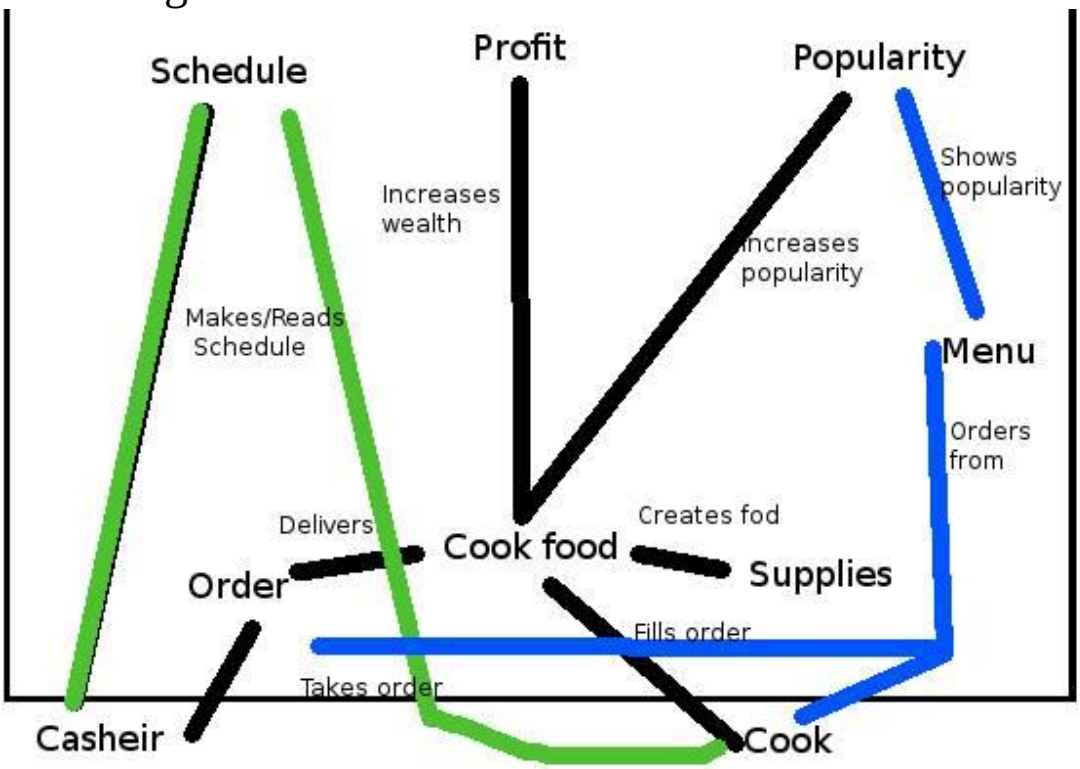
Domain Model

```
namespace ca.mcgill.ecse321.FTMS.model;

class FTMSManager {
 singleton;
 1 -> * Equipment;
 * -> 1 Menu menu;
 1 -> * Staff staff;
 * -> * Supply;
}
class Staff {
 String role;
 * -> 1 Schedule;
}
class Schedule{
 Date sunday;
}
class Equipment {
 String name;
 Integer quantity;
}

class Menu {
 1 -> * MenuItem;
}
class MenuItem {
 String name;
 Integer popularity;
 Double price;
 * -> * Supply ingredients;
}
class Supply {
 String name;
 Integer quantity;
}
class Order {
 autounique id;
 1 -> * MenuItem;
}
```

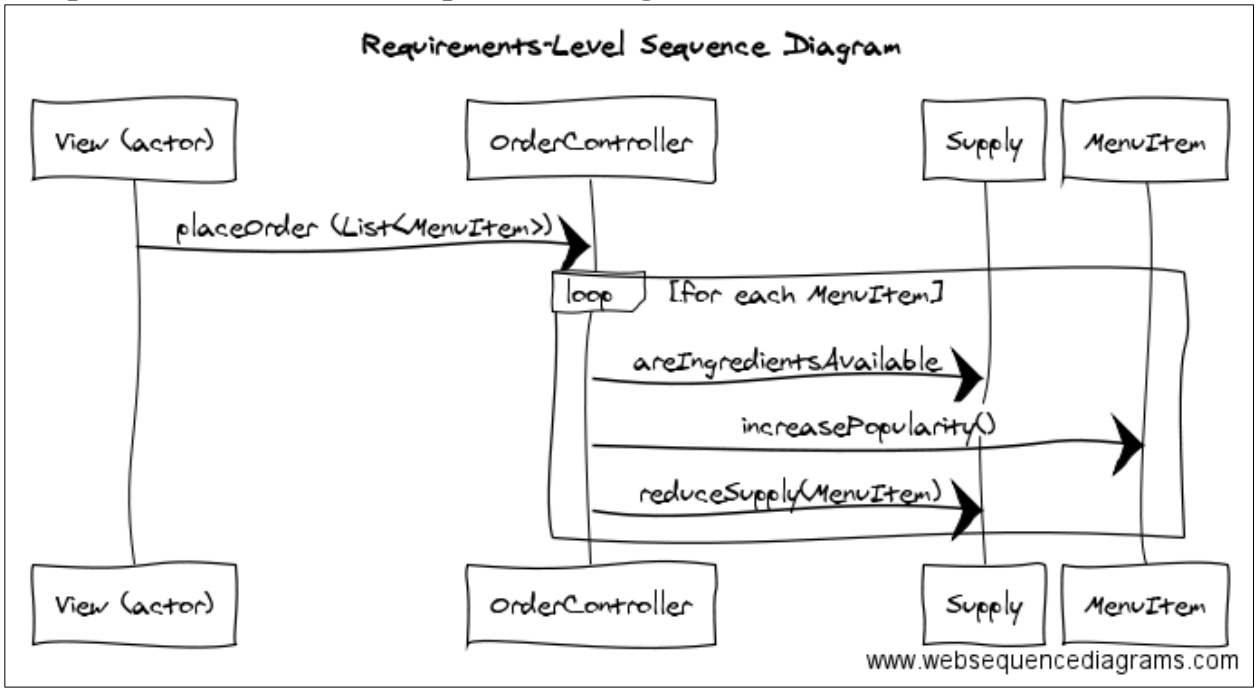
Use case diagram



Current use cases for our minimum viable product:

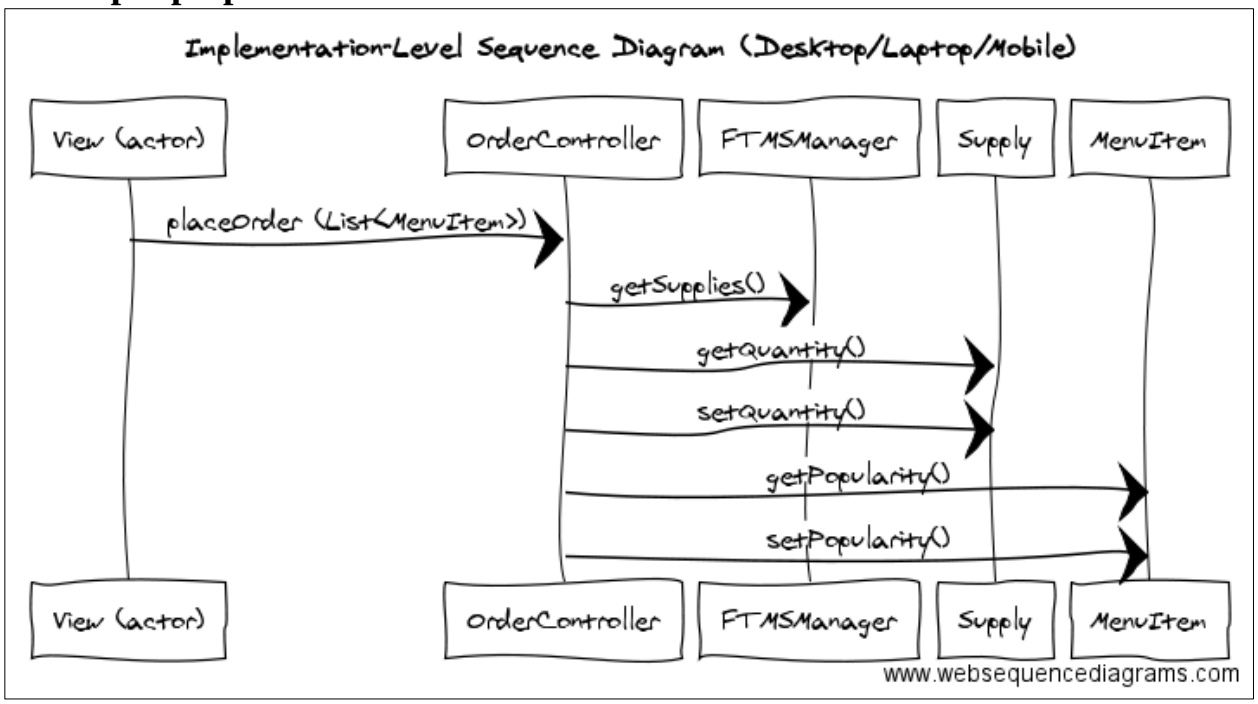
- *Order (selected use case)*
- Add to menu
- Modify Equipment
- Modify staff schedules/roles

Requirements-level sequence diagram

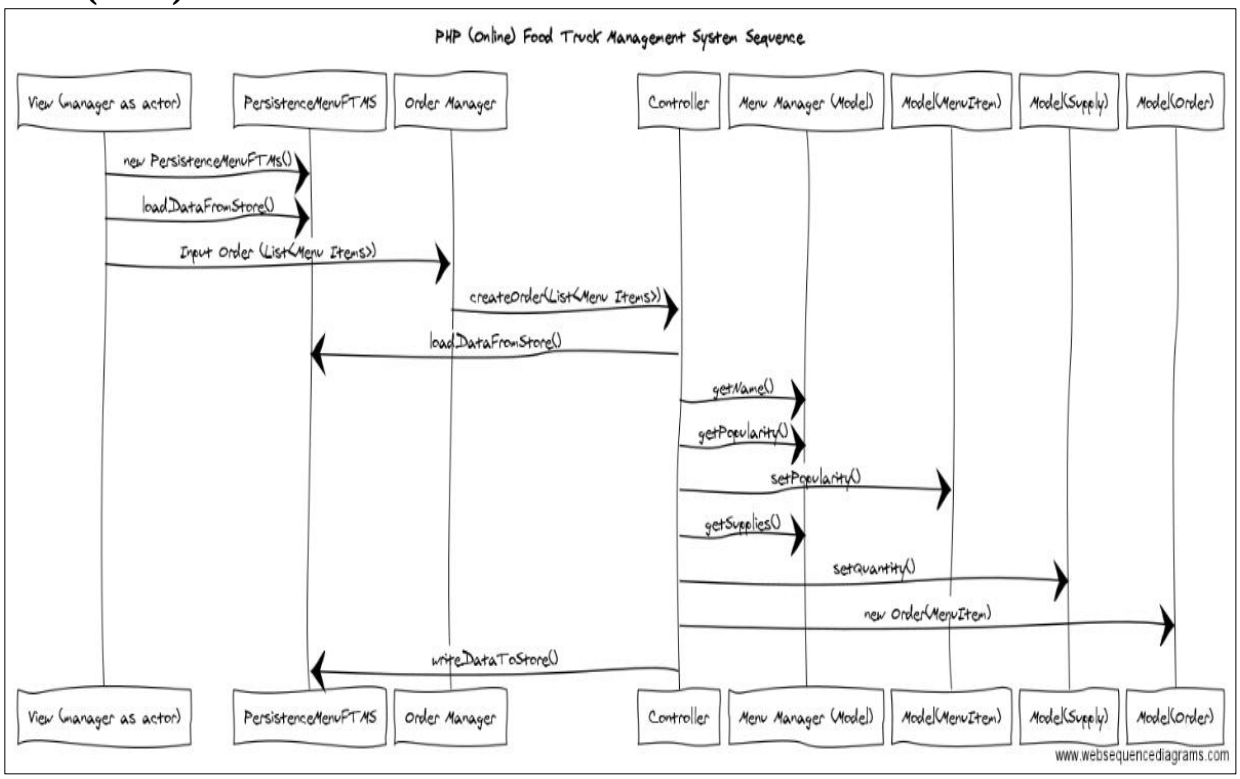


Implementation-level sequence diagrams

Desktop/laptop & Mobile



Web (PHP)


PHP (Online) Food Truck Management System Sequence
View (manager as actor)
PersistenceMenuFTMS
Order Manager
Controller
Menu Manager (Model)
Model(MenuItem)
Model(Supply)
Model(Order)
new PersistenceMenuFTMS()
loadDataFromStore()
Input Order (List<Menu Items>)
createOrder(List<Menu Items>)
loadDataFromStore()
getName()
getPopularity()
setPopularity()
getSupplies()
setQuantity()
new Order(MenuItem)
writeDataToStore()
View (manager as actor)
PersistenceMenuFTMS
Order Manager
Controller
Menu Manager (Model)
Model(MenuItem)
Model(Supply)
Model(Order)
www.websequencediagrams.com

Work Plan

Deliverable 2:

The block diagram shows the overall architecture of our software. It shows the relationships and dependencies between the main systems. We only show the systems on a global level, without writing details about those so it will take around 2 hours to create.

The class diagram is a more detailed design of how the different classes interact with each other and the methods they use. We need the domain model and use cases to help create the class diagram. Having these in hand, this should only take about two hours.

These should be done by October 31th.

Deliverable 3:

Once we know what functions we want to have in place and what they should perform, we will create unit tests for all the simple functions we create, which should not take more than an hour. Some functions will use many simple functions inside of them to perform something bigger and more important in relation to our program. We will create component tests for those, which should also take around an hour. Finally, we will create tests on a global scale of the software to see if our main functionalities work and if we reach our goals regarding the program. The systems tests should be faster to create than the component tests, and even faster than unit tests because we are working on the most global scale and we have fewer things to test. These tests stack up to each other. If the unit tests are not successful, then the component and system tests will not be either.

Performance/stress testing will be done after all tests pass. We will try to run our program with large amounts of objects and see how well it performs. This test should take the least amount of time, so around 30 minutes.

These should be done by November 14th.

Deliverable 4:

The release pipeline will require collaboration between all of our team members as it will integrate all the different tools of our software and release it to the user. It should show all the different stages of handing the program from the team to the user. It shows which stages trigger others automatically or manually. For example, once all the tests and builds are successful, the test stage will manually trigger the release stage. An expected time of 3-5 hours will be dedicated for this.

The pipeline plan should be done by November 21st.

Deliverable 5:

Once the pipeline plan is complete and we have all other documents and systems created for our software, we will be ready to start preparing our presentation. We need to have a clear understanding of our program in order to present it to others and we need to encapsulate every part of it. This is expected to take the longest amount time so we will dedicate 7-10 hours for it.

The presentation material should be done by November 27th so we have time to get prepared for the actual presentation.

Deliverable 6:

After the presentation is complete, we will take as many critics into account to improve the last bits of our program before submitting the full implementation. We will use an estimate of 2-3 hours for this part.

The final source code should be done by December 15th.