Architecture

Group 17

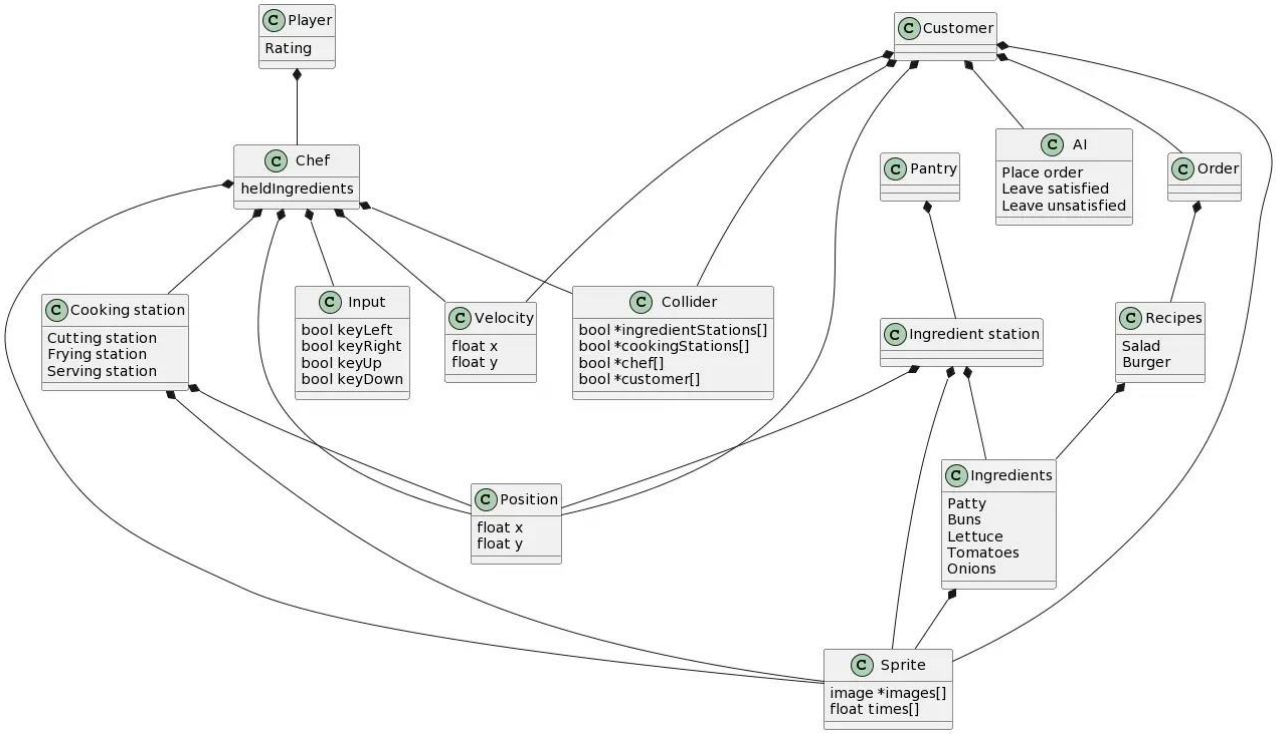
Team Loading

Joel Warren
Charlie Wilson
Jake Keast
Hari Bhandari
Ishrit Thakur
Joshua Hills

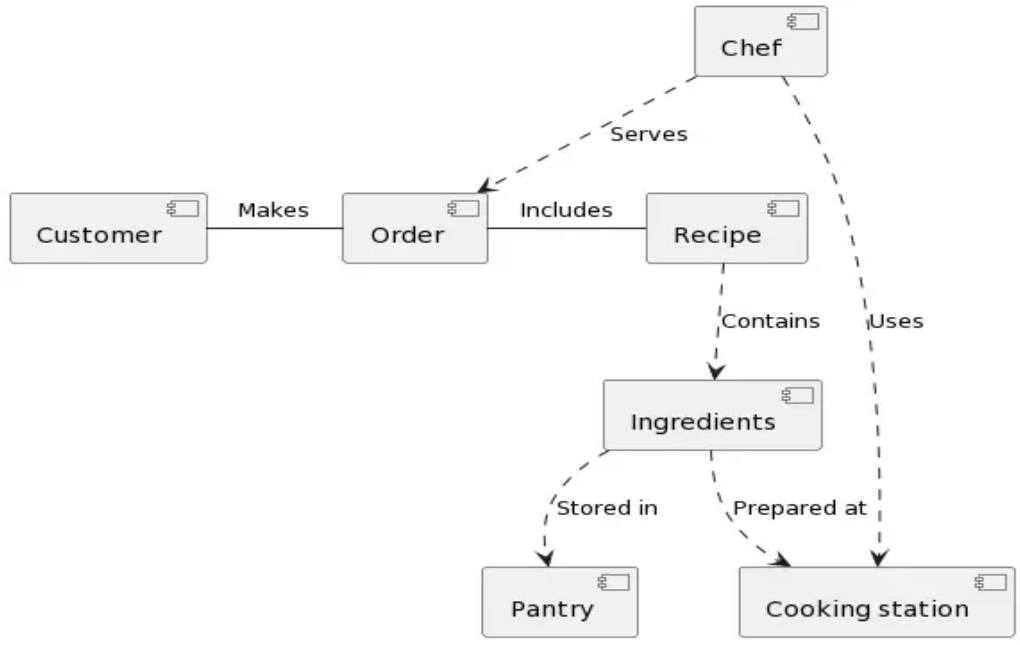
Architecture 3. a)

Structural diagrams

Class Diagram

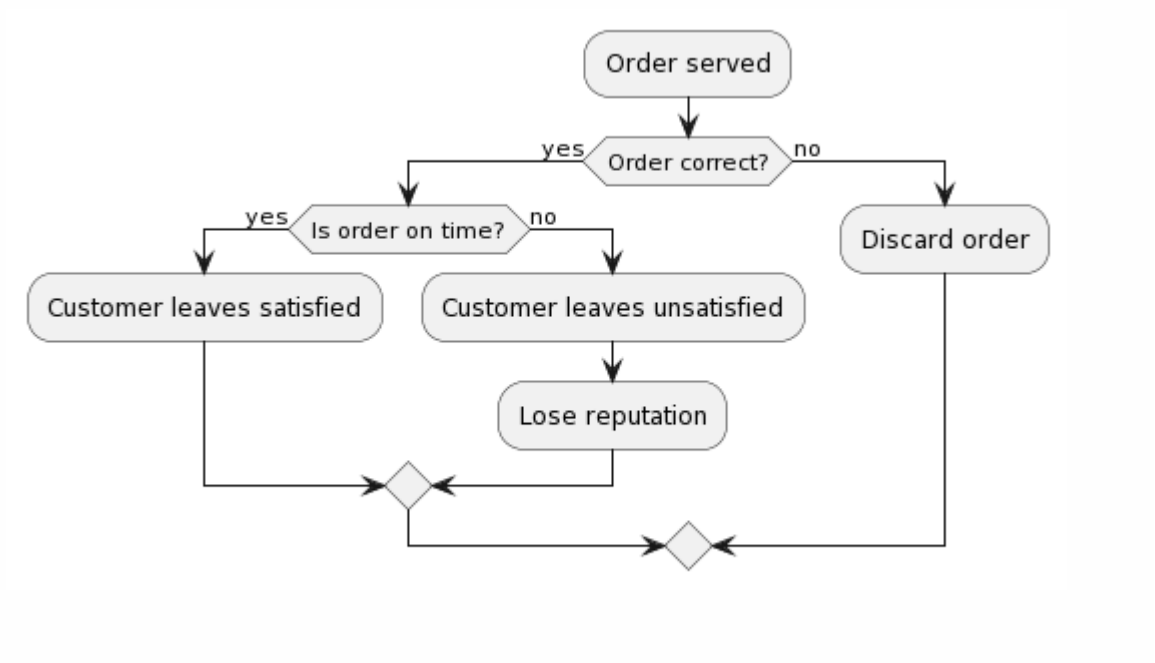


Component Diagram

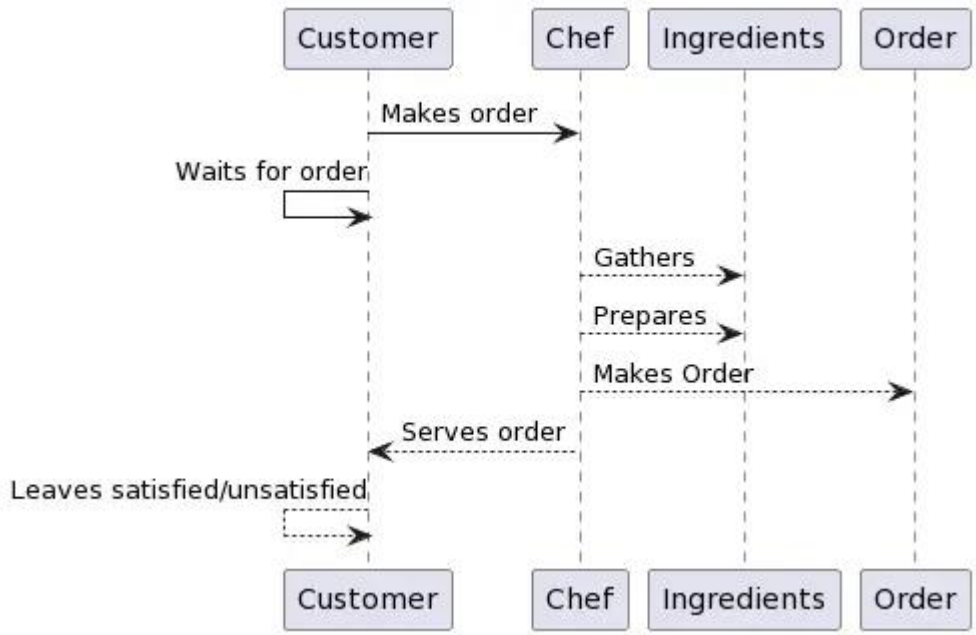


Behavioural diagrams

Activity Diagram

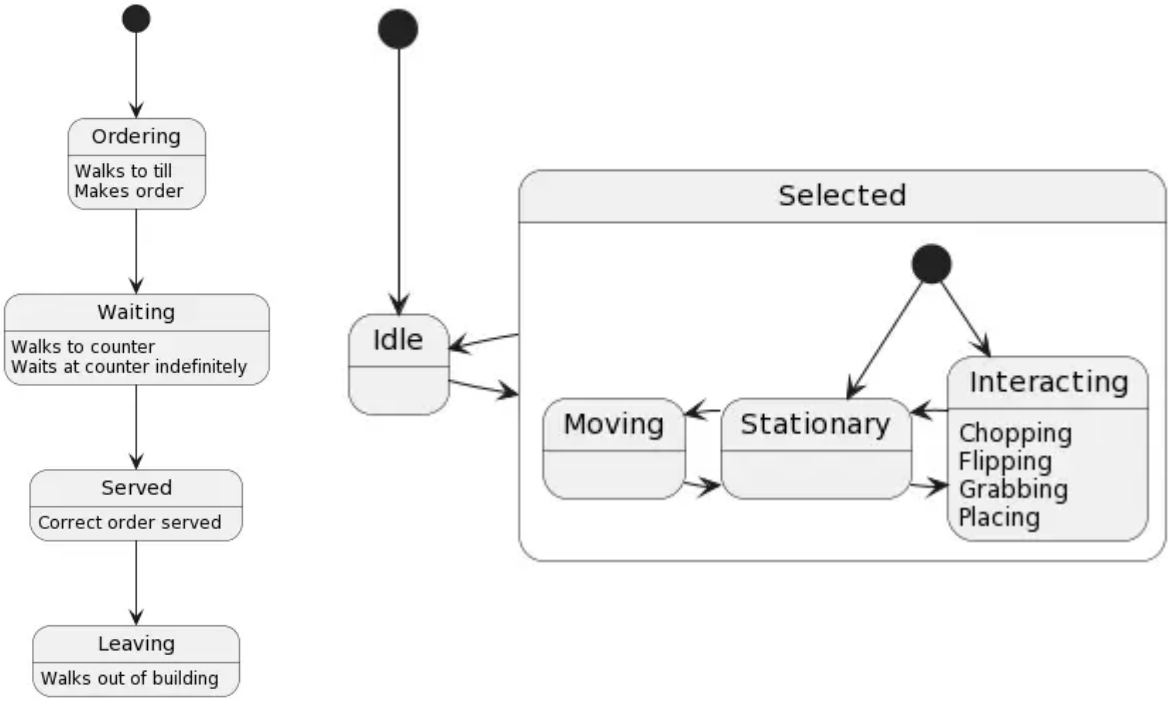


Sequence Diagram



State diagrams

(Customer) (Chef)



Languages / tools used

UML, a general purpose modelling language, was used to create these diagrams, a very flexible and broad tool, with the following:

Structural diagrams used:

- Class
  - Used to capture core entities, their features and relationships
- Component
  - Used for high-level decomposition

Behavioural diagrams used:

- Activity - Used to show object communication
- Sequence - Used to show object communication
- State - Used to capture various system states

The tool used was PlantUML, which supported quickly constructing these diagrams. PlantUML also had high automation, such as the layout, reducing the risk of human error and making it an ideal tool.

Architecture 3. b)

Justification for architecture

Justification for tool (PlantUML):

In order to create clear and robust architecture diagrams, PlantUML was used. PlantUML is an open-source program able to create a variety of different diagrams, including Class, Component, Activity, and Sequence diagrams. It was also used to create the Gantt Charts to aid in communication and planning for the project.

Justification for diagrams used:

Class Diagram:

- We created a Class Diagram in order to provide a basic structure for the implementation of our code. This diagram would give an ideal starting point and class hierarchy needed to base our code on.

Component Diagram:

- We created a Component Diagram to provide a more logical description of what should be happening within the game. This higher level design will enable a clear understanding of the processes requiring implementation.

Activity Diagram:

- The Activity Diagram shows the overall loop that the program needs to follow. This diagram shows what needs to happen and why, whereas the Component Diagram only shows what needs to happen, with no context from the actual scenario in-game.

Sequence Diagram:

- The sequence diagram depicts the sequence of paths that will be followed when the program runs. The diagram shows what steps will be executed after each step.

State Diagrams:

- The State Diagrams show the various “phases” that each of the actors on screen could be in. Each of these phases will provide a different behaviour for the actor and inputs and processes should differ based upon these phases. This allows those working on implementation to ensure that nothing is unaccounted for and that each phase behaves correctly.

Justification for design process:

We derived the architecture from the requirements because this was a recommended process in both lectures provided and external literature, as described below. This also allowed us to create accurate diagrams which sufficiently reflected the user requirements and thus correctly representing the game’s components. This did however create delays in waiting for the requirements statement to be finished beforehand. The rest of the process, i.e. workflow approach to identifying initial components, was also carried out after studying the related lectures on designing architecture.

Initial design and evolution

The design process we followed to create the software architecture involved:

- Identifying initial components
- Assigning Requirements to Components
- Analysing Roles and Responsibilities
- Analysing Architecture Characteristics
- Restructuring Components

All architectural diagrams including interim versions can be found at:

https://eng1-loading.github.io/Loading/architecture

To identify the initial components, a workflow approach was used. We identified activities inferred from the product brief, such as making an order, then built components around these. This allowed us to create the initial component diagram, shown on the website.

Once we had a complete statement of user and system requirements, we could then assign requirements to the components. We could now directly analyse the user and system requirements, which highlighted unrepresented requirements in our current component diagram. This led to a restructure of components, shown by “Component Diagram” on the website.

Based on the initial activities identified, we then created further and more detailed architectural diagrams to further reflect the various requirements. This involved a similar workflow approach, such as identifying the key activities such as delivering and order, then building e.g. an activity diagram from this.

Once we started implementation, this also gave us a clearer understanding of the key structures and behaviours, allowing the architecture to further develop and evolve. Implementing the game gave us more knowledge in key functionalities of how components behaved. From this, we could further restructure our components and diagrams accordingly.

Relation to requirements

Architecture	**Related Requirements**
Class diagram	● UR_COOK_CONTROL ● UR_ORDER_RECEIVE ● UR_ORDER_COMPLETE ● FR_PANTRY ● FR_COOKING_STATIONS
Component diagram	● UR_ORDER_RECEIVE ● UR_ORDER_COMPLETE ● FR_PANTRY ● FR_COOKING_STATIONS
Activity diagram	● UR_ORDER_COMPLETE ● FR_ORDER_SERVED ● FR_ORDER_INCOMPLETE
Sequence diagram	● UR_COOK_CONTROL ● UR_ORDER_RECEIVE ● UR_ORDER_COMPLETE ● FR_ORDER_SERVED ● FR_PANTRY ● FR_COOKING_STATIONS
State diagram (customer)	● FR_CUSTOMER_ARRIVE ● FR_CUSTOMER_WAIT ● FR_ORDER_SERVED
State diagram (chef)	● UR_COOK_SWITCH ● UR_COOK_CONTROL ● UR_ORDER_RECEIVE ● UR_ORDER_COMPLETE ● FR_PANTRY ● FR_COOKING_STATIONS

** Architecture references may need to be updated for subsequent deliverables to reflect changes and/or additions to the architecture.