האקדמית כנרת

הנדסה, חברה ורוח

Course: Distributed Systems - Semester 1 of 5785
Assignment 1

Directions

A. Due Date: 1 December 2024 at 11:55pm

B. Groups of three (3) or four (4) students may submit this assignment.

C. Code for this assignment (Ass1) must be submitted via Github using the per-assignment private repository opened for you in the organization. More details on the repository are found below.

D. There are 100 points total on this assignment.

E. What to turn in:

   (a) All source code and library files necessary to compile and run your programs

   (b) Queue server design document

   (c) Database design document

   (d) Testing document

   (e) Gradle build scripts

   (f) README.md file with:

      i. The name of the course, semester, and year

      ii. The names and IDs of all students

      iii. The task performed by each student

      iv. The number of hours worked by each student on the tasks

General Requirements

1. All of the code below must be written in Java and compilable and executable with Java 19.
2. All code must have comments - each function must have an introductory comment detailing its purpose, input parameters, and return value.

Parking System Stage 1: Communication

This semester we will build a system in three stages. The system will implement a number of use cases related to a single story: a cellular parking system. The system will force you to deal with numerous aspects of distributed systems including communication, data storage, communication, and synchronization.

1 User Story: Mulligan

We will build a system called Mulligan. Mulligan will have a front end and back end that are a simplified version of Pango. The system will have a set of customers, parking spaces, parking zones, vehicles that will be loaded up from an initialization file. A customer can begin parking in a parking space by sending a Start message to the server. The customer can stop parking in a parking space by sending a Stop message to the server. Each parking space is associated with a parking area that dictates how much the space costs per hour. If a customer begins parking in a parking space, but hasn't closed a previous parking event, the system Stops the previous event automatically.

When the customer send a Stop message to the server, it generates a payment transaction that is sent to the queuing service.

Parking Enforcement Officers (PEOs) can query the Mulligan system to see if a vehicle is parked legally in a particular parking space. The query includes the vehicle number and the parking space number. Mulligan will return a response Ok or Not Ok based on whether the vehicle is currently legally parked in the space. If the vehicle is not legally parked in the space, the officer can write a parking ticket (citation/fine).

When the PEO issues is a citation, it generates a citation event that is sent to the queuing service. The citation includes the vehicle number, the parking space number, the parking zone, and the time of inspection.

Municipalities can also use Mulligan to track payment transactions and citations. The municipality can print out a list of payment transactions and issued citations.

A customer can request the list of past parking events that he performed from Mulligan, including the total amount of money the customer has paid. The customer can then use that information to challenge a parking ticket (citation/fine) that he received erroneously at the Municipality.

2 List of Actors

The following are the actors in the system:

Name	Goal
Parking Customer	To pay for parking a vehicle legally
Parking Enforcement Officer (PEO)	To check whether vehicles are currently legally parked in a space
Municipal Officer (MO)	To check whether a vehicle was legally parked in a space

3 Use Cases

The user story is short and covers only limited parts of what a real parking system will need to perform. This semester we will implement the following use cases in the system. The use cases below do not cover all of the Mulligan system's needs. You'll need to add functionality as required to support the use cases.

System Use Case 1:	Starting a Parking Event
Actors and interests:	Parking Customer: To start a new parking event
Stakeholders:	
Pre-conditions:	The Parking Customer is logged in to the system
Post-conditions:	Mulligan system created a new parking event for the Parking Customer's vehicle in a the given space in the Start state.
Trigger:	Parking Customer selected "Start Parking" in the Mulligan customer app
MSS	1. Mulligan system shows a screen to enter the parking space number. 2. Parking Customer enters a parking space number 3. Mulligan system validates the parking space number. 4. Mulligan system ensures that there are no other parking events for the Customer's vehicle in the Start state. 5. Mulligan system starts a new parking event for the parking space with Customer's vehicle. 6. The Mulligan customer app shows a "Parking Started" message.
Branch A:	Alternative from step 3 of the MSS: The parking space entered isn't valid 3A1 Mulligan app shows an invalid parking space message. 3A2 Return to step 1 of the MSS.
Branch B:	Alternative from step 4 of the MSS: There is another parking event for the customer in the Start state. 4B1 Mulligan runs SUC-2 (Stopping a Parking Event) for the given customer. 4B2 Return to step 5 of the MSS.

System Use Case 2:	Stopping a Parking Event
Actors and interests:	• Parking Customer: To stop a parking event in the Start state • Spontaneous: To automatically stop an parking event.
Stakeholders:	
Pre-conditions:	If the actor is the Parking Customer, the Parking Customer is logged into to the system
Post-conditions:	Mulligan system has stopped registering the Parking Customer's vehicle as being legally parked.
Trigger:	1. Parking Customer selected "Stop Parking" in the Mulligan customer app 2. Mulligan system detects that the customer has a Start parking event for another space.
MSS	1. Mulligan system finds the currently Started parking event for the customer's vehicle. 2. Mulligan system stops the Started parking event with the end time being the current time. 3. Mulligan app shows a "Parking Stopped" message. 4. Mulligan system issues a payment transaction event in the queuing system.
Branch A:	Exception from step 1 of the MSS: There is no parking event for the customer's vehicle in the Start state 1A1 Mulligan system sends an error message that there is no open parking event 1A2 End of use case.

System Use Case 3:	Retrieving list of parking events
Actors and interests:	Parking Customer: To receive a list of previous parking events
Stakeholders:	Municipality Officer: Let customers appeal parking tickets incorrectly received
Pre-conditions:	The Parking Customer is logged in to the system
Post-conditions:	The Parking Customer has received a list of parking events for his vehicle.
Trigger:	Parking Customer selects "Get Parking Events List" on the Mulligan app
MSS:	1. Mulligan system checks its database for parking events for the customer's vehicle. 2. Mulligan app displays a table of parking events. For events that are stopped, it shows the beginning time, end time, and parking space id. For events that are not stopped, it shows beginning time and parking space id. 3. Mulligan app displays the total amount of money owed by the Customer for all parking events.
Branch A:	Exception from step 2 of the MSS: There are no parking events for the customer 2A1 Mulligan app shows a message that there are no events for the customer's vehicle. 2A2 Mulligan app shows a zero total amount of money owed. 2A3 End of the use case.

System Use Case 4:	Investigating parked vehicle
Actors and interests:	Parking Enforcement Officer (PEO): To check if a vehicle is currently legally parked
Stakeholders:	Parking Customer: Ensure that if he started a parking event, he will not get a ticket for the parking space
Pre-conditions:	The Parking Enforcement Officer is logged in to the system
Post-conditions:	1. The PEO received a message whether the given vehicle is legally parked. 2. Mulligan recorded the query and its parameters in the system log.
Trigger:	PEO selects "Check Vehicle" on the Mulligan PEO app
MSS:	1. PEO enters the vehicle number and parking space id into the Mulligan PEO app 2. Mulligan system checks that the vehicle number and parking space id are valid 3. Mulligan system finds a parking event for the vehicle in the given space that is in the Start state and for which the maximum allowed parking time has not passed. 4. Mulligan app shows a "Parking Ok" message to the PEO. 5. Mulligan system records the query information (current time and date, vehicle number, parking space id, response) in the system log.
Branch A:	Alternative from step 2 of the MSS: The vehicle number or parking space id are not valid 2A1 Mulligan app shows an appropriate error message. 2A2 Return to step 1 of the MSS.
Branch B:	Alternative from step 3 of the MSS: There is no parking event for the given vehicle and parking space in the Start state. 3B1 Mulligan app shows a "Parking Not Ok" message to the PEO 3B2 PEO begins citation issuance use case (SUC-5) 3B3 Return to step 5 of the MSS.

System Use Case 5:	Citation Issuance
Actors and interests:	Parking Enforcement Officer (PEO): Issue a citation for an illegally parked vehicle
Stakeholders:	Municipality Officer: Ensure parking is paid for
Pre-conditions:	PEO inspected a vehicle and received a Not Ok response
Post-conditions:	1. The PEO has issued a citation. 2. Mulligan recorded citation in the appropriate queue.
Trigger:	PEO selects “Issue citation” on the PEO app
MSS:	1. PEO enters the vehicle number, parking space id, and citation cost in NIS 2. PEO adds current query time to the citation 3. Mulligan system sends a citation message in its queuing system
Branch A:	Alternative from step 2 of the MSS: The vehicle number or parking space id are not valid 2A1 Mulligan app shows an appropriate error message. 2A2 Return to step 1 of the MSS.

System Use Case 6:	Parking Transaction Report
Actors and interests:	Municipality Officer (MO): To see a report of all transactions
Stakeholders:	
Pre-conditions:	The Municipality Officer is logged in to the system
Post-conditions:	The MO received a list of paid transactions.
Trigger:	MO selects “Get Transaction Report” on the Mulligan MO app
MSS:	1. Mulligan system retrieves all transactions from the transaction queue 2. Mulligan app shows the list of transactions, including vehicle numbers, parking space ids, parking zones, dates, start times, stop times, and amount of money owed. For events that are in the Start state, the Stop time is left blank.
Branch A:	Alternative from step 2 of the MSS: There are no parking events 2A1 Mulligan app shows an empty list to the MO 2A2 End of use case

System Use Case 7:	Parking Citation Report
Actors and interests:	Municipality Officer (MO): To see a report of all citations
Stakeholders:	
Pre-conditions:	The Municipality Officer is logged in to the system
Post-conditions:	The MO received a list of citations.
Trigger:	MO selects "Get Citation Report" on the Mulligan MO app
MSS:	1. Mulligan system retrieves all citations from the citation queue 2. Mulligan app shows the list of citations, including vehicle numbers, parking space ids, parking zones, dates, inspection times, and amount of money owed.
Branch A:	Alternative from step 2 of the MSS: There are no citations 2A1 Mulligan app shows an empty list to the MO 2A2 End of use case

4 System Architecture

The system architecture is distributed and consists of several computers:

Node	Task
Customer mobile	Computer the customer uses to start and stop parking
PEO mobile	Computer the PEO uses to inspect vehicles and issue citations
Mulligan DB server	Database server that stores started and stopped parking records
Queue server	Receives updates about completed (stopped) parking transactions and issued citations. Manages 2 queues, one for completed transactions and one for citations.
Municipality PC	Allows MO to generate reports about completed transactions and issued citations. Interacts with the Queue server

The physical and logical architectures for Mulligan are shown in Figures 1 and 2.

4.1 Message Broker

Mulligan will use RabbitMQ as a message broker between Customers, PEOs, and MOs. RabbitMQ (`https://www.rabbitmq.com/`) is an open source and free message broker system used in many projects across the industry. Part of the learning outcomes for this assignment will be gaining familiarity with building a simple RabbitMQ server, two queues, producers, and consumers. You can find lots of online tutorials for Java, including a few walkthrough tutorials at `https://www.rabbitmq.com/tutorials/tutorial-one-java` that should suffice for this assignment.

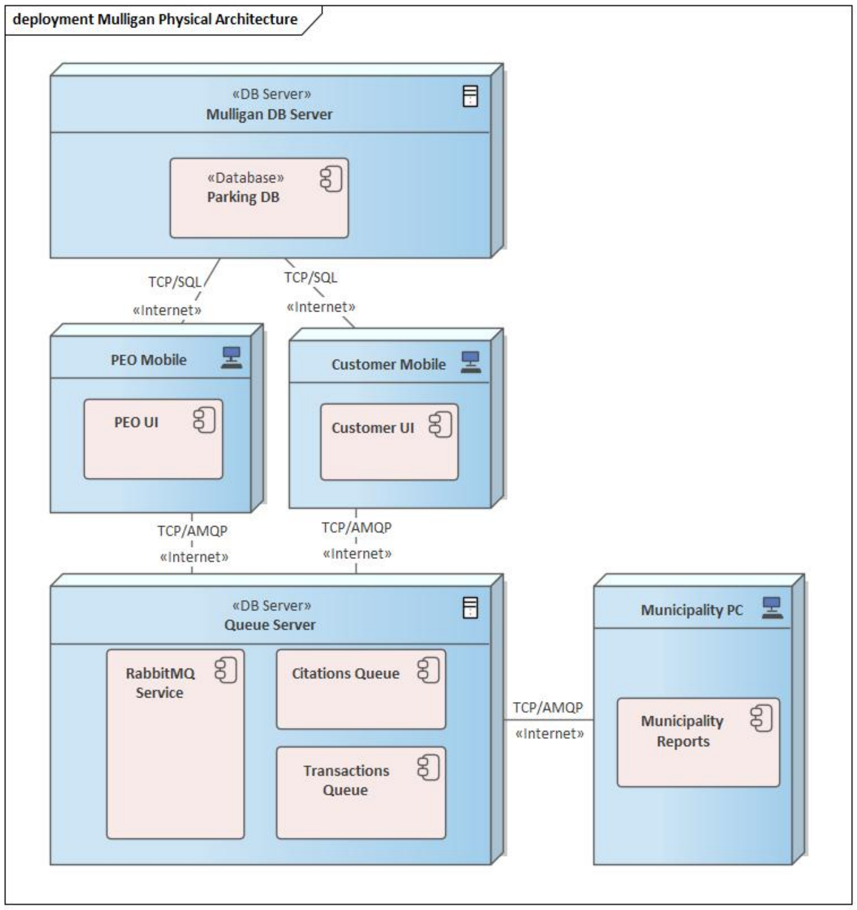

Figure 1: Mulligan deployment diagram

5 Step 1: First build with communication

In the first step of the assignment, you will implement the use cases above using the architecture shown in Figure 1. There are four major tasks to be performed here. They are designed to be distributed to four different people in your team. Each team must specify who took each task. Each individual will be given a personal grade for their part in the work as well as a collective team grade for the collective effort. In future iterations, team members will swap task roles, so be sure that everything is documented clearly and written cleanly.

5.1 Task 1: User Interfaces

There are 3 user interfaces on the client side:

1. The Customer UI
2. The PEO UI
3. The MO UI

What to do: Build the three apps' user interfaces using Java FX. The mockups should have clickable buttons and fields and offer functionality using a Controller class.

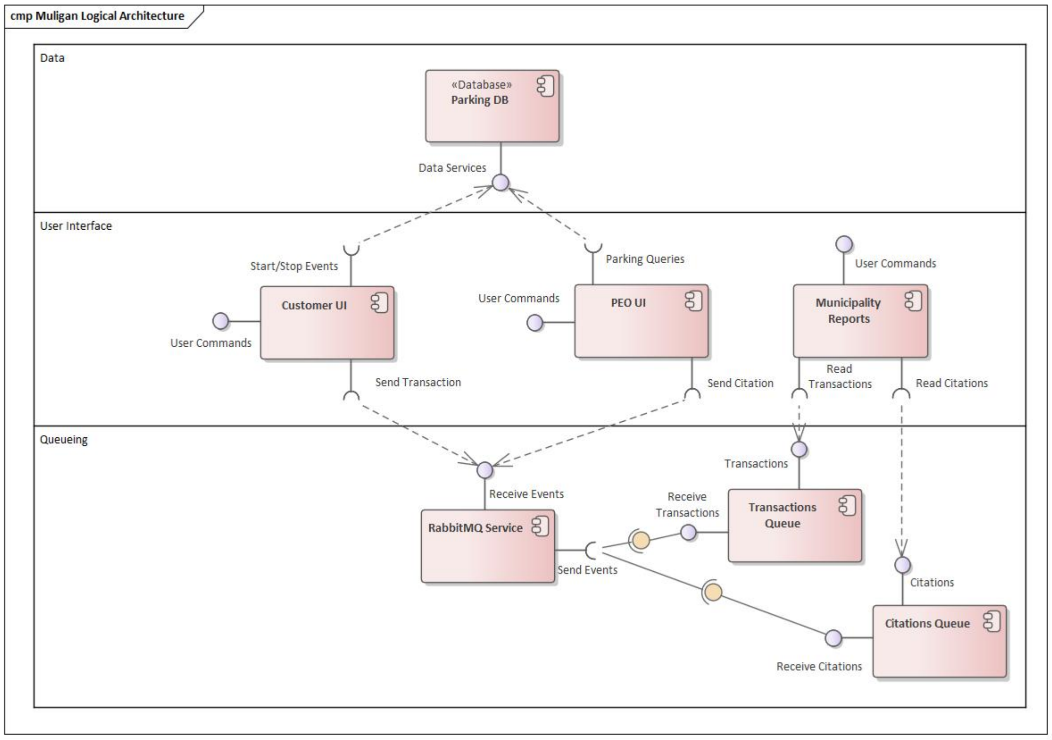

Figure 2: Mulligan logical architecture

5.2 Task 2: Queue Server

The queue server must include a RabbitMQ server instance and 2 queues, one for transactions and one for citations. The server must accept messages from the customer and PEO UIs and route them to the correct queues.

What to do:

1. Build the server using RabbitMQ and two queues
2. Make the server available to work with the UIs.

5.3 Task 3: Database and Storage

The database server must store information about vehicles, parking spaces, parking zones, and parking events. Create a database with tables or documents that support that. Add sample data with at least 10 vehicles, 10 parking zones with various parking costs, and 100 parking spaces.

Create queries and retrieval mechanisms to allow the PEO and customer UIs to store and retrieve data.

What to do:

1. Create the database and schemas
2. Insert the sample data
3. Write queries as necessary for the UIs to work
4. Create a design document for the database, including the schemas used and important queries.

5.4 Task 4: Documentation, DevOps, Build and Test

Document all classes with JavaDoc. Write meaningful comments on all classes and public methods.

The clients and server will be stored in GitHub and written in Java. Prepare the foundation projects in the GitHub repository so that they can be built automatically and tested automatically as much as possible.

To ensure correctness, you will write unit tests for all public and protected methods. Each unit test must include success and failure boundary tests.

Prepare test scripts for testing each use case listed above.

What to do:

1. Document all classes and public methods with Javadoc.
2. Create a design document for how the queue server operates, including decisions about how events are built, how routing is done, and how it interacts with the clients and other servers.
3. Create the GitHub repository for the team and put in Java projects for the client apps and the servers.
4. Create GitHub actions that build each project automatically.
5. Create JUnit tests for all public and protected methods.
6. Create a testing plan that has acceptance tests. The testing plan must consist primarily of tables with the following columns. The row shown here is a sample. Remove it and replace with your actual information. Create a table for each of the use cases above, covering the MSS, pre-condition, post-conditions, and all branches.

#	Artifact Tested	Pre-conditions	Test Steps	Expected Result	Passed?
1	Customer app	Customer logged in	(a) Click on Stop Parking (b) ...	Parking stopped shown	Yes
2	...	...	...	...	...

6 For 3 person teams

If your team has only 3 people, distributed the documentation and DevOps task among the team members appropriately. The UI person should write the documentation and unit tests for the UIs. The queue server person should write the documentation and tests for the queue server. The database person should write the tests and documentation for the database.

7 Submission and Grading

Submit all of your work using the repository opened for you in GitHub Classroom. Submit all documents and code in the repository.

Grading:

1. 50% individual work on the task per person
2. 50% overall group grade