Course: Distributed Systems - Semester 1 of 5785
Assignment 3

Directions

A. Due Date: 2 February 2025 at 11:00am

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

C. Code for this assignment (Ass3) 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) Consensus protocol design document

   (g) 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 3: Recommender and Consensus

In the previous assignments we set up a parking architecture with user interfaces and distributed storage. In this assignment we will explore features related to recommending and consensus gathering in distributed systems. The physical architecture will include a new node - the recommender server cluster. The resulting physical architecture is shown in Figure 1. The logical architecture has been modified accordingly, including the new recommender component in Figure 2.

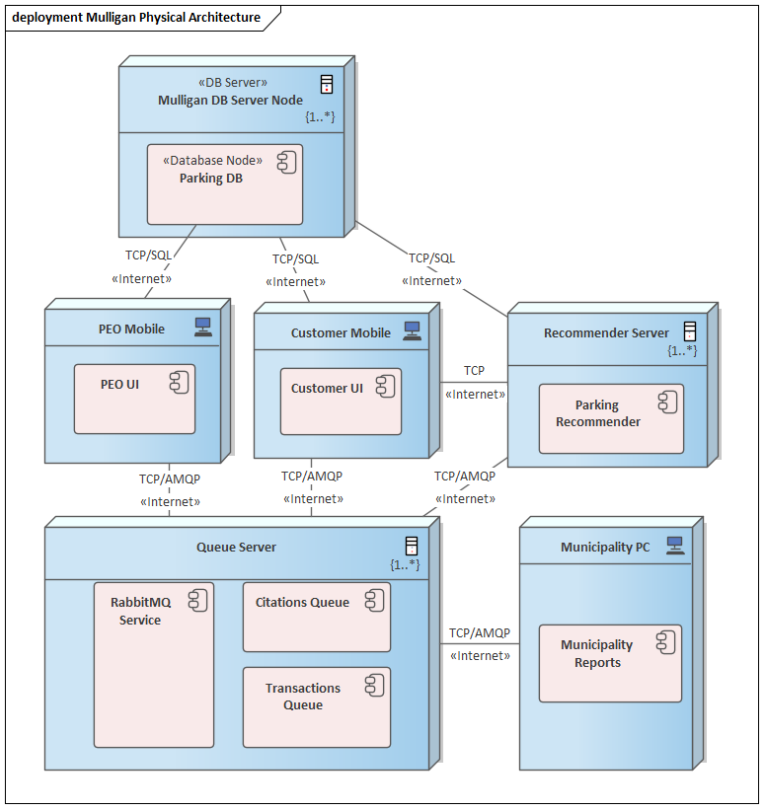

Figure 1: Mulligan with recommender physical architecture

1 Recommending

The job of the new component is to recommend the best parking space to a client who is trying to find out where to park. The customer provides an initial idea of where he wants to park - a parking zone and parking space number - and can use the system to search for a recommended parking space. The recommender service takes the provided parking space and generate a list of good or better parking spaces based on availability and the number of parking citations issued for cars parked those spaces. The user interaction is detailed in the following new use case.

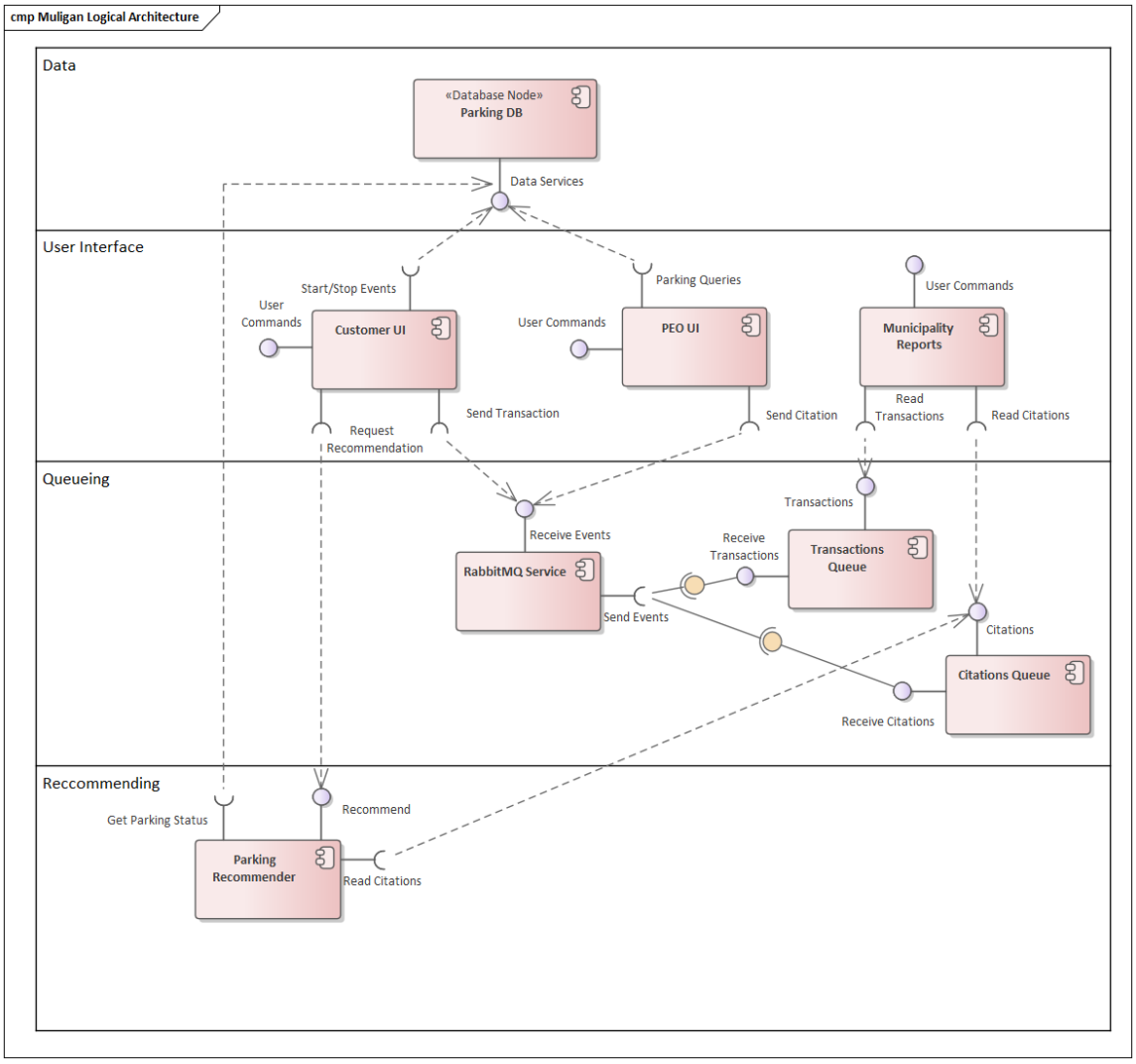

Figure 2: Mulligan with recommender logical architecture

System Use Case 8:	Parking Space Recommendation
Actors and interests:	Parking Customer: To find the best options for parking
Stakeholders:	
Pre-conditions:	The Parking Customer is logged in to the system
Post-conditions:	The customer has received a list of best parking spaces for his request.
Trigger:	Parking Customer selected "Recommend 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 finds the nearest available parking space(s) in the parking zone with the smallest number of issued citations. 5. Mulligan system displays the selected parking space(s) and the number of citations issued for each.
Branch A:	Alternative from step 4 of the MSS: There are multiple available parking spaces in the parking zone with the minimum number, not including the entered parking space. 4A1 Mulligan returns the available parking space(s) with the minimum number of citations with the smallest absolute value distance from the desired parking space. 4A2 Return to step 5 of the MSS.
Branch B:	Alternative from step 4 of the MSS: There are multiple available parking spaces in the parking zone with the minimum number, including the entered parking space. 4B1 Mulligan returns the entered parking space. 4B2 Return to step 5 of the MSS.
Branch C:	Alternative from step 4 of the MSS: No parking spaces are available. 4C1 Mulligan returns an empty list of parking spaces. 4C2 Return to step 5 of the MSS.
Branch D:	Alternative from step 3 of the MSS: The parking space provided is invalid. 3D1 Mulligan shows an appropriate error message. 3D2 Return to step 2 of the MSS.

The selection of which parking space(s) are the best is a bit complicated to describe. To help understand the algorithm, I prepared a flow chart of the decision making process as well as several concrete examples. The flow chart is shown in Figure 3. The examples are found in a separate PDF file on Moodle.

2 Recommender Cluster and Consensus

To experiment with consensus algorithms, we will build a cluster of recommender servers. The customer will contact one recommender (could be any member of the cluster) and the cluster will perform a consensus algorithm to reach a decision. The algorithm has roughly three steps:

1. All cluster nodes will calculate the list of recommended parking spaces
2. All cluster nodes will send their lists to the cluster leader
3. The cluster leader will use a majority vote algorithm to decide the list to return.

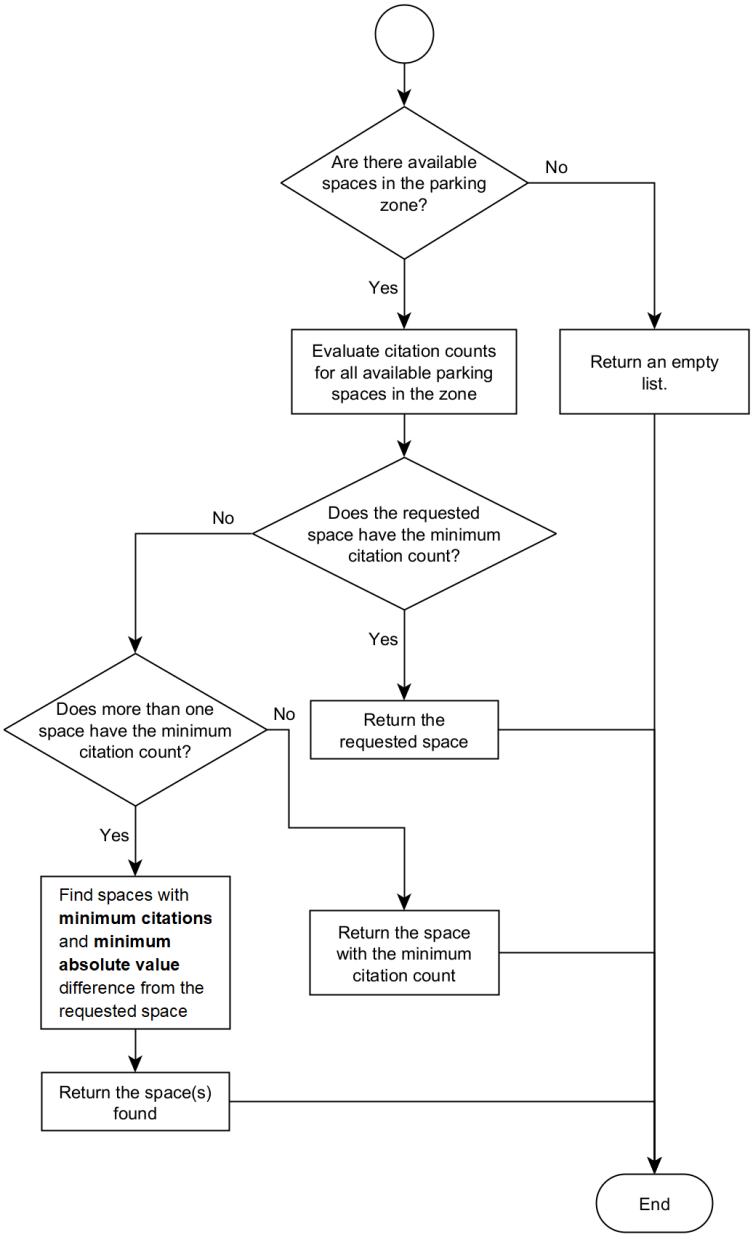

Figure 3: Flow chart for choosing spaces with smallest citation count.

The leader will compare the decisions from all nodes, including itself to decide what to return. If there is no majority in the decision, the customer is returned a failure message.

2.1 Consensus Examples

Assume we have a cluster of 3 recommender servers who are working to provide a result for a customer who wants to park in parking space 3 in parking zone A. Server1 is the leader in the following cases.

2.1.1 Example: All Agree

The following results are returned from the recommenders:

Server	Results
Server1	3;1
Server2	3;1
Server3	3;1

The leader returns the list that all agree on: 3;1 (parking space 3 with 1 parking citation issued for it).

2.1.2 Example: Majority Agree

The following results are returned from the recommenders:

Server	Results
Server1	3;1
Server2	4;1
Server3	3;1

The leader returns the list that the majority agree on: 3;1 (parking space 3 with 1 parking citation issued for it).

2.1.3 Example: No Agreement

The following results are returned from the recommenders:

Server	Results
Server1	3;1
Server2	4;1
Server3	5;1

The leader returns a failure message since a majority did not vote on a shared list.

2.1.4 Example: No Agreement

The following results are returned from the recommenders:

Server	Results
Server1	3;1, 4;1
Server2	4;1
Server3	4;1, 5;1

The leader returns a failure message since a majority did not vote on a shared list.

2.1.5 Example: No Agreement

The following results are returned from the recommenders:

Server	Results
Server1	3;1
Server2	3;1, 5;1
Server3	5;1

The leader returns a failure message since a majority did not vote on a shared list.

2.1.6 Example: Missing Response

The following results are returned from the recommenders:

Server	Results
Server1	3;1
Server2	(No Response)
Server3	3;1

The leader returns the list that the majority agree on: 3;1 (parking space 3 with 1 parking citation issued for it).

2.1.7 Example: Missing Responses

The following results are returned from the recommenders:

Server	Results
Server1	3;1
Server2	(No Response)
Server3	(No Response)

The leader returns a failure message since a majority did not vote on a shared list.

2.1.8 Example: Missing Response and No Agreement

The following results are returned from the recommenders:

Server	Results
Server1	3;1
Server2	4;1
Server3	(No Response)

The leader returns a failure message since a majority did not vote on a shared list.

2.2 Consensus Testing

To enable proper testing, add a feature causing a recommender to intentionally return incorrect results when queried. That will allow testing of the consensus algorithm more clearly.

3 Step 3: Consensus

There are three major tasks to be performed here. They are designed to be distributed to 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.

3.1 Task 1: User Interface Change

The client user interface should change to include support for the recommender interaction. The other UIs should not need to be changed.

What to do:

1. Modify the user interface to work properly with the recommender cluster as defined.

3.2 Task 2: Recommender Server

Create the recommender server code that can receive messages and generate response lists.

What to do:

1. Build the recommender server.
2. Add access to the database and RabbitMQ server as necessary.

3.3 Task 3: Consensus Protocol

Create a consensus protocol that allows the recommender servers to communicate and coordinate their responses. Ensure that the protocol works even when there isn't consensus and when some servers don't respond.

What to do:

1. Create the consensus protocol
2. Implement the consensus protocol in Java
3. Create a protocol description document that includes the messages sent and steps taken by the servers when producing a recommendation

3.4 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.

Update the testing plan to include the following cases:

1. Failure of 1 recommender server
2. Failure of 2 recommender servers
3. 1 Recommender server returns incorrect answers
4. 2 Recommender server return incorrect answers

4 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 recommender server person should write the documentation and tests for it. The protocol person should write the protocol design document.

5 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