CSE 564: Software Design
Course Project

Spring 2022

SMART PARKING SYSTEM

Name 1	Name 2	Name 3	Name 4
Karan Dabas	Pranav Toggi	Siddharth Kale	Yinghua Li
Program of Study	Program of Study	Program of Study	Program of Study
MS CS	MS CS	MS CS	MS CS
School of Computing and Augmented Intelligence			
Arizona State University, Tempe, AZ, USA, 85281			

Table of Contents

	Karan Dabas	**Pranav Toggi**	**Siddharth Kale**	**Yinghua Li**		
Parts	**% Effort**	**% Effort**	**% Effort**	**% Effort**	**Points ****	**Earned Points**
Problem description	100	100	100	100	5	
SRC design specifications with description	100	100	100	100	25	
UML design specifications with description	100	100	100	100	25	
Implementation	100	95	100	100	10	
Experiments & results	100	100	100	100	10	
Conclusions	100	90	100	100	5	
Demonstration	100	100	100	100	10	
Presentation	100	100	100	100	10	
Code quality	100	100	100	100	5	
Report quality	100	100	100	100	5	
Total	100	100	100	100	110	

* Team Representative
** 10 points is bonus

1 Problem Description

There is a major inconvenience in the traditional parking structures that we have. The parking structures don't let the users know if there are any spots available to park. Also it is hard to locate parking structures in a locality that you are unaware of. Google maps is an alternative though it doesn't let you know if the parking is private, available or if there are any spots available or not. Our smart parking system tends to eliminate this problem by letting users find available parking spots using the app. It lets users find the parking spots that are close by and available to use, it provides direction to the spot, it works for multi tier parking structures, and it lets the users reserve parking spots in advance.

1.1 Relation to Cyber Physical System

This section describes how our system will be considered a CPS. For a system to be a CPS it has to fulfill certain criteria. All of those criteria are described below in relation to our problem statement and objectives.

Reactive Computation

A reactive system should interact with its environment in an ongoing manner via inputs and outputs. The physical component of our system is a barrier. This barrier is attached to all the entrance and exit gates of all the parking lots. If a user wishes to enter the parking lot or if the user is to exit the parking lot, the user will interact with the gate. The interaction between the user and the gate will be through a QR code. If the QR code scan is successful, the barrier will be lifted. Thus allowing users to either enter or exit the parking lot. Moreover, our system manipulates the database based on user inputs. One such example would be when a user makes a reservation. The system has to first allot a parking slot to the user for the user's preferred parking lot. Then the system has to manipulate the availability of parking slots for the particular input time. Again when the user exits the parking lot after the stipulated duration of the reservation. The system has to make the spot available for the public again.

Evident from above scenarios, our designed system does interact with the environment (be it physical or cyber) in an ongoing manner, many times the changes in physical components affect the cyber components and vice versa.

Concurrency

Our system supports multiple parking lots which are managed through our system and multiple registered users. Every single one of the parking lots/users can be considered a separate entity in the system. At any given moment there can be n users either trying to enter/exit m parking lots. Moreover there can be k users trying to book j parking spaces at the same time. All of these interactions are facilitated simultaneously by our system. To do so we have made use of a database lock technique that allows users to read and write data to it concurrently as our database server is hosted on a node server that is backed by a mongodb and is accessible to all the users at all times. We are also making use of threads to update UI components based on the user activity such as booking a slot or ending the booked session.

Feedback Control of the Physical World

Our system does have a feedback loop. Take for an example, someone interacts with the system using the app which is a physical entity to book a spot. Then internally system finds the specific parking lot and makes changes to the database to accommodate the booking. When the user arrives at the parking lot, a QR code is generated by the system. This code is then scanned by the scanner, the scanner then interacts with the system to validate the QR

Code. If validation is successful it opens up the barrier. So in this manner a feedback system is created where interactions and processing in the system creates a series of observable physical changes to the physical and virtual database of the system. Similarly when a user exits the parking lot the spot is made available system wide for other users to book and concurrently estimate the charges for the user in a live parallel thread once the booking is made up till the end of session.

Real Time Computation

Our system is very time sensitive. Based on when a user is entering/exiting the parking lot, the spot is made available/unavailable all throughout the system automatically. This spot then is either available for other people to book/or get assigned or blocked. This can cause different behaviors in the system. For example, users are notified about the status of the lots near them. Based on the availability of the parking spots in any particular parking lot, the system automatically has to block entry to the parking lot if all the spots are full.
Moreover, there is feedback of concurrent estimation of the charges that incur to any user current using a parking spot. This charge is updated on their app in real time based on the amount of time they have parked and the current parking charge. These parking charges also fluctuate based on how long the user is at the parking spot. The system has to adapt to this billing structure automatically and has to update the changes locally to the app and also to the central database for record management and payments. An example of this adaptive billing system is that the normal parking charge is $10 per hour, it switches to 15$ per hour if the user is at the parking spot for more than 24 hours.

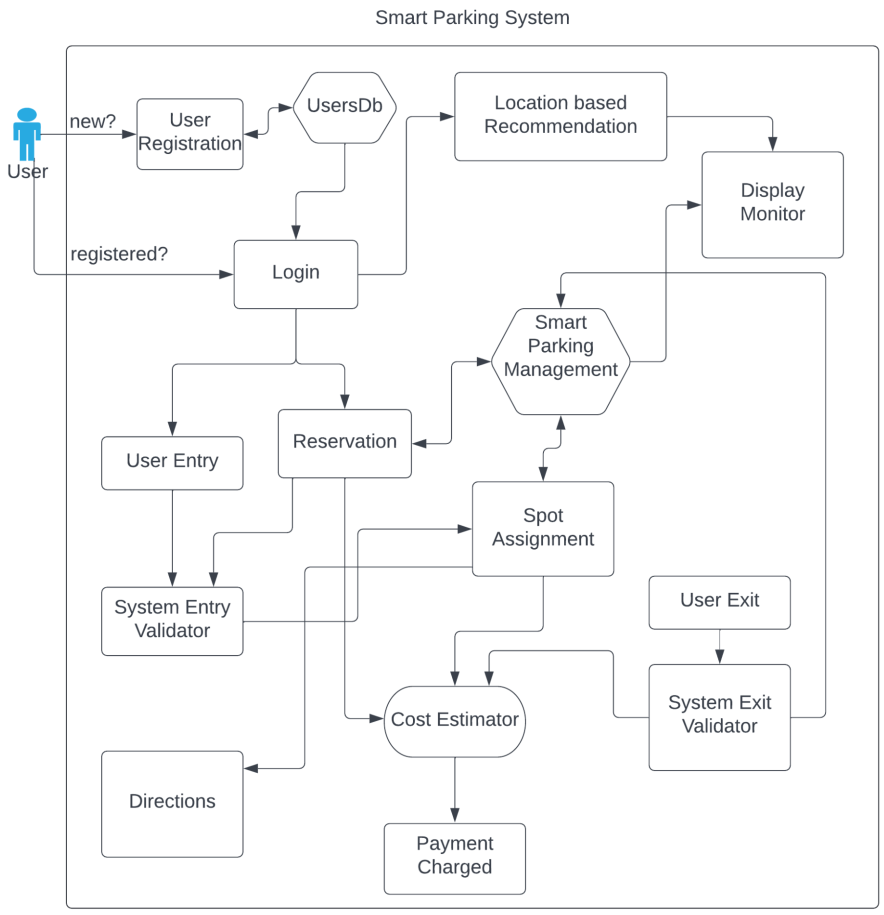
Safety Critical Application

Main issues with a traditional parking lot is the entry of unwanted people who want to cause harm or steal cars, or even users who park there but have no intention of paying the parking fees. This makes the parking lot unsafe for the other users and their owned cars.
Our system doesn't let unauthorized persons enter the parking lot no matter what. A user has to be registered with our system before using our services. A valid credit card has to be entered in order to successfully register. And there should be a valid payment method entered in the system before the user is let into any parking lot. Moreover at the entrance of any parking lot a user validation is done through a unique QR code that encodes userId, time of entry/reserve and the parking lot Id.
Furthermore, storing the payment method at the time of registration, and verifying the payment method ensures that we don't have users who don't intend to pay. An issue that occurs quite frequently with the traditional parking lots.

2 Design

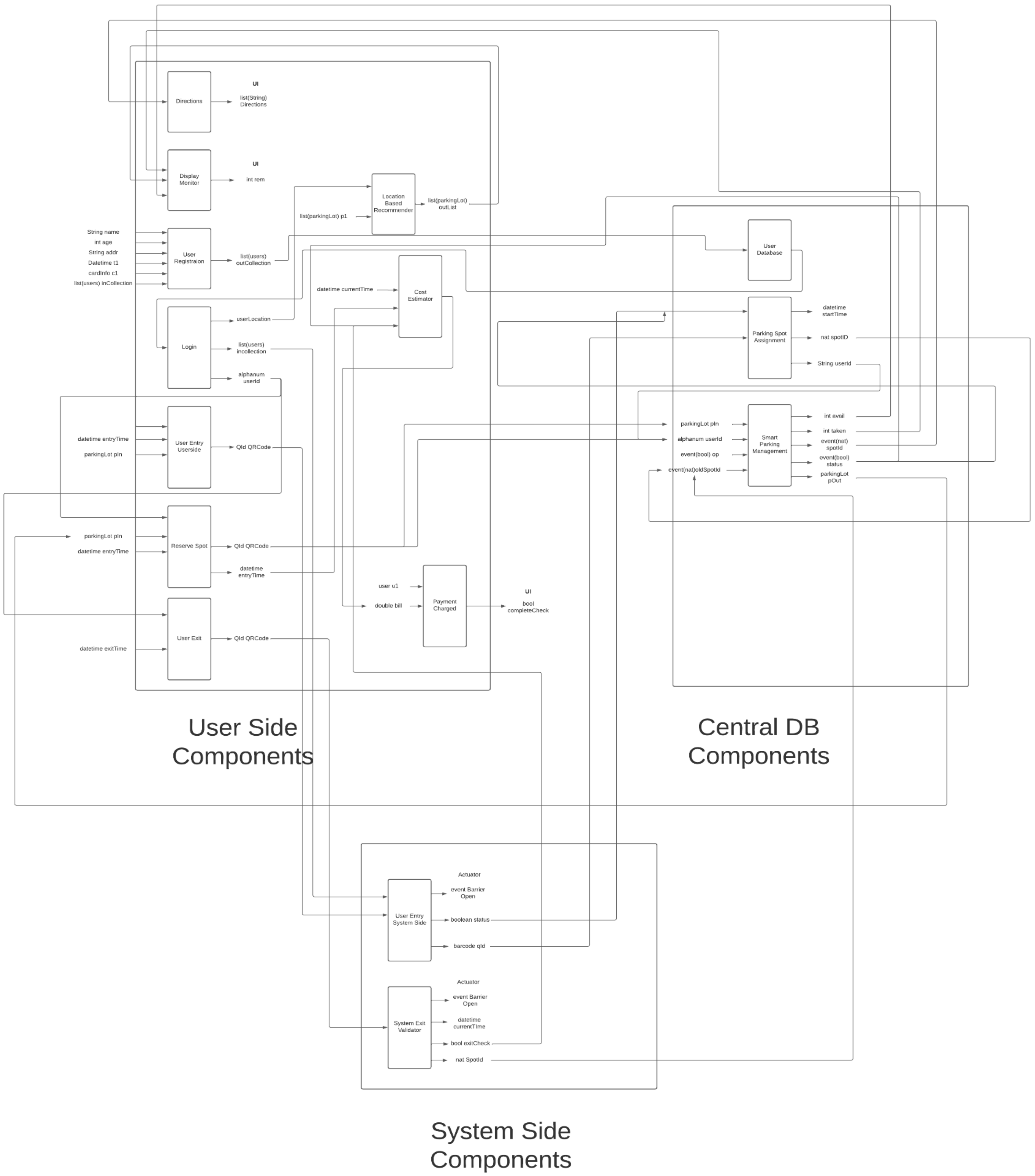
It describes an overview of a design for the project topic developed using formal and semi-formal methods.

The first figure below is an abstract overview of all the components of our smart parking system. The use case of such figures showcase the connectivity among components at high level.



The figure below gives a more detailed description of the system with a complete display of types of inputs and outputs coming in and going out of the blocks. We have also divided the components into three broad categories to show a clear level of separation between them using intra and inter communication across subsystems. This is an interface level communication overview of our entire system. It also has a hidden component of UI which you may see at places, this means its a message sent to our UI activities.

Software Design (CSE 564) **Smart Parking System**



2.1 SRC design specifications with descriptions

SRC design specifications for individual components of the Smart Parking System are listed below, throughout the project you will see some data types that are custom to our needs and the exact details of the same can be seen in the variable tables attached with each SRC. We made use of task graphs and have listed out the task dependencies with possible reactions and execution schedules.

1. User Registration: A registration page that takes in name, age, address, time and card info and outputs a database object that is written to the central DB. The user is registered only if they are 18 or above.

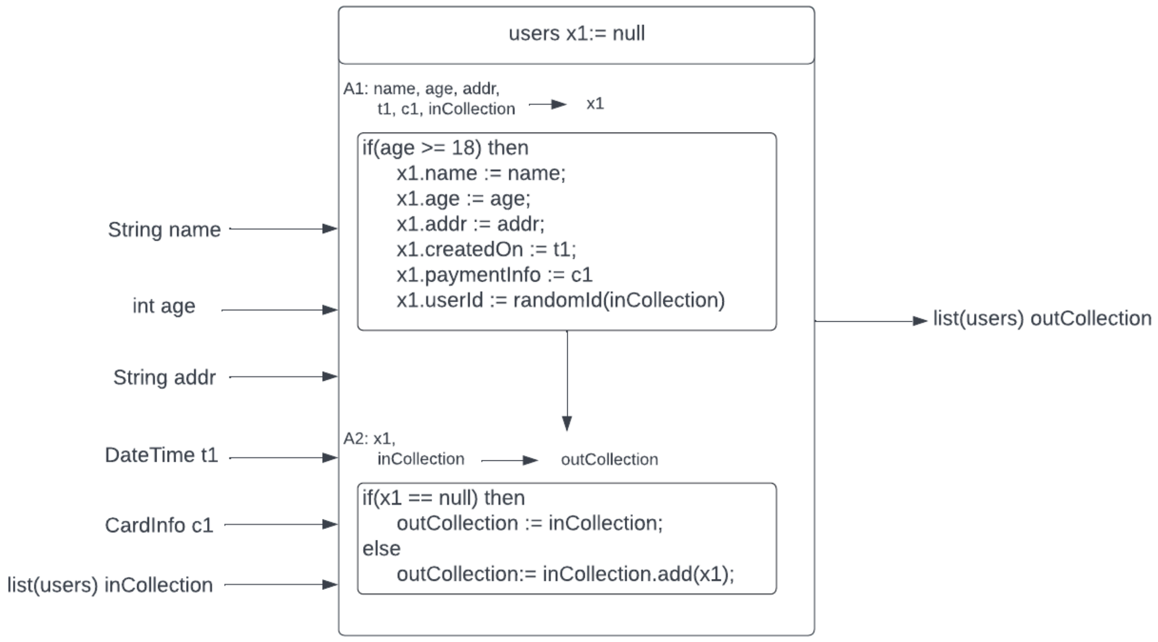
Here I is input set, S is state set, O is output set, Init is the initial setting of state variables and React is the reaction component specified figure below.

I = {name, age, addr, t1, c1, inCollection}; S = {x1} ; O = {outCollection}
[Initi] : users x1:= null;

Inputs for **User Registration**			
Name	Range of Values	Initial Value	Unit
name	[A-Z]*20	None	string
age	0-199	18	age
addr	[A-Z]*100	None	string
time	MM:1-12 YY: 2022-2030 HH: 1-24 MM: 1-60 SS: 1-60	None	datetime
c1	CardInfo contains following: (1) card number (2) name on card (3) valid thru date(YYMM) (4) security code number	None	cardInfo which is a collection of integer numbers, alphabets.
inCollection	List of users object	empty list	-

Outputs for **User Registration**			
Name	Range of Values	Initial Value	Unit
outCollections	List of users object	empty list	-

States for **User Registration**			
Name	Range of Values	Initial Value	Unit
x1	users consists of multiple attributes such as name, age, addr, createdOn, paymentInfo and userId	null	-



Possible Reactions:

$$(x1)\xrightarrow{(name, age, addr, t1, c1, inCollection\ )/outCollection} (x1\ *)$$

Task Dependencies:

A1 < A2

Execution Schedule:

A1,A2;

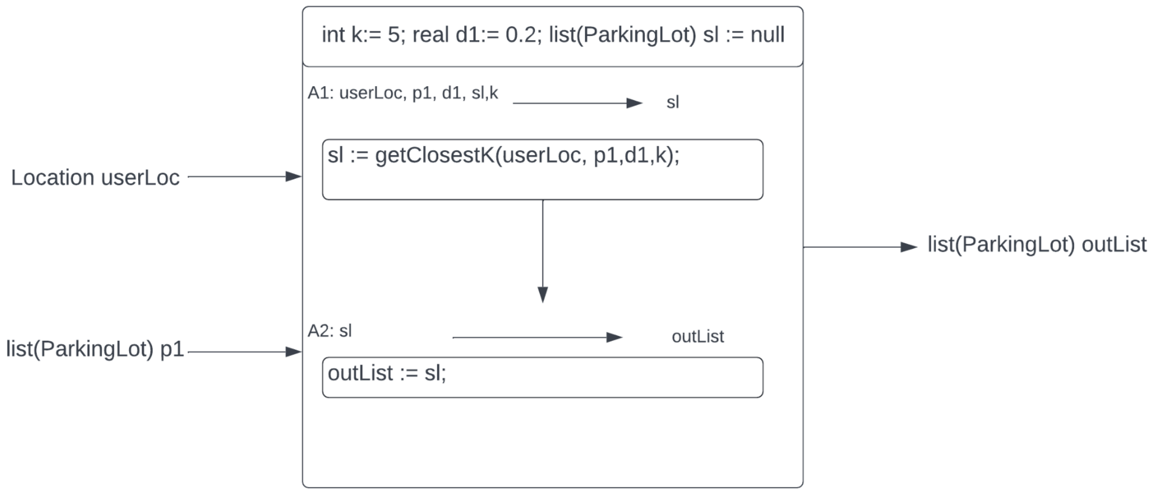
2. Location based recommendation: It takes two inputs: users destination location and our list of parking structure locations and returns a list of K options of parking structure that are near the user destination and have parking availability.

I = {userLoc, p1}; S = {k,d1,sl} ; O = {outList}
[Initi] : int k := 5; real dist:= 0.2; list(ParkingLot) sl = null

Inputs for **Location based recommendation**			
Name	Range of Values	Initial Value	Unit
userLoc	latitude -90; longitude -180 to longitude 180	None	location
p1	List of parking lots connected to the system. Each parking lot has a location(latitude and longitude)	None	locations

Outputs for **Location based recommendation**			
Name	Range of Values	Initial Value	Unit
outlist	List of parking lots connected to the system. Each parking lot has a location(latitude and longitude)	None	locations

States for **Location based recommendation**			
Name	Range of Values	Initial Value	Unit
k	0-10	5	count
d1	0-10	0.2	miles
s1	list of parking lot locations	null	location


int k:= 5; real d1:= 0.2; list(ParkingLot) sl := null
A1: userLoc, p1, d1, sl,k
sl
sl := getClosestK(userLoc, p1,d1,k);
Location userLoc
list(ParkingLot) outList
A2: sl
outList
list(ParkingLot) p1
outList := sl;

Possible Reactions:

$$(k, d1, s1) \xrightarrow{(userLoc, p1)/(outList)} (k, d1, s1*)$$

Task Dependencies:

A1 < A2;

Execution Schedule:

A1,A2

3.Parking Entry UserSide: The component is a combinational src that only takes two inputs: a userId and current time. The output is the unique QRcode that a user can use to enter the parking structure.

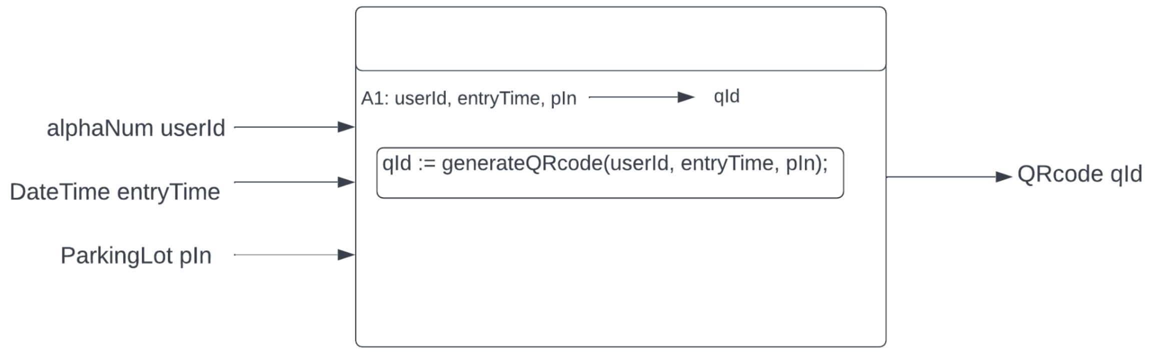
I = {userId, time}; S = {} ; O = {qId}
[Initi] : {} // combinational

Inputs for **User Entry**			
Name	Range of Values	Initial Value	Unit

userId	[0-9][A-Z]*10	A000000000	ID
entrytime	MM:1-12 YY: 2022-2030 HH: 1-24 MM: 1-60 SS: 1-60	None	datetime
pIn	parking lot location(Lat,Lon)	None	location

Outputs for **User Entry**			
Name	Range of Values	Initial Value	Unit
qId	-	-	barcode

States for **User Entry**			
Name	Range of Values	Initial Value	Unit
-	-	-	-


A1: userId, entryTime, pIn → qId
qId := generateQRcode(userId, entryTime, pIn);
alphaNum userId
DateTime entryTime
ParkingLot pIn
QRcode qId

Possible Reactions:

$$(null)\xrightarrow{(userId, entryTime, pIn\ )/qId} (null)$$

Execution Schedule:

A1

Execution Schedule:

A1

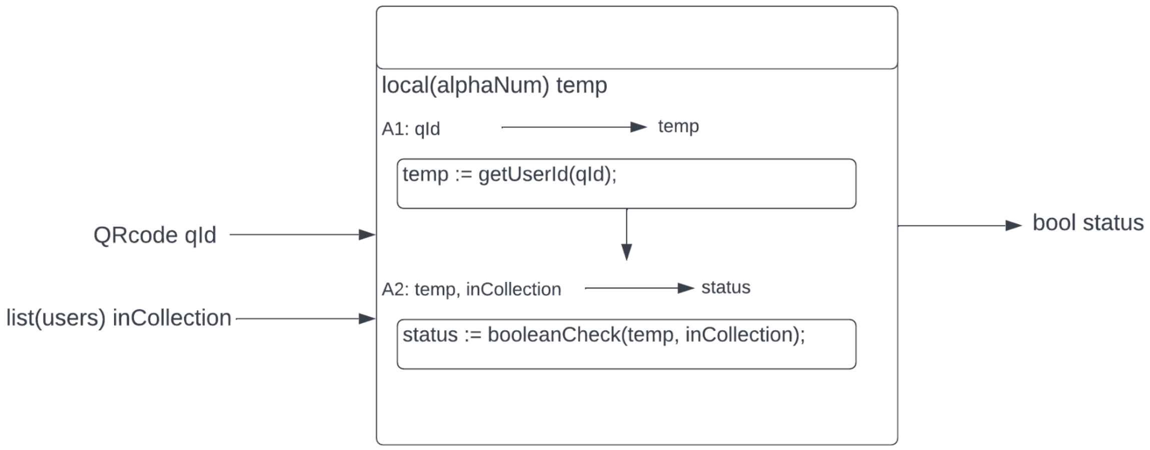
4.Parking Entry SystemSide: This is a system side SRC that is also combinational in nature and takes two inputs: a unique user QRcode and a collection of all user information on the platform and returns true if the QRcode is valid else return false. This src is used to authenticate a given user during entry to the parking structure.

I = {qId, inCollection}; S = {} ; O = {status}
[Initi] : {} // combinational

Inputs for **System Entry Validator**			
Name	Range of Values	Initial Value	Unit
qId	-	-	barcode
inCollection	List of users object	None	-

Outputs for **System Entry Validator**			
Name	Range of Values	Initial Value	Unit
status	{True,False}	None	boolean

States for **System Entry Validator**			
Name	Range of Values	Initial Value	Unit
-	-	-	-



Possible Reactions:

$$(null)\xrightarrow{(qId, inCollection\ )/status} (null)$$

Task Dependencies:

A1 < A2;
Execution Schedule:

A1,A2

5. Parking Spot Assignment : A component used to assign a parking spot in a given parking structure to a user if available, else assign a default code for spot that represents the case when no spots are available and still allow the user to enter the parking structure and direct the user to the nearest exit.

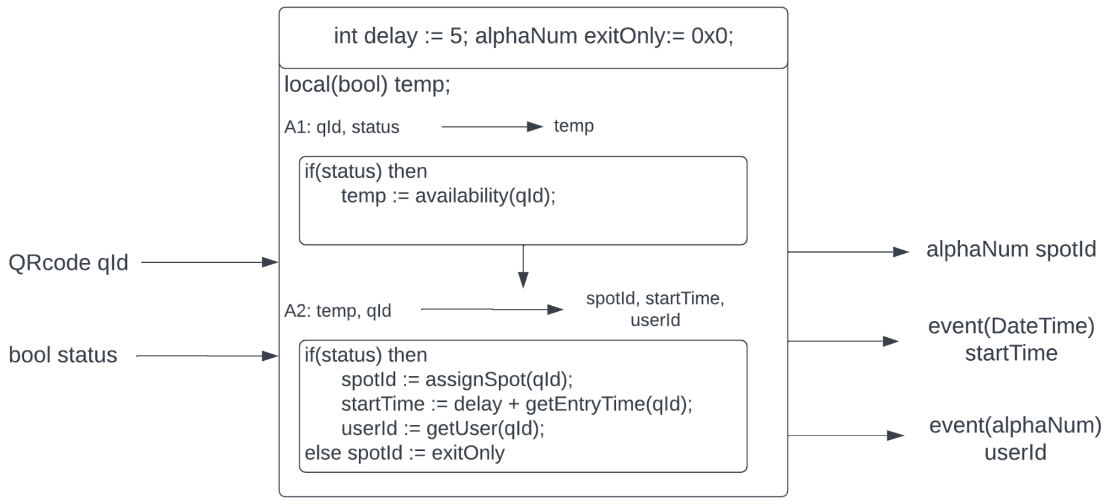
I = {status, qId}; S = {delay, exitOnly} ; O = {spotId, startTime, userId}
[Initi] : int delay:= 5; alphaNum exitOnly= 0x0

Inputs for **Spot Assignment**			
Name	Range of Values	Initial Value	Unit
status	{True,False}	None	boolean

qId	-	-	barcode

Outputs for **Spot Assignment**			
Name	Range of Values	Initial Value	Unit
spotId	[0-9][A-Z]*10	None	ID
startTime	{⊥,dateTime object}	None	datetime
userId	{⊥,[0-9][A-Z]*10}	None	ID

States for **Spot Assignment**			
Name	Range of Values	Initial Value	Unit
delay	5	5	minutes
exitOnly	[0-9][A-Z]*10	0x0	code


int delay := 5; alphaNum exitOnly:= 0x0;
local(bool) temp;
A1: qId, status → temp
if(status) then
temp := availability(qId);
A2: temp, qId → spotId, startTime, userId
if(status) then
spotId := assignSpot(qId);
startTime := delay + getEntryTime(qId);
userId := getUser(qId);
else spotId := exitOnly
QRcode qId
bool status
alphaNum spotId
event(DateTime) startTime
event(alphaNum) userId

Possible Reactions:

$$(delay, exitOnly)\xrightarrow{(qId, status)/(spotId, startTime, userId)} > (delay, exitOnly*)$$

Task Dependencies:

A1 < A2;

Execution Schedule:

A1,A2

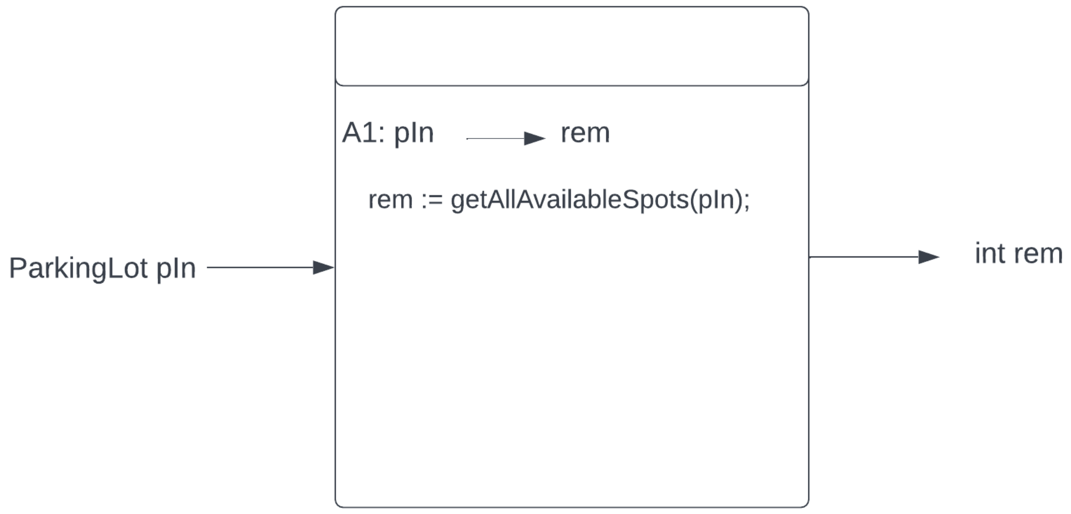
6. Display Monitor: An src designed to showcase the number of available parking spots in the parking structure at all times. This takes only one input, a parking structure id and outputs an integer number.

I = {pIn}; S = {} ; O = {rem}
[Initi] : {} // combinational

Inputs for **Display Monitor**			
Name	Range of Values	Initial Value	Unit
pIn	Latitude and longitude (-180 to 180)	-	location

Outputs for **Display Monitor**			
Name	Range of Values	Initial Value	Unit
rem	integer number	0	count

States for **Display Monitor**			
Name	Range of Values	Initial Value	Unit
-	-	-	-



Possible Reactions:

$$(null)\xrightarrow{(pIn)/(rem)} > (null)$$

Task Dependencies:

A1

Execution Schedule:

A1

7. Directions: SRC will take the output of "Spot Assignment" as input and show the direction to the specific spot on the app.

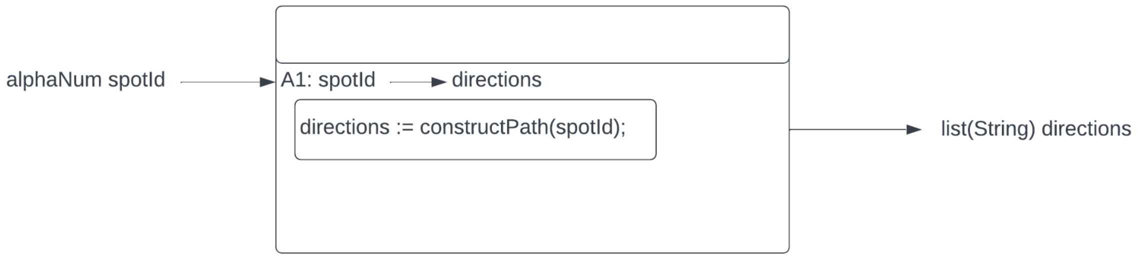
I = {spotId}; S = {} ; O = {list(String) directions} ; [Initi] : {}

Inputs for **Directions**			
Name	Range of Values	Initial Value	Unit

spotId	[0-9][A-Z]*10	None	ID

Outputs for **Directions**			
Name	Range of Values	Initial Value	Unit
directions	list of String directions	directions for exit	-

States for **Directions**			
Name	Range of Values	Initial Value	Unit
-	-	-	-



Possible Reactions:

$$(null)\xrightarrow{(spotId)/(directions)} > (null)$$

Task Dependencies:

A1

Execution Schedule:

A1

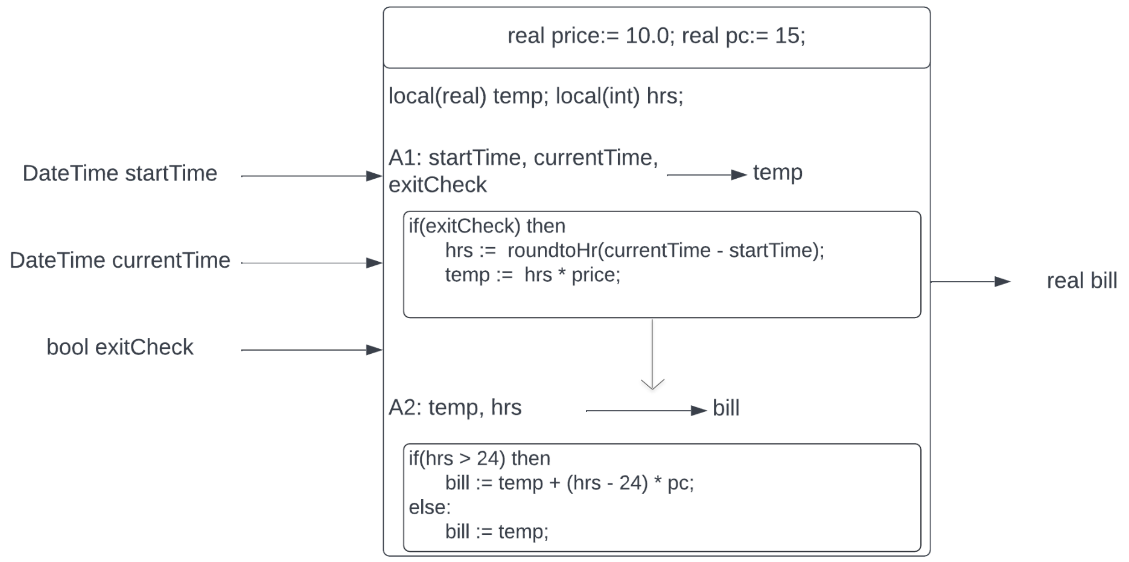
8. Cost Estimator: SRC will take the Leaving time and Starting time of users parking and time "cost per hour" and return a specific amount.

I = {startTime, currentTime, exitCheck} ; S = {price, pc} ; O = {bill} ; [Initi] : {real price := 10.0, real pc := 15} ;

Inputs for **Cost Estimator**			
Name	Range of Values	Initial Value	Unit
startTime	MM:1-12 YY: 2022-2030 HH: 1-24 MM: 1-60 SS: 1-60	None	datetime
currentTime	MM:1-12 YY: 2022-2030 HH: 1-24 MM: 1-60 SS: 1-60	None	datetime
exitCheck	{True,False}	False	bool

Outputs for **Cost Estimator**			
Name	Range of Values	Initial Value	Unit
bill	real number	0.0	dollar

States for **Cost Estimator**			
Name	Range of Values	Initial Value	Unit
price	real number	10.0	dollar
pc	real number	15.0	dollar


real price:= 10.0; real pc:= 15;
local(real) temp; local(int) hrs;
DateTime startTime
DateTime currentTime
bool exitCheck
A1: startTime, currentTime, exitCheck
temp
if(exitCheck) then
hrs := roundtoHr(currentTime - startTime);
temp := hrs * price;
real bill
A2: temp, hrs
bill
if(hrs > 24) then
bill := temp + (hrs - 24) * pc;
else:
bill := temp;

Possible Reactions:

$$(price, pc) \xrightarrow{(startTime, currentTime, exitCheck)/(bill)} (price, pc)$$

Task Dependencies:

A1 < A2;

Execution Schedule:

A1,A2

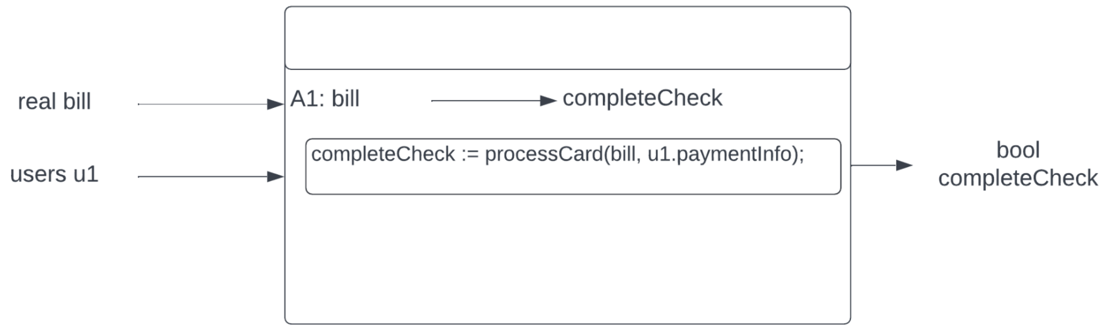
9.Payment Charged: System will take the output of "Cost Estimator" as input and make a charge request to users bank account for the fare amount and return a boolean type.

I = {double amount} ; S = {bool process} ; O = {bool out} ; [Initi] : {bool process = null} ;

Inputs for **Payment Charged**			
Name	Range of Values	Initial Value	Unit
bill	real number	None	dollar
u1	users object to whom the bill will be charged	None	-

Outputs for **Payment Charged**			
Name	Range of Values	Initial Value	Unit
completeCheck	{True,False}	None	bool

States for **Payment Charged**			
Name	Range of Values	Initial Value	Unit
-	-	-	-



Possible Reactions:

$$(null)\xrightarrow{(bill, u1)/(completeCheck)} > (null)$$

Task Dependencies:

A1

Execution Schedule:

A1

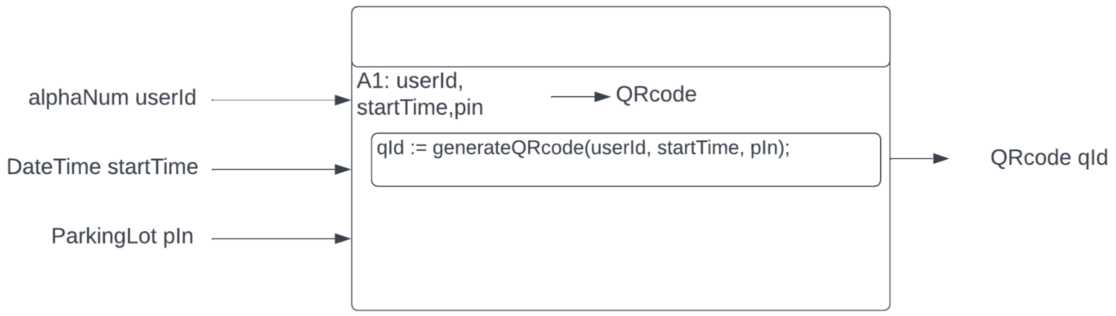
10. Reserve Spot: SRC will take input of userId, starttime, and parking lot info.. It will assign the user a special qr code which the user can scan while entering the parking lot. This QR code will ensure that the user gets a reserved parking spot on the time of the user's arrival. This SRC is a combinational type of SRC so there are no state variables.

I = {startTime,userID, pIn}
O = {qID}

Inputs for **Reserve Spot**			
Name	Range of Values	Initial Value	Unit
userId	[0-9][A-Z]*10	A000000000	ID
startTime	MM:1-12 YY: 2022-2030 HH: 1-24 MM: 1-60 SS: 1-60	None	datetime
pIn	lat/lon: -180 to 180	None	location

Outputs for **Reserve Spot**			
Name	Range of Values	Initial Value	Unit
qId	-	-	barcode

States for **Reserve Spot**			
Name	Range of Values	Initial Value	Unit
-	-	-	-


alphaNum userId
DateTime startTime
ParkingLot pIn
A1: userId, startTime,pin
QRcode
qId := generateQRcode(userId, startTime, pIn);
QRcode qId

Possible Reactions:

$$(null)\xrightarrow{(userId, startTime, pIn)/(qId)} > (null)$$

Task Dependencies:

A1
Execution Schedule:

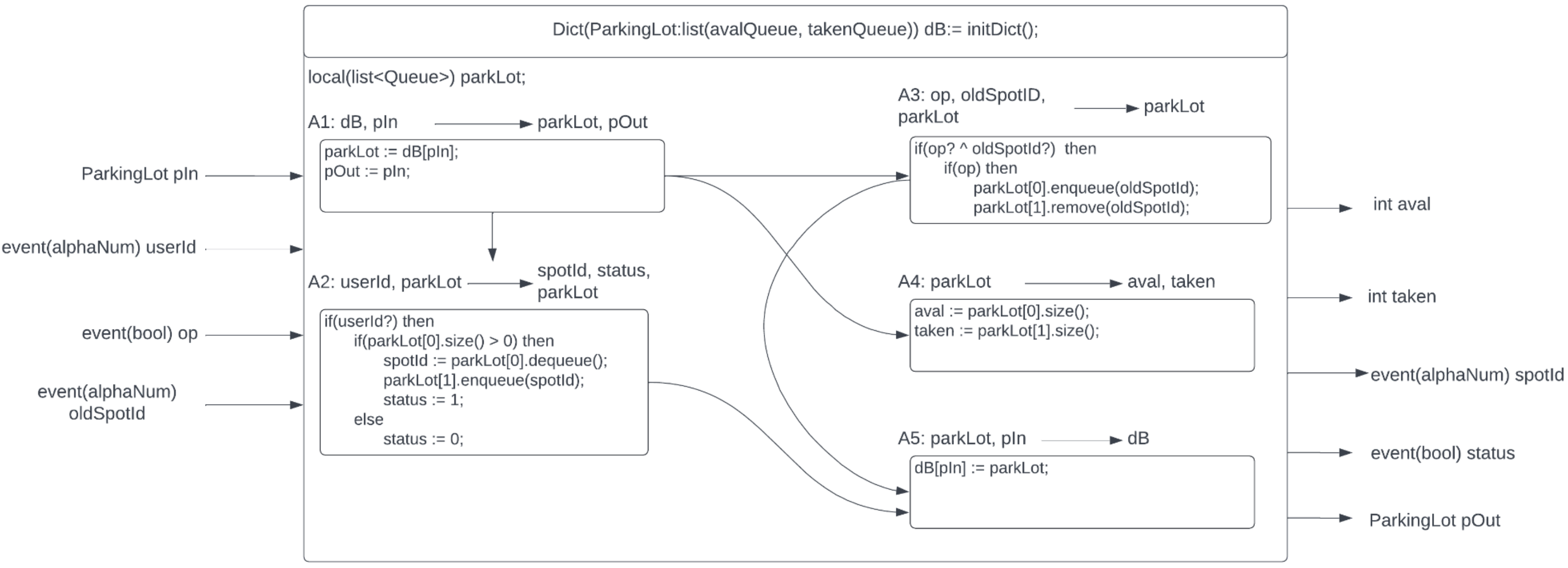
A1

11. Smart Parking Management: This SRC acts as a central management of the parking spots database and act as a hub for various SRCs. This SRC outputs the number of available parking spots, number of already taken parking spots, status, parkinglot location,and spotId. This SRC changes the status of a parking spot based on if a user is currently being assigned the parking spot or if the user is vacating the parking spot. Based on this information spotId is generated which also is of the type event. Similarly for inputs, if a userId is available, then only a particular task will be performed where a parking spot is assigned unavailable from the particular parking lot.

Inputs for **Smart Parking Management**			
Name	Range of Values	Initial Value	Unit
pIn	lat/lon: -180 to 180	-	location
userId	[0-9][A-Z]*10	A000000000	ID
oldSpotId	{absent, [0-9][A-Z]*10}	None	ID
op	{absent, free, taken}	None	action

Outputs for **Smart Parking Management**			
Name	Range of Values	Initial Value	Unit
avail	0-500	0	int
taken	0-500	0	int
spotId	{absent, [0-9][A-Z]*10}	None	ID
status	{True, False}	False	bool
pOut	lat/lon: -180 to 180	-	location

States for **Smart Parking Management**			
Name	Range of Values	Initial Value	Unit
dB	-	-	dictionary


Dict(ParkingLot:list(avalQueue, takenQueue)) dB:= initDict();
local(list<Queue>) parkLot;
A1: dB, pIn → parkLot, pOut
parkLot := dB[pIn];
pOut := pIn;
A2: userId, parkLot → spotId, status, parkLot
if(userId?) then
if(parkLot[0].size() > 0) then
spotId := parkLot[0].dequeue();
parkLot[1].enqueue(spotId);
status := 1;
else
status := 0;
A3: op, oldSpotID, parkLot → parkLot
if(op? ^ oldSpotId?) then
if(op) then
parkLot[0].enqueue(oldSpotId);
parkLot[1].remove(oldSpotId);
A4: parkLot → aval, taken
aval := parkLot[0].size();
taken := parkLot[1].size();
A5: parkLot, pIn → dB
dB[pIn] := parkLot;
ParkingLot pIn
event(alphaNum) userId
event(bool) op
event(alphaNum) oldSpotId
int aval
int taken
event(alphaNum) spotId
event(bool) status
ParkingLot pOut

Possible Reactions:

$$(dB)\xrightarrow{(userId, op, pIn, oldSpotId)/(avail, taken, spotId, status, pOut)} > (dB*)$$

Task Dependencies:

A1 < A2; A1 < A3; A1 < A4; A2 < A5; A3 < A5

Execution Schedule:

A1,A2,A3,A4,A5; A1,A2,A4,A3,A5

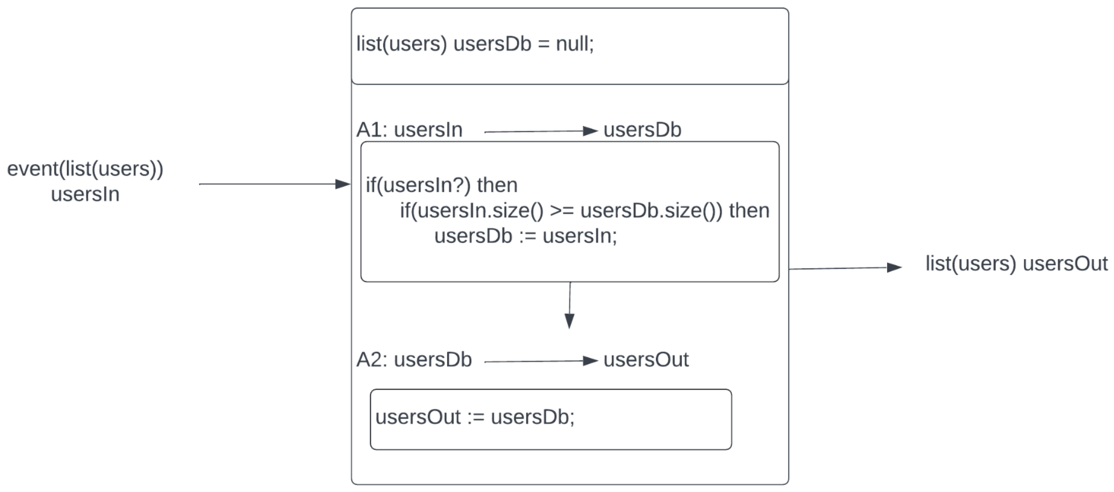
12. Users Store: This is the users database SRC, it takes at most one input of the list of users registered and saves it to the SRCs state and outputs the updated list based on the state variable.

I = {usersIn}
O = {usersOut}
S = {usersDb}

Inputs for **Users Store**			
Name	Range of Values	Initial Value	Unit
usersIn	{⊥, list of users object}	None	

Outputs for **Users Store**			
Name	Range of Values	Initial Value	Unit
usersOut	list of user objects	None	-

States for **Users Store**			
Name	Range of Values	Initial Value	Unit
usersDb	list of users object	Null	-



```
if(usersIn?) then
    if(usersIn.size() >= usersDb.size()) then
        usersDb := usersIn;
```

```
usersOut := usersDb;
```

Possible Reactions:

$$(usersdB)\xrightarrow{(\ usersIn)/(usersOut)} > (usersdB\ *)$$

Task Dependencies:

A1 < A2

Execution Schedule:

A1,A2

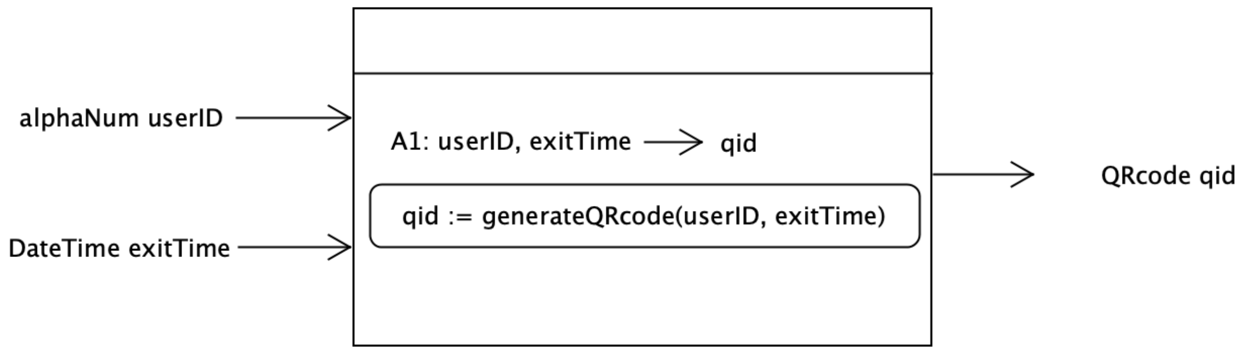
13. User Exit: This SRC takes the userID and the exit time stamp as the input and generates the QR code to be scanned at the exit point to initialize the fare payment process.

I = {userID, exitTime}
O = {qid}
S = {}

Inputs for **User Exit**			
Name	Range of Values	Initial Value	Unit
userId	[0-9][A-Z]*10	A000000000	ID
exitTime	MM:1-12 YY: 2022-2030 HH: 1-24 MM: 1-60 SS: 1-60	None	datetime

Outputs for **User Exit**			
Name	Range of Values	Initial Value	Unit
qId	-	-	barcode

States for **User Exit**			
Name	Range of Values	Initial Value	Unit
-	-	-	-


alphaNum userID
DateTime exitTime
A1: userID, exitTime → qid
qid := generateQRcode(userID, exitTime)
QRcode qid

Possible Reactions:

$$(null)\xrightarrow{(userId, exitTime)/(qId)} > (null)$$

Task Dependencies:

A1

Execution Schedule:

A1

14. System Exit Validator: This SRC scans the QR code of the user at the exit point as the input and outputs the currentTime as the exit time, a boolean flag exitCheck as an exit signal for the Cost Estimator module and the spotID vacated by the user to the Smart Parking Management.

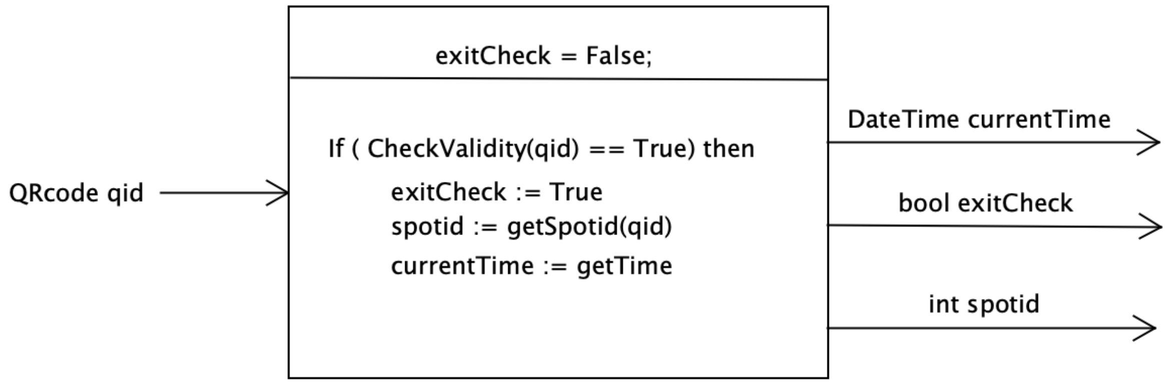
I = {qid}
O = {currentTime, exitCheck, spotid}
S = {}
Init => exitCheck := False

Inputs for **System Exit Validator**

Name	Range of Values	Initial Value	Unit
qid	-	-	barcode

Outputs for **System Exit Validator**			
Name	Range of Values	Initial Value	Unit
currentTime	MM:1-12 YY: 2022-2030 HH: 1-24 MM: 1-60 SS: 1-60	None	datetime
exitCheck	{True, False}	False	bool
spotid	{absent, [0-9][A-Z]*10}	None	ID

States for **System Exit Validator**			
Name	Range of Values	Initial Value	Unit
-	-	-	-



Possible Reactions:

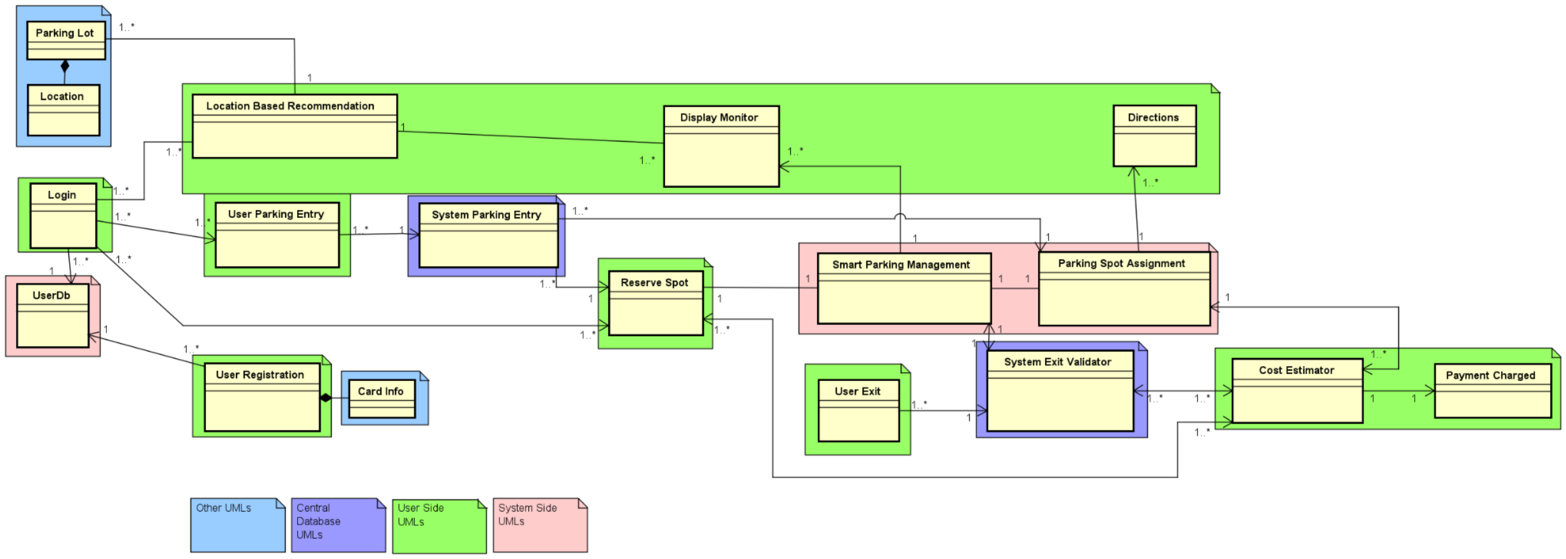
$$(exitCheck)\xrightarrow{(\,qId)/(currentTime, exitCheck, spotId)} > (exitCheck)$$

2.2 UML design specifications with descriptions

It provides UML design specifications including class diagrams with descriptions describing design choices and tradeoffs [2,4].

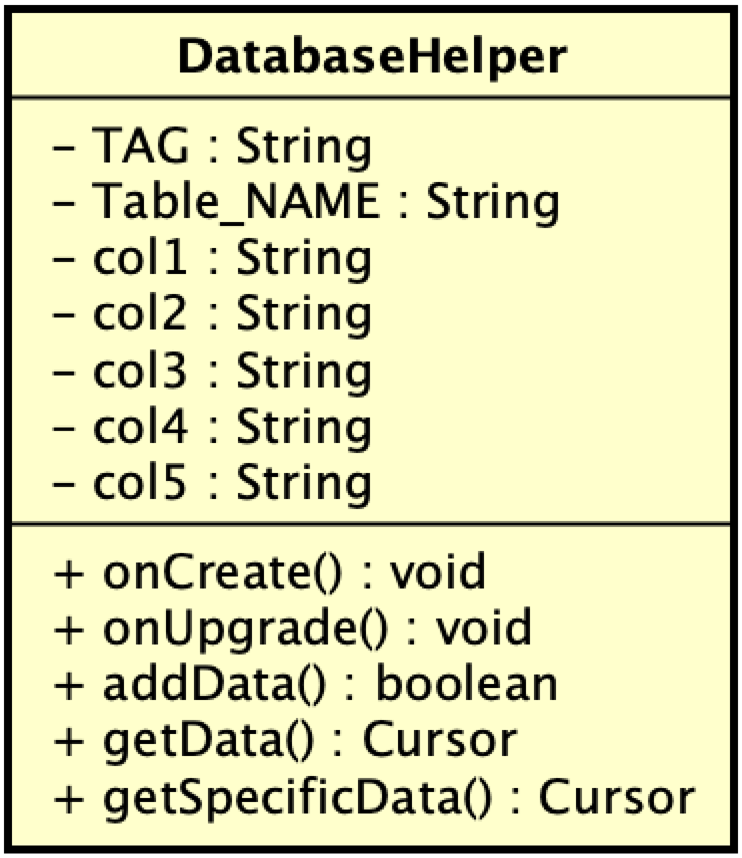
1. UML Abstract System Overview

Below is the overview of all the UML diagrams that we have designed as part of the formal specification of the system. In the class diagram below, we have only included the class name, not the attributes and methods. This is due to the fact that we are showing the relationship between the classes. A detailed description of each class complete with attributes and methods is provided in the next section.
As for the overview diagram. We have divided all the classes into four broad categories. First category is the User Side UML, which is highlighted by the green background. Second category is the system side UMLs denoted by the pink background. Third category is the Central Database UMLs highlighted by a purple background. And lastly the other UMLs which are highlighted by blue background.
Furthermore, these different kinds of UMLs have different kinds of relationships. We have association, composition, one to one relationship, one to many relationship, and many to one relationships. All the relationships are denoted in the overview diagram.



2. Individual UML diagrams with complete details

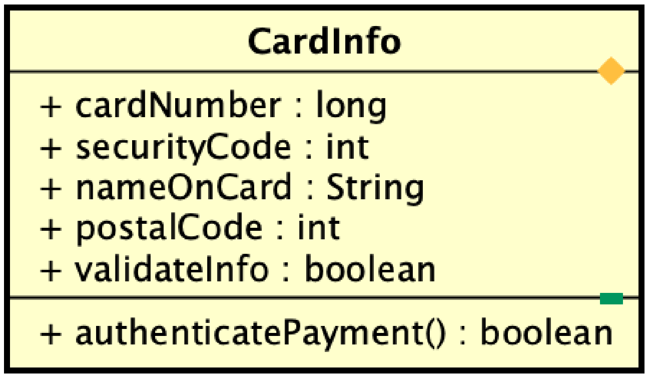
1-



The Database Helper class consists of the Tag, table name, string attributes.
The col1 is the ID of the user
col2 represents the first name of the user
col3 represents the last name of the user
col4 represents the password set by the user
col5 represents the Joined on date of the user

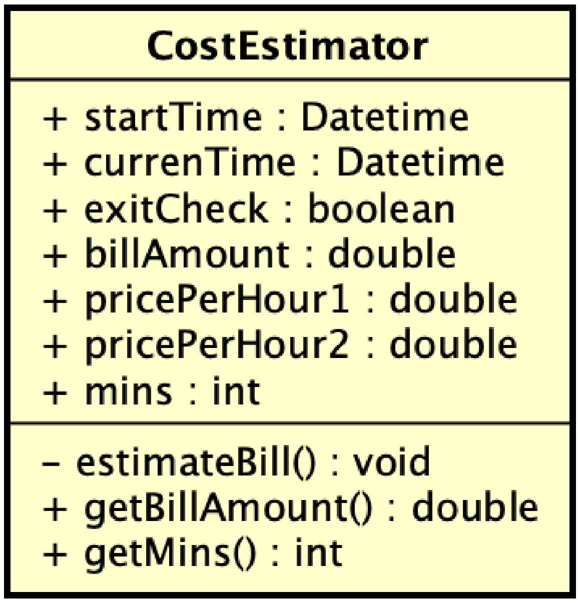
The method onCreate executes an SQL command to generate a table
The onUpgrade method executes an SQL command to drop the table if the table already exists.
The addData adds the columns 1 through 5 to the database table
The getData executes an sql command to retrieve the data from the database table
The getSpecificData method executes an sql command to retrieve the data based on a where condition from the database table.
This is used to authenticate a user at local level on the given mobile device.

2-



The CardInfo class has attributes to store the card number, the 3 digit security code, name of the card holder and the postal code of the area in which the card was issued.
The validateInfo is a boolean attribute to record of the payment has been authenticated.
The method authenticatePayment validates the card information and returns a boolean value.

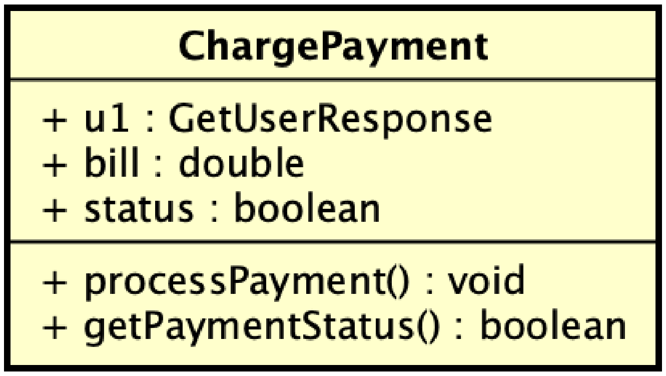
3-



The startTime attributed is used to record the start time of the reservation of the parking spot.
The currenTime stores the current time in that time zone to estimate the cost from start time.
The exitCheck is a boolean value to record of the exit event of the user is validated.
The billAmount stores the due charges of the user at the currenTime.
The pricePerHour1 and pricePerHour2 store the 2 different pricings of the parking structure.
The mins attribute is used to record the time difference in minutes form for sake of calculation.
The method estimateBill calculates the bill amount based of the startTime and currenTime of the user.
The getBillAmount presents the bill amount due to the user.

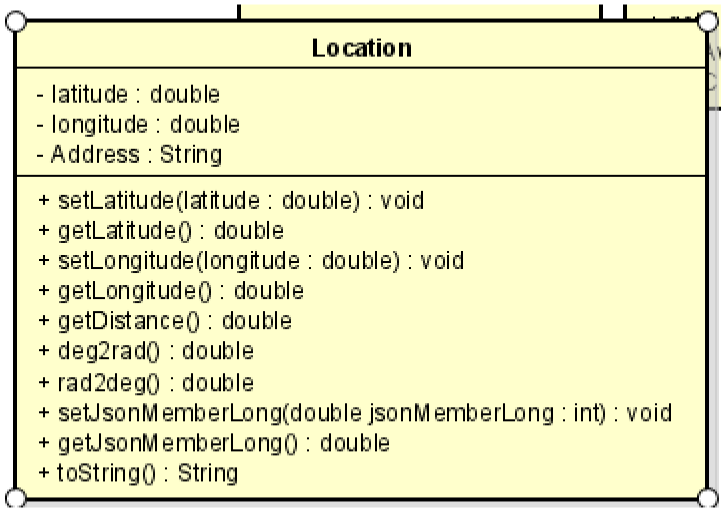
The getMins method presents the rounded of minutes to the next hour.

4-



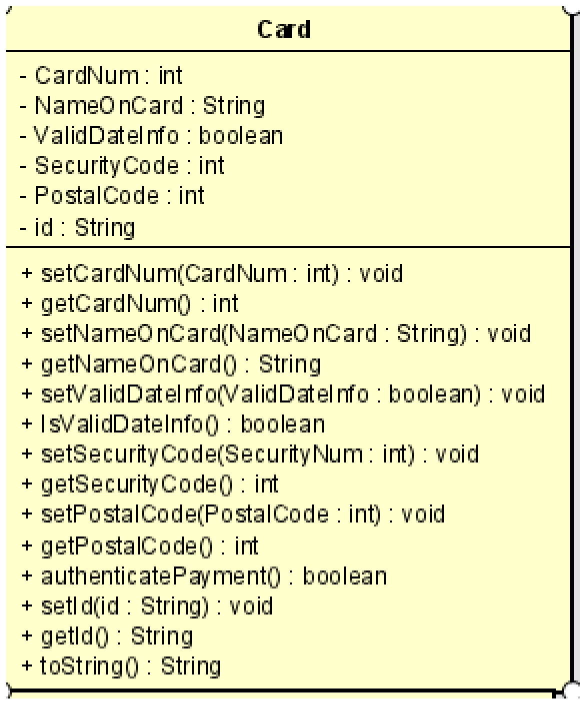
The u1 id a GetUserResponce object corresponding to the user.
The bill attribute stores the bill amount due by the user.
the status attribute hold the status the the payment transaction as a boolean
The processPayment method is a dummy method to demonstrate a payment gateway by getting the user's card information.
The getPaymentStatus returns the status of the transaction based on the payment processing.

5-


Location
- latitude : double
- longitude : double
- Address : String
+ setLatitude(latitude : double) : void
+ getLatitude() : double
+ setLongitude(longitude : double) : void
+ getLongitude() : double
+ getDistance() : double
+ deg2rad() : double
+ rad2deg() : double
+ setJsonMemberLong(double jsonMemberLong : int) : void
+ getJsonMemberLong() : double
+ toString() : String

latitude and longitude represents the location attributes.
Address represents a string of exact address attributes.
There are getters and setters for latitude and longitude.
getDistance will take two location objects and return the distance between them.
deg2rad will convert decimal degrees to radians.
rad2deg will convert radians to decimal degrees.
setJsonMemberLong will take a longitude and set the default longitude.
toString will return all the given information to a string.

6-

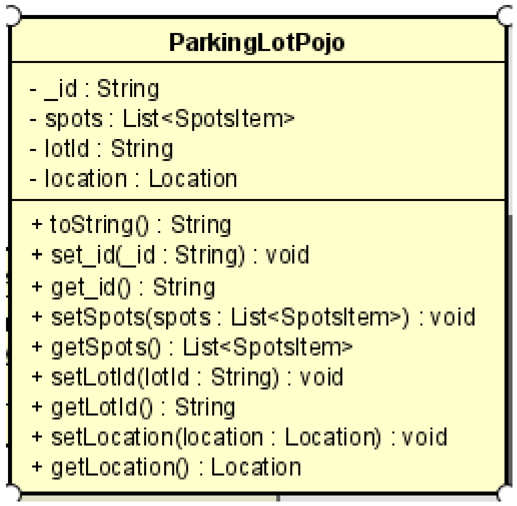

Card

- CardNum : int
- NameOnCard : String
- ValidDateInfo : boolean
- SecurityCode : int
- PostalCode : int
- id : String

```
+ setCardNum(CardNum : int) : void
+ getCardNum() : int
+ setNameOnCard(NameOnCard : String) : void
+ getNameOnCard() : String
+ setValidDateInfo(ValidDateInfo : boolean) : void
+ IsValidDateInfo() : boolean
+ setSecurityCode(SecurityNum : int) : void
+ getSecurityCode() : int
+ setPostalCode(PostalCode : int) : void
+ getPostalCode() : int
+ authenticatePayment() : boolean
+ setId(id : String) : void
+ getId() : String
+ toString() : String
```

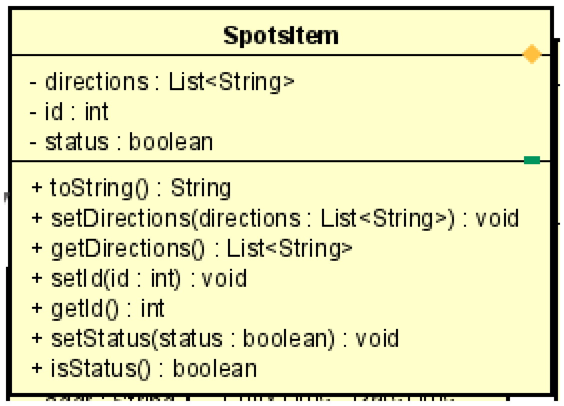
It has attributes: nameOnCard, postalCode, securityCode, validateInfo, id, card number, all represent the card information.
It also has getters and setters for all the attributes.
It has a toString operation that returns all given information to a string.
It has an authenticatePayment operation which returns the validateInfo of the card.

7-


ParkingLotPojo
- _id : String
- spots : List<SpotsItem>
- lotId : String
- location : Location
+ toString() : String
+ set_id(_id : String) : void
+ get_id() : String
+ setSpots(spots : List<SpotsItem>) : void
+ getSpots() : List<SpotsItem>
+ setLotId(lotId : String) : void
+ getLotId() : String
+ setLocation(location : Location) : void
+ getLocation() : Location

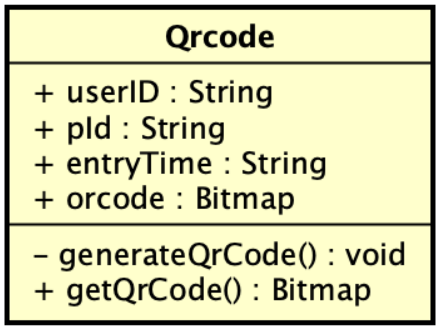
It has attribute _id which represents the parking lot id.
It has attribute spots which store all the spots in the parking lot in a list.
It has an attribute location which represents the location of the parking lot.
It has getters and setters for given attributes.
It has a toString operation which returns given information to a string.

8-


SpotsItem
- directions : List<String>
- id : int
- status : boolean
+ toString() : String
+ setDirections(directions : List<String>) : void
+ getDirections() : List<String>
+ setId(id : int) : void
+ getId() : int
+ setStatus(status : boolean) : void
+ isStatus() : boolean

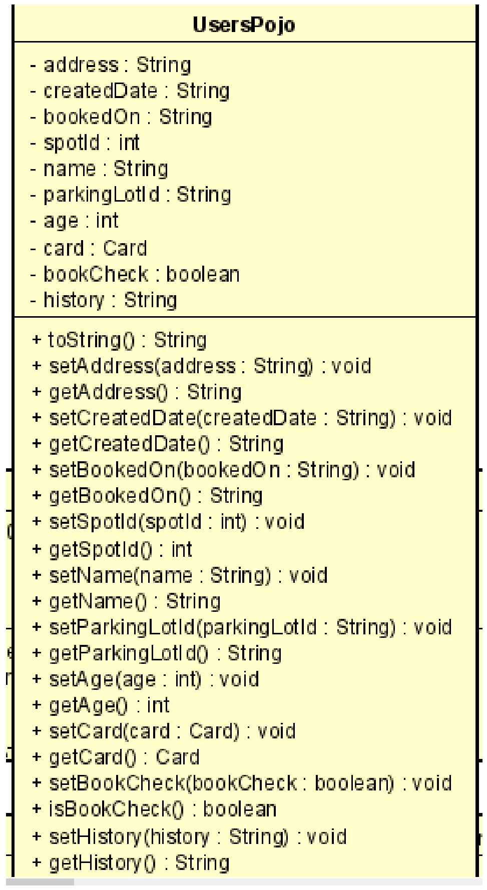
It has an attribute directions which represents the directions information in a String list.
It has an attribute id which represents the id of the spot.
It has an attribute status which represents the current status of the spot, empty or taken.
It has getters and setters for all attributes.
It has a toString operation which returns given information to a string.

9-



It has a userID attribute which is randomly generated for a user.
It has a pid attribute which represents the id for a specified parking lot.
It has an entryTime attribute which records the entry time of the user.
It has a qrcode attribute which stores user qrcode information.
It has generateQrCode operation to generate a new qr code for users.
It has a getQrCode operation to get the qrCode information. .

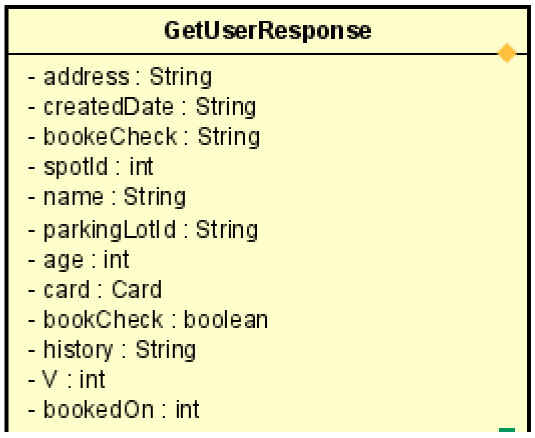
10-


UsersPojo
- address : String
- createdDate : String
- bookedOn : String
- spotId : int
- name : String
- parkingLotId : String
- age : int
- card : Card
- bookCheck : boolean
- history : String
+ toString() : String
+ setAddress(address : String) : void
+ getAddress() : String
+ setCreatedDate(createdDate : String) : void
+ getCreatedDate() : String
+ setBookedOn(bookedOn : String) : void
+ getBookedOn() : String
+ setSpotId(spotId : int) : void
+ getSpotId() : int
+ setName(name : String) : void
+ getName() : String
+ setParkingLotId(parkingLotId : String) : void
+ getParkingLotId() : String
+ setAge(age : int) : void
+ getAge() : int
+ setCard(card : Card) : void
+ getCard() : Card
+ setBookCheck(bookCheck : boolean) : void
+ isBookCheck() : boolean
+ setHistory(history : String) : void
+ getHistory() : String

It has address, createdDate, bookedOn, spotId, name, parkingLotId, age, card, bookCheck, history attributes which are all the information of a registered user.

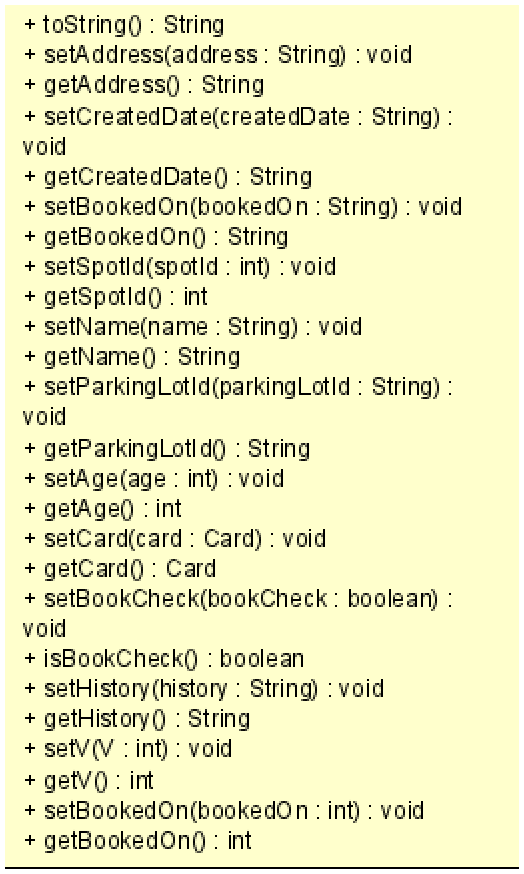
It has all the getters and setters for all the attributes.
It has a toString operation that returns all given attributes to a string.

11-



GetUserResponse

- address : String
- createdDate : String
- bookeCheck : String
- spotId : int
- name : String
- parkingLotId : String
- age : int
- card : Card
- bookCheck : boolean
- history : String
- V : int
- bookedOn : int

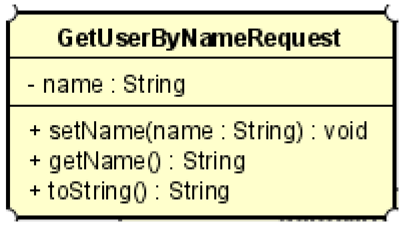


+ toString() : String
+ setAddress(address : String) : void
+ getAddress() : String
+ setCreatedDate(createdDate : String) : void
+ getCreatedDate() : String
+ setBookedOn(bookedOn : String) : void
+ getBookedOn() : String
+ setSpotId(spotId : int) : void
+ getSpotId() : int
+ setName(name : String) : void
+ getName() : String
+ setParkingLotId(parkingLotId : String) : void
+ getParkingLotId() : String
+ setAge(age : int) : void
+ getAge() : int
+ setCard(card : Card) : void
+ getCard() : Card
+ setBookCheck(bookCheck : boolean) : void
+ isBookCheck() : boolean
+ setHistory(history : String) : void
+ getHistory() : String
+ setV(V : int) : void
+ getV() : int
+ setBookedOn(bookedOn : int) : void
+ getBookedOn() : int

It has address, V, createdDate, bookedOn, spotId, name, parkingLotId, age, card, bookCheck, history attributes which represents the user’s response.

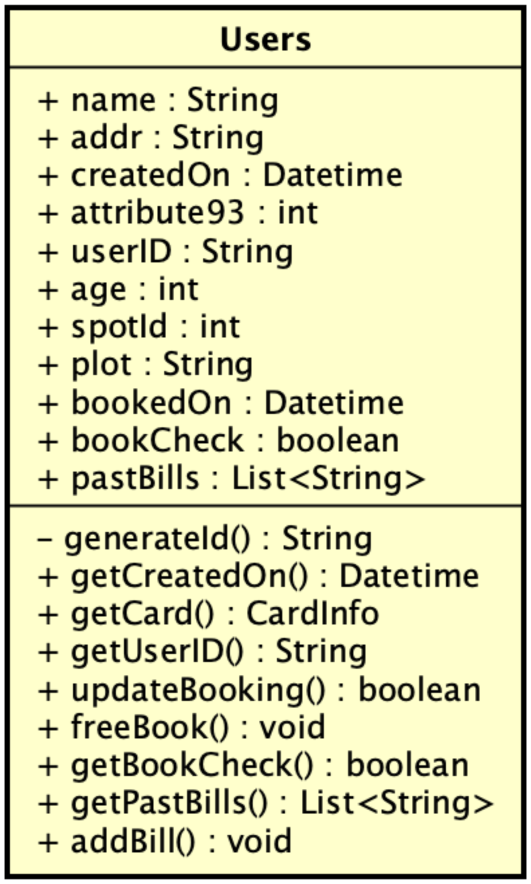
It has all the getters and setters for all the attributes.
It has a toString operation that returns all given attributes to a string.

12-



It has an attribute name that represents the user’s name.
It has getter and setter methods for the name attribute.
It has the toString attribute that returns the information as a string.

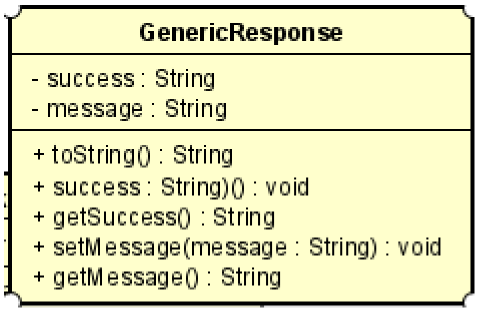
13-



It has attributes name, addr, createdOn, card, userId, age, spotId, plot, bookedOn, bookCheck, pastBills as a given user’s information.
It has getters for createdOn, Card, userId attributes..
It has a generatedId operation that generates an user id randomly.

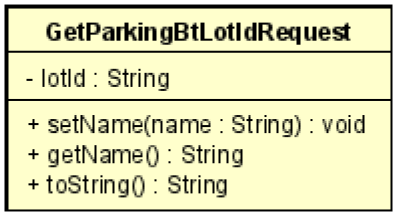
It has an updateBooking operation that updates a user's booking of a given parking lot.
It has a freeBook operation that frees the user's book.
It has a getPastBills operation that returns a list of strings of users' past bills.
It has an addBill operation that adds a bill to the pastBills list.

14-



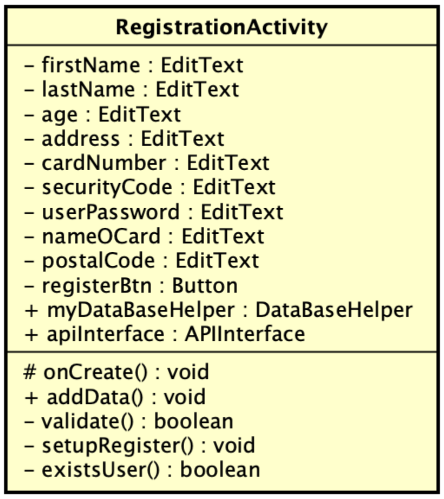
It has the attribute success that represents the status.
It has the attribute message that stores the response message.
It has getters and setters for the attributes.
It has the operation to String which returns given attributes to a string.

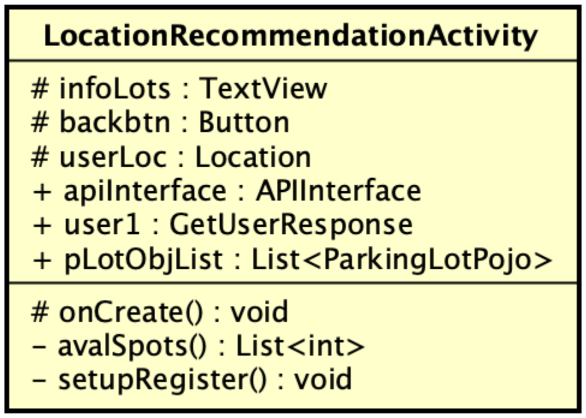
15-

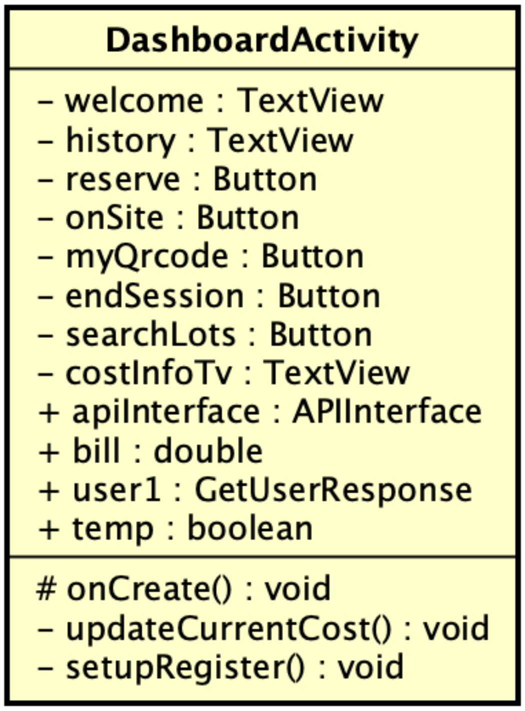


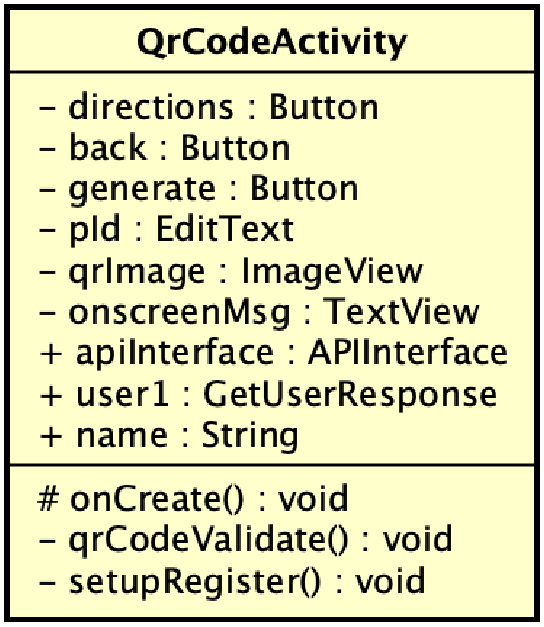
It has an attribute lotId that represents the id of a parking lot.
It has getters and setters of the attribute.
It has a toString operation that returns the given attribute to a string.

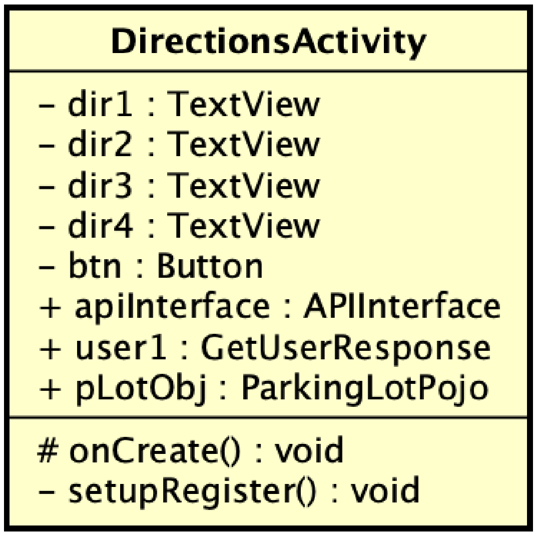
16- User Interface Class UML diagrams:


RegistrationActivity
- firstName : EditText
- lastName : EditText
- age : EditText
- address : EditText
- cardNumber : EditText
- securityCode : EditText
- userPassword : EditText
- nameOCard : EditText
- postalCode : EditText
- registerBtn : Button
+ myDataBaseHelper : DataBaseHelper
+ apiInterface : APIInterface
# onCreate() : void
+ addData() : void
- validate() : boolean
- setupRegister() : void
- existsUser() : boolean


LocationRecommendationActivity
# infoLots : TextView
# backbtn : Button
# userLoc : Location
+ apiInterface : APIInterface
+ user1 : GetUserResponse
+ pLotObjList : List<ParkingLotPojo>
# onCreate() : void
- avalSpots() : List<int>
- setupRegister() : void


DashboardActivity
– welcome : TextView
– history : TextView
– reserve : Button
– onSite : Button
– myQrcode : Button
– endSession : Button
– searchLots : Button
– costInfoTv : TextView
+ apiInterface : APIInterface
+ bill : double
+ user1 : GetUserResponse
+ temp : boolean
# onCreate() : void
– updateCurrentCost() : void
– setupRegister() : void


QrCodeActivity
– directions : Button
– back : Button
– generate : Button
– pId : EditText
– qrImage : ImageView
– onscreenMsg : TextView
+ apiInterface : APIInterface
+ user1 : GetUserResponse
+ name : String
# onCreate() : void
– qrCodeValidate() : void
– setupRegister() : void



3 Implementation

It provides the list of implemented classes, interface, and other classifiers [3]. Each implemented classifier provides its number of lines and size. The list of other files such as input and output files is to be included. A brief description on forward engineering the Astah UML designs to code is to be provided.

After designing all the UML diagrams we proceeded with implementation of the system. We had a good amount of class diagrams at our hand. These diagrams were mostly complete with all the attributes and methods that we accounted for might be needed.

In Android, we started with creating two different applications. One for the user side and other for the system side. For each class diagram that we had created in Astah we proceeded with creating an activity and or helper classes in the Android. We created Java classes for each of the functionality. Given the attributes that we had, we created constructors for each class that let us initialize the attributes with supplied desired values. Additionally, we implemented all the methods described in the UML diagrams. As we proceeded, some new methods were added and some additional attributes were required due to changes we had to make to give our system a higher performance and scalability aspect.

We also made use of Astah features that directly converted the designed UML to corresponding Java class, this saved us a lot of time while writing the codebase. We also made use of POJO in android that converts a JSON response to appropriate class, this was done because we are saving all the information in mongoDb and it was an easy trick to simply write up the structure of datasample we want for users and parking lots and directly get a rudimentary class structure for API classes in android studio.

User Application (aka. Smart Parking App) ****Both the Apps are compatible for almost 69% of the android mobile devices in existence today.**

Development Environment	Class Name	Programming Language	# Code Lines
Android	ChargePayment	Java	34
Android	CostEstimator	Java	49
Android	DataBaseHelper	Java	66
Android	Qrcode	Java	37
Android	DashboardActivity	Java	181
Android	DirectionsActivity	Java	95
Android	ExitActivity	Java	134
Android	LocationRecommendationActivity	Java	95
Android	LoginActivity	Java	86
Android	MainActivity	Java	87
Android	QrCodeActivity	Java	162
Android	RegistrationActivity	Java	141
Android	UsersPojo	Java	148
Android	ParkingPojo	Java	98
Android	APIInterface	Java	48
Android	APIClient	Java	31
Android	Card	Java	110

Android	GetUserByNameRequest	Java	29
Android	GetParkingByLotIdrequest	Java	29
Android	GetUserResponse	Java	159
Android	SpotItems	Java	50
Android	Location	Java	79

Node.js Server

Development Environment	Class/file Name	Programming Language	# Code Lines
VS Code	app.js	Javascript	27
VS Code	card.js	Javascript	17
VS Code	parkingRouter.js	Javascript	67
VS Code	userRouter.js	Javascript	96
VS Code	parkinglot.js	Javascript	20
VS Code	user.js	Javascript	31

System Application (aka. Parking System App)

Development Environment	Class Name	Programming Language	# Code Lines
Android	ValidateEntry	Java	145
Android	AddParking	Java	87
Android	UsersPojo	Java	148
Android	ParkingPojo	Java	98
Android	APIInterface	Java	48
Android	APIClient	Java	31
Android	Card	Java	110
Android	GetUserByNameRequest	Java	29

Android	GetParkingByLotIdrequest	Java	29
Android	GetUserResponse	Java	159
Android	SpotItems	Java	50
Android	MainActivity	Java	68
Android	Location	Java	79
Android	ValidateExit	Java	140

4 Experiments and results

It provides outputs for the normal operation, configuration, and other things consistent with the project topic [5].

For the purpose of this project, we experimented with a different variety of scenarios for the proof of correctness and proof of scalability. Since out system is an end to end application(s) we do provide necessary means to add new users through our registration page and we tested this component heavily with a variety of situations like what if the user does not enter their first name or last name or both, what if user is under the age of 18 or what if the user does not provide correct number of credit/debit card number (i.e, 16 digits) or what if they entered a security code less than or greater than 3 digits or a pin code will values over 5 digits. For all such scenarios our system throws a toast/message on the application platform to ask the user to update a specific field. We have implemented various checks to make sure that each and every user that registers to the system is over 18, has both first and last name and does have the correct number of digits for card information.

Similarly we did checks for each individual activity/screen interface of the applications under different settings like how will the UI look when the user does not have any ongoing books and what buttons/actions can they perform and how things change when the user starts a booking. For example, In the demo video you can see that the user initially starts with no bookings and has access to reserve, on the go, parking lot location recommendation buttons only, this is done to let users have access to various parking lots around them, and the right to request for booking through a dedicated button. Once the user starts a booking, all these actions are now inaccessible to the user and now they are shown a new button called my qrcode that displays their booking code and the live information of the amount they are owed based on the amount of time since their booking. Apart from this they also have a new button called end session to terminate their ongoing session and exit for payment. We heavily tested all these functionalities both for a single user case and multiple parallel user situations.

We also tested our database APIs that returns useful information and also helps update the central db that is shared over the host of users and parking lots. To test our APIs we made use of Postman which is an API platform for developers to create and test their implementations.

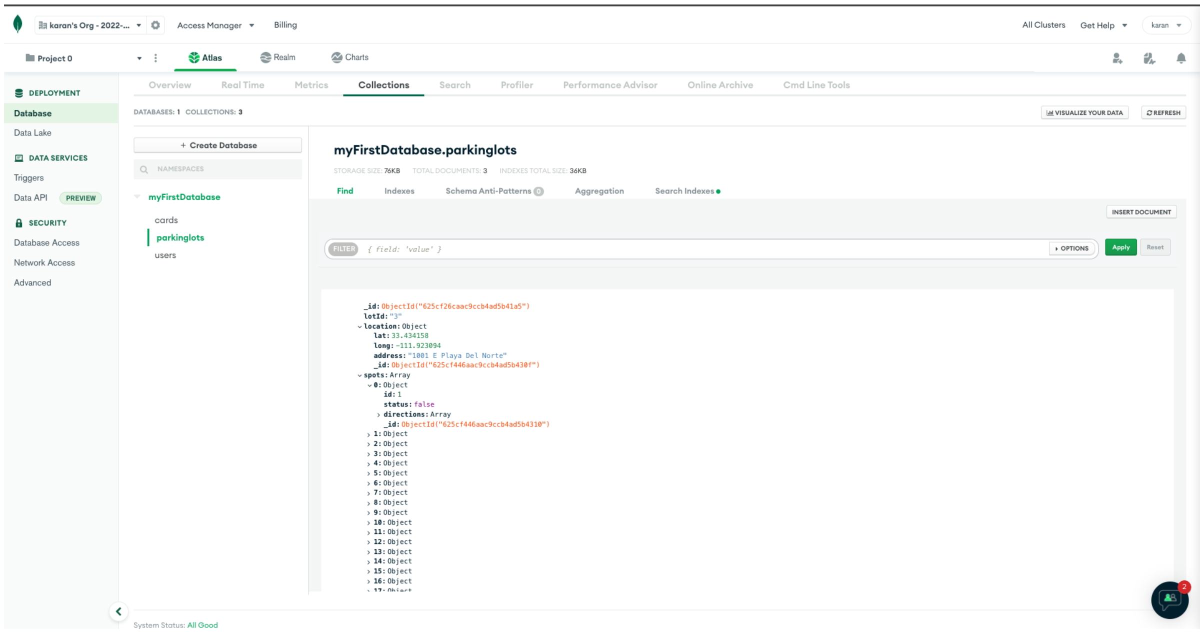
On the system side, we provide various functionalities like validate entry, validate exit, add new parking lots among others. For each such feature we again did extensive testing for correctness and validated all the behavior that it induces to the central database.

Apart from the correctness validation, we also did performance testing for each of the above mentioned component and cross component communication. This was made possible by the datetime library in Java. We noted a timestamp at the start of the operation call and at the finish of it and calculated the time difference between two timestamps to figure out the best possible solution for each aspect. For example, our node server that hosts our mongodb cluster takes around 125 milliseconds to return a query from the database to the mobile application. Here we initially were relying heavily on making use of multiple API calls to and from the database for each action that the user was performing on its session. This approach, even though giving correct results was too slow, as we are doing multiple parallel operations like live cost update on the user platform. To improve the performance of our entire application side system we decided to calculate the API latency and that is how we found that the mongoDB cluster querying commands are not instantaneous. We then decided to make use of the Intent function in android application where we were bundling the user object and passing it to other android activities/screens within a single session and directly making use of updates on the object and at intervals pushing updates to the central server. We also prioritized the calls to the server such that we always make calls when a given user object update can create inconsistencies across different users, for example, when a user books a slot at a parkingLot we treat this as high priority call and always make sure that it gets updated at that moment but other calls like live cost estimation updates happen in intervals and all the calculations are localized to the individual users device. As such calls do not hamper/block the actions for other users.

All in all, we followed every single aspect of software design specifications that includes structure, behavior, complexity and scale of our software solution.

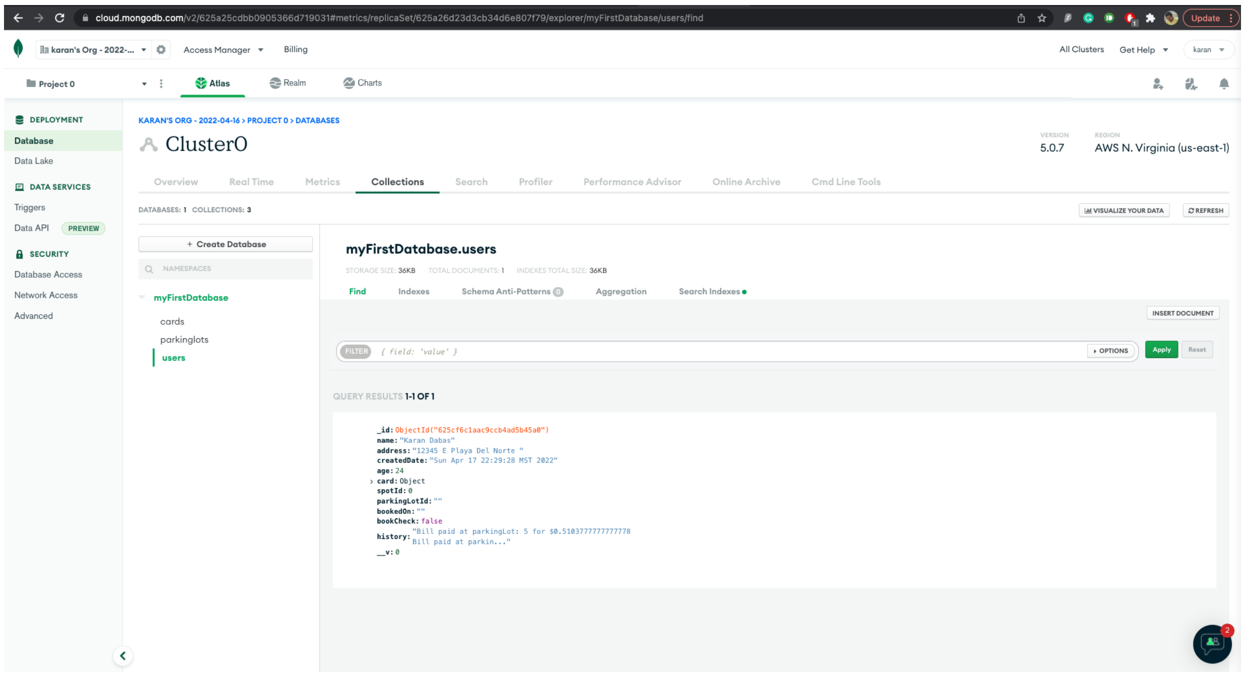
Here is the link to the final demo: https://youtu.be/iHrm-9MA2Jk

Below are a few screenshots of our implementation in android and VS code(node server) and running of a central database on mongodb cluster and postman with get and post requests.



```
{ field: 'value' }
```

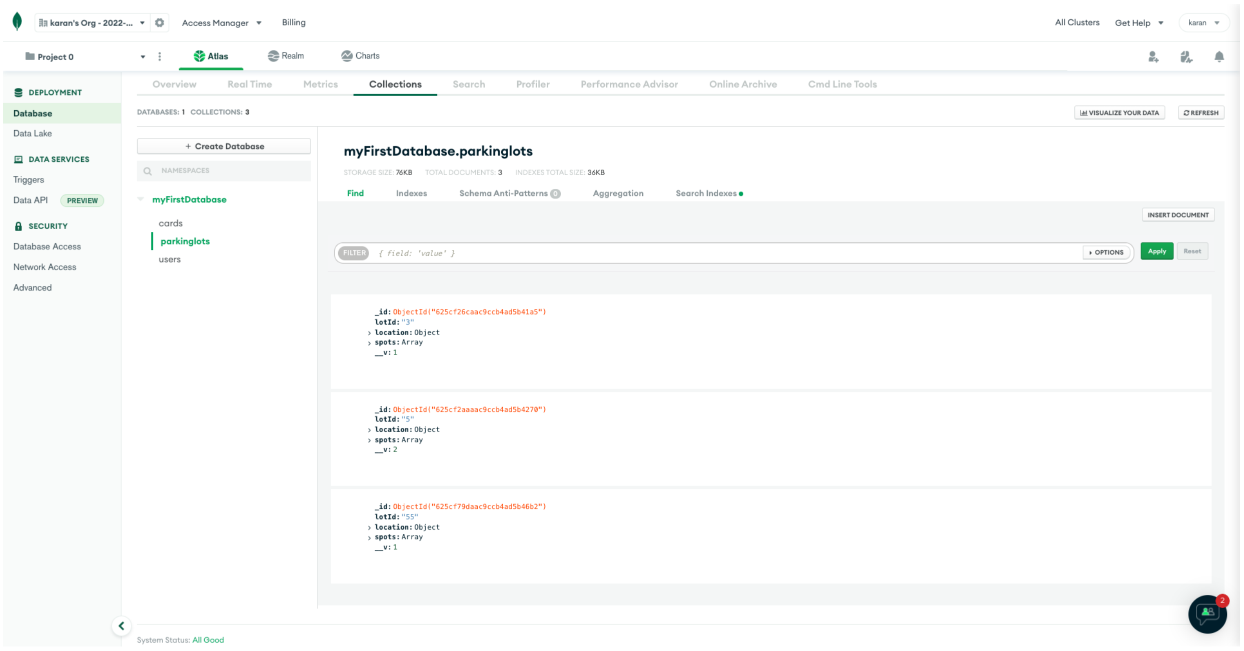
```
   _id: ObjectId("625cf26caac9ccb4ad5b41a5")
   lotId: "3"
 ⌄ location: Object
     lat: 33.434158
     long: -111.923094
     address: "1001 E Playa Del Norte"
     _id: ObjectId("625cf446aac9ccb4ad5b430f")
 ⌄ spots: Array
   ⌄ 0: Object
       id: 1
       status: false
     › directions: Array
       _id: ObjectId("625cf446aac9ccb4ad5b4310")
   › 1: Object
   › 2: Object
   › 3: Object
   › 4: Object
   › 5: Object
   › 6: Object
   › 7: Object
   › 8: Object
   › 9: Object
   › 10: Object
   › 11: Object
   › 12: Object
   › 13: Object
   › 14: Object
   › 15: Object
   › 16: Object
```



```
{ field: 'value' }
```

QUERY RESULTS 1-1 OF 1

```
   _id: ObjectId("625cf6c1aac9ccb4ad5b45a0")
   name: "Karan Dabas"
   address: "12345 E Playa Del Norte "
   createdDate: "Sun Apr 17 22:29:28 MST 2022"
   age: 24
 › card: Object
   spotId: 0
   parkingLotId: ""
   bookedOn: ""
   bookCheck: false
   history: "Bill paid at parkingLot: 5 for $0.5103777777777778
            Bill paid at parkin..."
   __v: 0
```



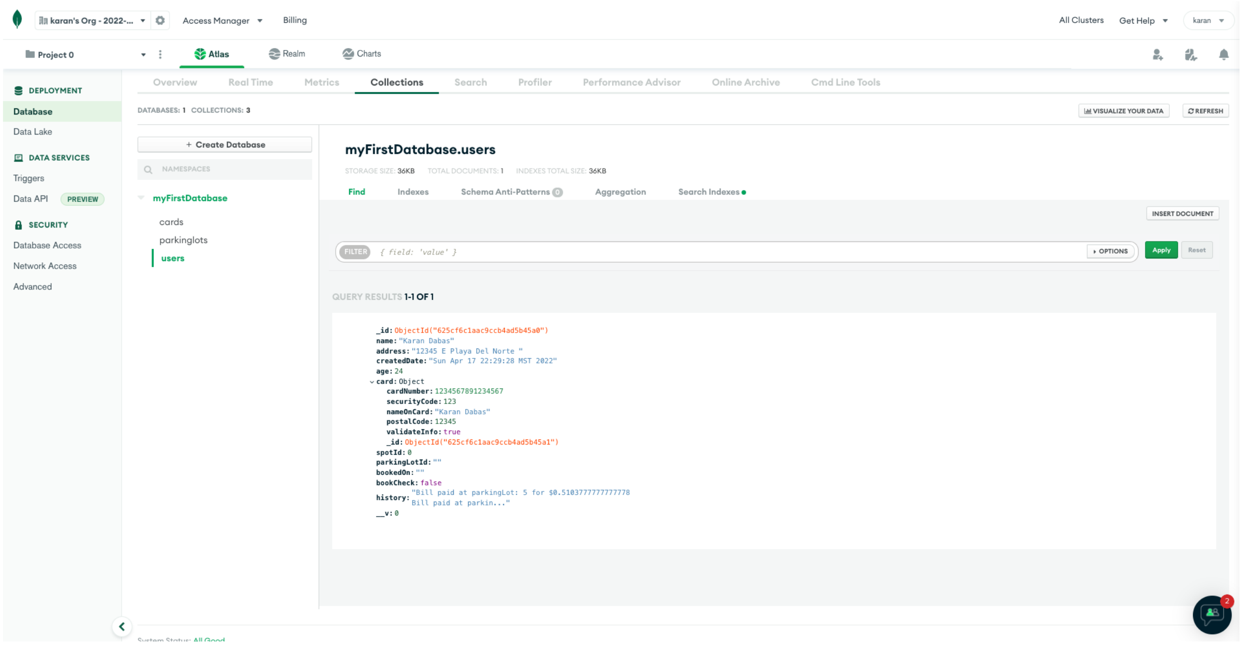
myFirstDatabase.parkinglots

STORAGE SIZE: 76KB TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 36KB

```
  _id: ObjectId("625cf26caac9ccb4ad5b41a5")
  lotId: "3"
> location: Object
> spots: Array
  __v: 1

  _id: ObjectId("625cf2aaaac9ccb4ad5b4270")
  lotId: "5"
> location: Object
> spots: Array
  __v: 2

  _id: ObjectId("625cf79daac9ccb4ad5b46b2")
  lotId: "55"
> location: Object
> spots: Array
  __v: 1
```

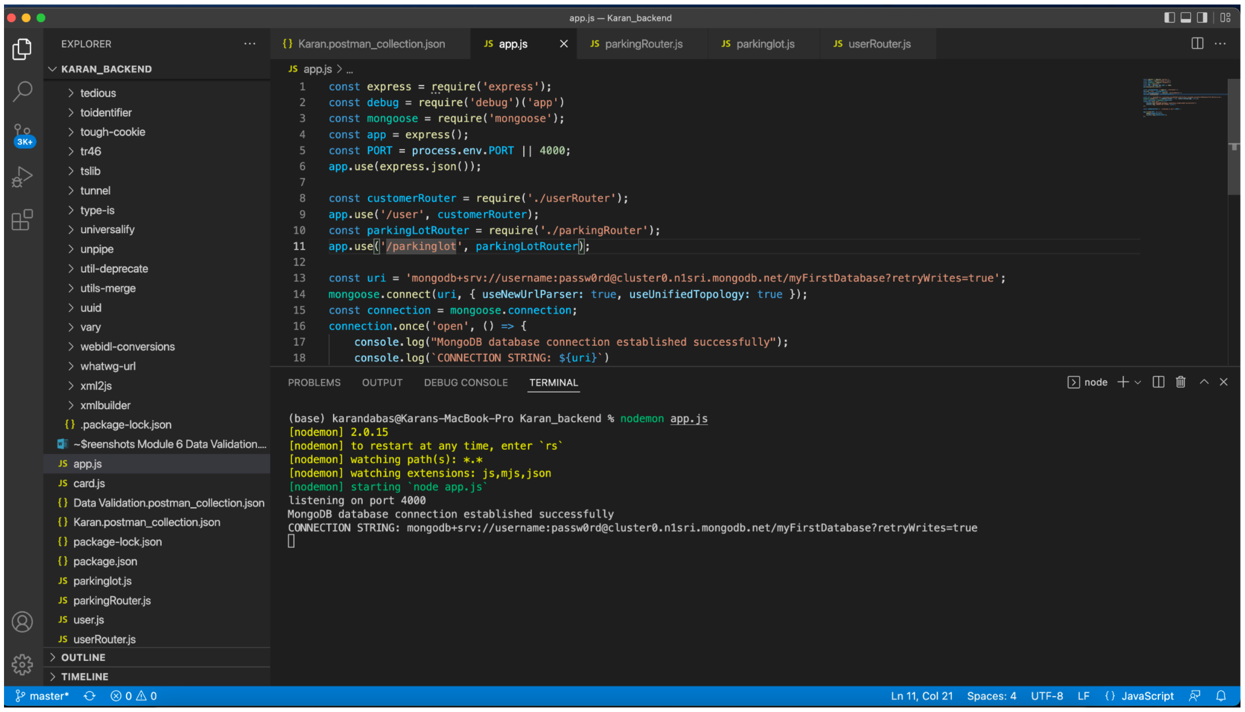


myFirstDatabase.users

STORAGE SIZE: 36KB TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS 1-1 OF 1

```
  _id: ObjectId("625cf6c1aac9ccb4ad5b45a0")
  name: "Karan Dabas"
  address: "12345 E Playa Del Norte "
  createdDate: "Sun Apr 17 22:29:28 MST 2022"
  age: 24
v card: Object
    cardNumber: 1234567891234567
    securityCode: 123
    nameOnCard: "Karan Dabas"
    postalCode: 12345
    validateInfo: true
    _id: ObjectId("625cf6c1aac9ccb4ad5b45a1")
  spotId: 0
  parkingLotId: ""
  bookedOn: ""
  bookCheck: false
  history: "Bill paid at parkingLot: 5 for $0.5103777777777778
           Bill paid at parkin..."
  __v: 0
```

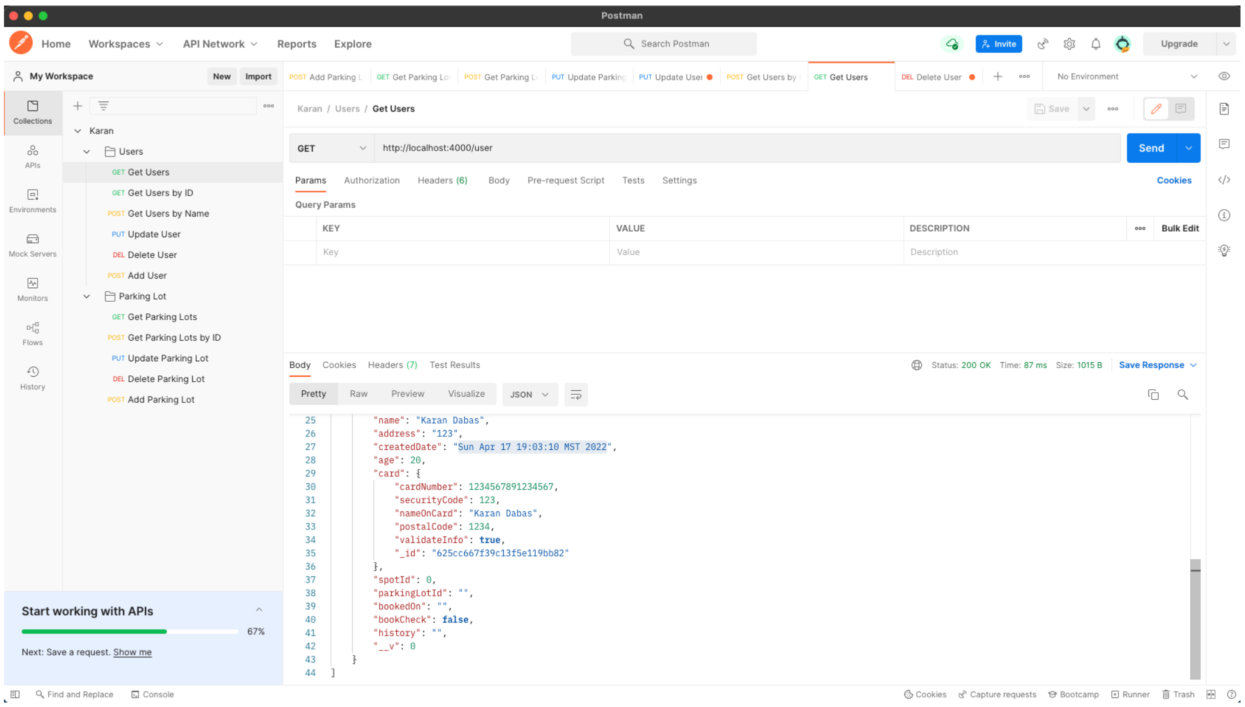


```
const express = require('express');
const debug = require('debug')('app')
const mongoose = require('mongoose');
const app = express();
const PORT = process.env.PORT || 4000;
app.use(express.json());

const customerRouter = require('./userRouter');
app.use('/user', customerRouter);
const parkingLotRouter = require('./parkingRouter');
app.use('/parkinglot', parkingLotRouter);

const uri = 'mongodb+srv://username:passw0rd@cluster0.n1sri.mongodb.net/myFirstDatabase?retryWrites=true';
mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true });
const connection = mongoose.connection;
connection.once('open', () => {
    console.log("MongoDB database connection established successfully");
    console.log(`CONNECTION STRING: ${uri}`)
```

```
(base) karandabas@Karans-MacBook-Pro Karan_backend % nodemon app.js
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
listening on port 4000
MongoDB database connection established successfully
CONNECTION STRING: mongodb+srv://username:passw0rd@cluster0.n1sri.mongodb.net/myFirstDatabase?retryWrites=true
```



GET http://localhost:4000/user

Status: 200 OK Time: 87 ms Size: 1015 B

```
            "name": "Karan Dabas",
            "address": "123",
            "createdDate": "Sun Apr 17 19:03:10 MST 2022",
            "age": 20,
            "card": {
                "cardNumber": 1234567891234567,
                "securityCode": 123,
                "nameOnCard": "Karan Dabas",
                "postalCode": 1234,
                "validateInfo": true,
                "_id": "625cc667f39c13f5e119bb82"
            },
            "spotId": 0,
            "parkingLotId": "",
            "bookedOn": "",
            "bookCheck": false,
            "history": "",
            "__v": 0
        }
]
```

5 Frameworks and SOFTWARE Tools

List of all things (e.g., Astah, data sources, etc.) that are used for designing, coding, testing, experimenting, and evaluating the course project [6,7,8,9,10].

We can divide the entire project into 3 broad categories: Formal design specification, Semi-Formal design specification and implementation & testing. Below are the tools and frameworks used by our team to successfully complete this project.

1. Formal Specification: We made use of a web-based diagramming application called **Lucidchart**, we made use of the free version and since we were a group we decided to save each and every diagram as a file to a shared **Google drive** folder so that we can directly access them at any point to make updates when necessary. This includes all the SRCs and overview diagrams for both simple and detailed structures.

2. Semi-Formal Specifications: We used **Astah** to design the UML abstract system level overview and each individual class diagram for our complete system.

3. Implementation & Testing: We designed and implemented two separate **android** mobile applications using **Java** programming language in **Android Studio**. Both of these applications support API infrastructure that we incorporated within the systems to make calls to the running database server through an in-house **Node server** that implements a set of **Javascript** APIs for Users and ParkingLots that hosts a **Mongodb cluster**. For testing purposes, we made use of an **Android emulator** for testing the correctness of our application during preliminary implementations and later started using **two android mobile phones** under developer mode. For testing the correctness of our APIs we made use of a popular platform called **Postman**. On the database side, we used a NoSQL program framework called **MongoDb** as a data platform. The data samples are in **JSON** format as these samples are stored on MongoDB as documents. The main advantage of JSON is that we can make quick queries on the structured document to access each individual attribute of an object and return the interested matches with longer than 100 milliseconds.

   Pictures of our Project with all these can be seen in Experiments and Results sections above.

6 Conclusions

The conclusion should include lessons learned, recommendations, and experiences you have had in this course. This section should also include self-evaluation for each team member as well as the team as a whole.

Overall Conclusion*:* As our system has a variety of moving parts both within the single application and across different applications(user-side, system-side and node server) we looked at the interoperability (i.e. How do things work with each other?) and found that we did not account for some of the bottlenecks that we faced in the implementation stage which we never imagined during the formal and semi-formal specifications. We also extensively evaluated our system to capture the overall system performance. The factor of performance is very crucial to what we aimed to build, a real end to end system. We faced the issue of latency, the time it takes to get results from the database server to the application. Even though we started off with SRC specifications and built our UMLs on top of them we never thought that getting data objects in time will be something that we have to worry about. This is where we truly saw the difference between logical and real-time stages.

We did component division into three broad categories: system side, user side and central db. In order to place each of the individual functionalities/subsystems we considered the level of communication needed among components to put them under specific categories and later into corresponding applications. We used the idea of intra and inter communication to clear the level of separation among them and help us make a more robust and resilient system. For example, from the communication level overview diagram in SRC sections you can see that most of the components that heavily rely on user objects are placed together into the user side application. This is mainly done to support high communication level between these components as a user switches among different modes/actions in a single active user session on the application. We made use of smart intent packaging to share input and outputs for these components, which are spread across different screens. In between them we have few components that need to relay information to and from the central database but this level of communications is not that intensively used and that is why are kept separate. Additionally, we wanted to make our system scalable and to make this possible we had to come up with a solution that can concurrently handle multiple users and parking lot structures. So we made an informed decision on further dividing the user side from system side. As now we can have multiple users and multiple parking lots and all of them can communicate at different levels with each other and to the central database. This is where we make use of the reuse feature, as every user will have the same structure underneath but have different information stored, we can simply use the entire user side interface and put that as whole on an application that supports such a feature. The use of android applications can help us do exactly that, similarly we created a system side application that manages entry and exit validations along with adding new parking lots.

Overall this was a really informative project, we came across different stages of software design and development of our final product with the highest level of efficiency. This is something that none of us had any experience with, all of us being graduate students we normally used to read the problem and start coding and iteratively update the code logic multiple times to get to the desirable solution. But this course taught us the right way, and techniques and stages developed as part of this course made our entire course project more efficient and logically strong. We did not waste any time in redoing things again and again as we completed one stage. We made sure that everything was perfect and logically sound before moving forward. Such solutions helped us mitigate the cascaded error effect and we only end up making changes to the current stage.

Self Evaluation (by Karan Dabas): Being the team lead for the course project, I started off with itemizing each individual aspect of the project and going over it with fellow team members to finalize the entire workings of the system at a preliminary stage. After this, I proposed to divide the individual components such as cost estimation, user entry/exit, smart spot management, etc… randomly among the entire team. I used a worker-master approach where every assigned work like SRCs, UML or part of implementation that one does have a validator/master to crosscheck the work and provide timely feedback so as to minimize the overall errors and prevent it from being cascaded down to lower levels. Most of the work split and assignment can be seen in the appendix section.

Being a graduate student with no prior experience in industry, I wanted to get into the design aspect of the software development stage and in this course I first hand learned the formal specification using SRCs and ARCs that gives a more concrete mathematical notion to the entire system. Going with such an approach does in fact help mitigate major issues at the preliminary stage, this is way before we start implementing and testing the logic. I also learned the importance of UML as a semi-formal specification and how I can translate the formal specification into it. The idea of starting with formal language gives one the clear tools to describe the relations among input, output and internal variables of a system and include the relationships among different tasks or different RC components. And building on these concepts how we truly implement an entire system in the real world using different programming languages, frameworks and tools to test it.

I feel proud to be part of this team, I can happily say that each and every team member worked equally and even at difficult times where one might find themselves stuck at a part others took initiative to help them and lead them to a quality solution.

Self Evaluation (by Siddharth Kale): As part of the team, we came up with the idea of dividing different parts of the project among ourselves. As per the requirement we started on working on writing formal and semi formal specifications for the system. We first came up with a system overview and had discussions on how the overall system would work and interact. I came up with the idea of how this proposed system would be formulated into a well designed Cyber Physical System. Once we were satisfied that the proposed system is a CPS, we went ahead with writing formal specifications.

Writing formal specifications was something new for me. As I have very little industry experience in designing software, I was not aware of how the designing process works. Learning from the lectures, books, and especially homework, I started working on creating SRCs for different components. I got to learn how to design Input, output for a component. How to decide upon state variables, the initialization of those variables, and overall how the system will have the interactions in form of the reactions.

Proceeding onto the UML specification, I got to learn how these formal specifications are converted into a UML diagram. I got to design before even writing a single line of code for various components. I had to decide upon the methods, attributes, and variables. Overall, creating UML diagrams in Astah was a very different experience in a good way. I was in awe that writing out all these specifications really helped streamline the coding process. It was much easier to decide upon the classes, and their interaction. As each interaction was already decided and everything was specified down to even all possible ranges of each variable, it was much easier to implement the design.

Lastly I would like to say that I am glad that I got to learn about how to properly design a piece of software. How to ensure that the system will not fail under any circumstances, and how the system will perform in real time, with concurrent requests that the system has to handle, that too while in a feedback loop with the real world. Designing and implementing a CPS system helped me learn the overall process of designing a system. I really enjoyed designing the system, and in the future while working in an industry this is something I will look forward to.

Self Evaluation (By Yinghua Li): Being a team member of the course project. We split the tasks to different parts taken by four teammates and we adapted a worker-master working philosophy. So each of the team members can work on their parts and check other team members' work to guarantee the accuracy. We discussed the requirements and workload. We split the tasks to design the SRC diagrams and UML diagrams, and analyzed the significance of designing a CPS project. Then we split our work to do the code implementation for the project.

This is my first time learning formal design of a software. I learned formal representation design such as SRCs and ARCs to model the entire flow of the software. Then the semi-formal design like UML diagrams to specify the requirements and details of each class of the software. The process gives me a clear understanding of how the formal software development process should be and how we should apply this knowledge into real world applications.

I certainly learned a lot working as a team for this course project. We clearly understood we should implement and move forward in every process and we collaboratively made our design throughout the project. It's a valuable experience for me to solidify my team working skills and software design skills.

Self Evaluation (By Pranav Toggi): Being part of this Software Design team while taking on this project has been a very rewarding experience. All team members contributed to my learning experience equally. I began working on SRCs for various components after learning from lectures, literature, and especially homework. I learned how to create input and output for a component. Having never written formal specifications before, I had no idea how the software design process worked because I had no prior knowledge in the field.

I also learned how to use UML as a semi-formal specification and how to convert a formal specification into it. Starting using formal language provides unambiguous tools for describing the links between a system's input, output, and internal variables, as well as relationships between distinct tasks or RC components. And, building on these ideas, how do we test a whole system in the actual world utilizing various programming languages, frameworks, and tools. I wanted to go into the design component of the software development stage as a Master's student, and in this course I first learnt the formal specification using SRCs and ARCs, which gives the entire system a more concrete mathematical notion.

The team building I witnessed throughout the project was very fruitful and the lessons I learned along the way will always stay with me throughout my professional career.

References

Additional references should be included as needed.

[1] R. Alur, (2015), Principles of Cyber-Physical Systems, MIT Press.
[2] G. Booch, et al., (2007), Object Oriented Analysis and Design (OOAD), 3rd Ed., Addison Wesley.
[3] Java Platform, Standard Edition (Java SE), (2019), https://www.oracle.com/java/technologies/java-se.html.
[4] OMG 2012. "Unified Modeling Language version 2.5.1". https://www.omg.org/spec/UML/2.5.1/.
[5] H.S. Sarjoughian, (2020), Software Design Course Project, CSE 564: Software Design (2019 Spring) 2019Spring-T-CSE564-32043 (Blackboard).
[6] Postman Javascript reference for designing, testing APIs, https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/
[7] Setting up node server with javascript, https://nodejs.org/en/docs/guides/getting-started-guide/
[8] MongoDb application data platform setup, https://www.mongodb.com/
[9] QrCode generator API for android application, https://github.com/kenglxn/QRGen
[10] Camera module setup to capture, recognize and decode Qrcode image, https://www.dynamsoft.com/codepool/android-qr-code-scanner.html

A Appendices

Work Division

SRC work spread		
List of Components	Description	Names(Validator,Worker)
User Registration	A registration src that takes in name, age, address, time and card info and outputs an instance of database object that is written to the central DB. The user is registered only if they are 18 or above.	Karan,Yi

Location based recommendation	It takes two inputs: users destination location and our list of parking structure locations and returns a list of K options of parking structure that are near the user destination.	Karan,Yi
Parking Entry(User)	The component is a combinational src that only takes two inputs: a userId and current time. The output is the unique QRcode that a user can use to enter the parking structure.	Karan,Yi
Parking Entry(System)	This is a system side SRC that is also combinational in nature and takes two inputs: a unique user QRcode and a collection of all user information on the platform and returns true if the QRcode is valid else return false. This src is used to authenticate a given user during entry to the parking structure.	Karan,Yi
Spot Assignment	A component used to assign a parking spot in a given parking structure to a user if available, else assign a default code for spot that represents the case when no spots are available and still allow the user to enter the parking structure and direct the user to the nearest exit.	Karan,Siddharth
Availability Display	An src designed to showcase the number of available parking spots in the parking structure at all times. This takes only one input, parking structure id and outputs an integer number.	Karan, Siddharth
Directions	SRC will take the output of “Spot Assignment” as input and show the direction to the specific spot on the app.	Yi,Siddharth
Cost Estimator	Cost will be take Leaving time and Starting time of users parking and time “cost per hour” and return a specific amount	Yi,Pranav

Payment charged	System will take the output of "Cost Estimator" as input and make a charge request to users bank account for the fare amount and return a boolean type.	Yi,Pranav
Reserve Spot	SRC will take input of location, date, from time to time, available spots info, and qrcode id. It will assign a spot if any spot for the location is available for the given time. Output will be spot id, time, userid, amount, and book. All of the output are of typeevent except the last one will be boolean	Siddharth,Pranav
Smart Spot Management	This SRC acts as a central management of the parking spots database and acts as a hub for various SRCs. This SRC outputs the number of available parking spots, number of already taken parking spots, status, parkinglot location,and spotId. This SRC changes the status of a parking spot based on if a user is currently being assigned the parking spot or if the user is vacating the parking spot. Based on this information spotId is generated which also is of the type event. Similarly for inputs, if a userId is available, then only a particular task will be performed where a parking spot is assigned unavailable from the particular parking lot.	Siddharth,Karan
Parking Exit(User)	The component is a combinational src that takes the userID and the exit timestamp as the input and generates the QR code to be scanned at the exit point to initialize the fare payment process.	Pranav, Karan
Parking Exit(System)	This SRC scans the QR code of the user at the exit point as the input and outputs the currentTime as the exit time, a boolean flag exitCheck	Pranav, Karan

	as an exit signal for the Cost Estimator module and the spotID vacated by the user to the Smart Parking Management.	

UML work spread	
List of Components	Names(Validator,Worker)
User Registration	Karan,Yi
Location based recommendation	Karan,Yi
Parking Entry(User)	Karan,Yi
Parking Entry(System)	Karan,Yi
Spot Assignment	Karan,Siddharth
Availability Display	Karan, Siddharth
Directions	Yi,Siddharth
Cost Estimator	Yi,Pranav
Payment charged	Yi,Pranav
Reserve Spot	Siddharth,Pranav
Smart Spot Management	Siddharth,Karan
Parking Exit(User)	Pranav, Karan
Parking Exit(System)	Pranav, Karan

Implementation work spread	
List of Components	Names(Validator,Worker)
User Application	Karan, Siddharth

System Application	Yi, Pranav
API Backend(Node server)	Karan
Mongodb Cluster	Siddharth, Yi
Postman setup	Pranav, Yi
Performance testing	Karan, Siddharth
Correctness testing	Karan, Pranav