GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

Project Proposal – Group 12

GROUP MEMBERS

200104004093 MEHMET METE ŞAMLIOĞLU

1901042678 ERVA AKSU

1901042656 EREN ÇAKAR

1801042620 BARIŞ YURDAKUL

1901042633 MUSTAFA BERKAY BAYGUT

200104004014 OZAN ARMAĞAN

215008003084 OĞUZHAN KÖSE

1801042094 ÖMER FARUK AKDUMAN

161044121 MD SARWAR HOSSAIN

PROBLEM DEFINITION

Bridge crossings constitute an important part of highway transportation. In fact, more than half a million cars pass through some bridges daily. Therefore, the access system on bridges must be complete and error-free. However, the excess of vehicle passage, the security risks arising from the excessive speed of the vehicles, the deficiencies in the vehicle tracking system are some of the problems of the bridge automation system.

Our goal is to ensure the passage of vehicles in the most efficient way and to solve the above-mentioned problems in the most rational way.

In the bridge automation system, there are users, ticket controllers, officer, and admin. Each user has some characteristics: a balance, vehicles etc. User’s vehicles also have some characteristics: owner, in blacklist (can be stolen) etc. We aim to increase the security and quality of access by controlling this information. We have manual passes for our unregistered vehicles that are not in our database. Ticket controllers control these passes. We also have a speed control mechanism to prevent excessive speed. This system provides a deterrent by notifying the police about speeding violations according to the vehicle's entry time and exit time. By using this system, we aim to provide an uninterrupted, secure bridge crossing experience where waiting is minimized.

USERS OF THE SYSTEM

Administrator manages the system by

+ adding, removing, and updating by vehicle, officer, ticket controller

+ can track passing summary

+ can display total revenue

User use the automation system by

+ adding, removing and updating by vehicle

+ can display own balance

+ can add money to own account

+ show own penalties

+ pay own penalties

Toll Clerk check manual passes by

+ controlling manual passes check in-out

+ can query passing summary

+ can query total revenue

Officer can

+ send penalties and warnings

+ remove penalties

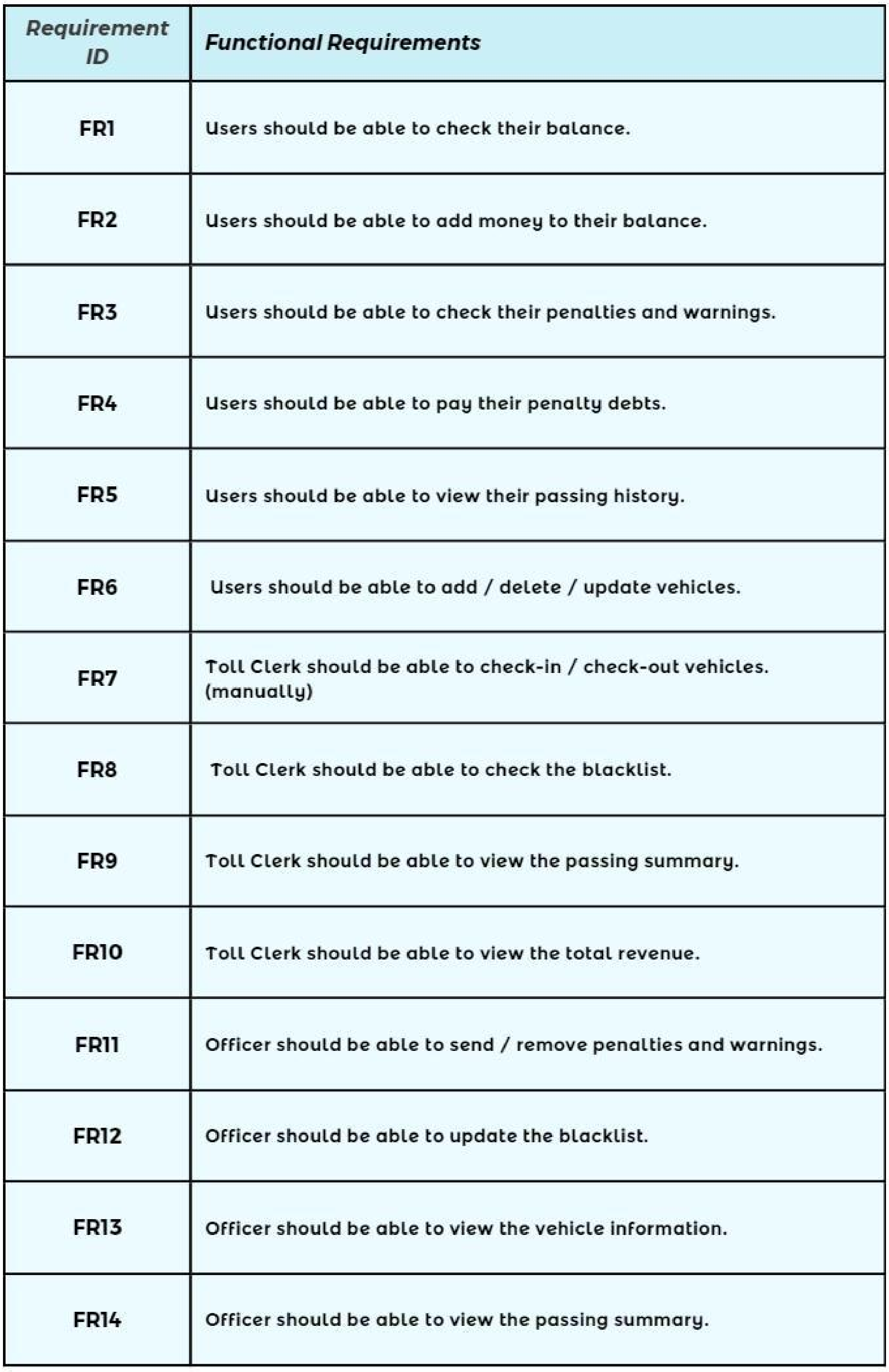
+ show passing summary

+ edit blacklist

+ display vehicle information

REQUIREMENTS IN DETAILS

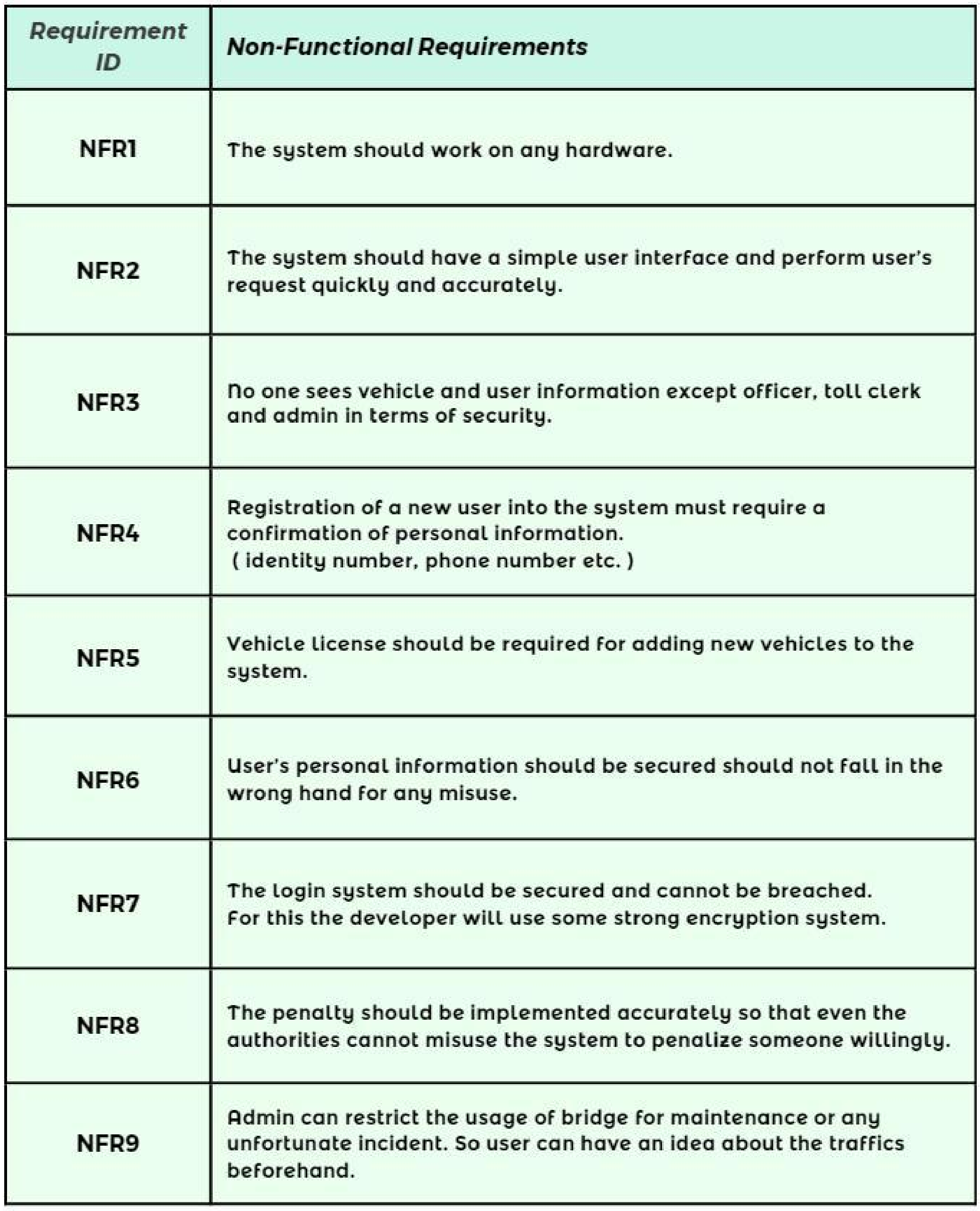
FUNCTIONAL REQUIREMENTS:



Requirement ID	*Functional Requirements*
FR1	Users should be able to check their balance.
FR2	Users should be able to add money to their balance.
FR3	Users should be able to check their penalties and warnings.
FR4	Users should be able to pay their penalty debts.
FR5	Users should be able to view their passing history.
FR6	Users should be able to add / delete / update vehicles.
FR7	Toll Clerk should be able to check-in / check-out vehicles. (manually)
FR8	Toll Clerk should be able to check the blacklist.
FR9	Toll Clerk should be able to view the passing summary.
FR10	Toll Clerk should be able to view the total revenue.
FR11	Officer should be able to send / remove penalties and warnings.
FR12	Officer should be able to update the blacklist.
FR13	Officer should be able to view the vehicle information.
FR14	Officer should be able to view the passing summary.

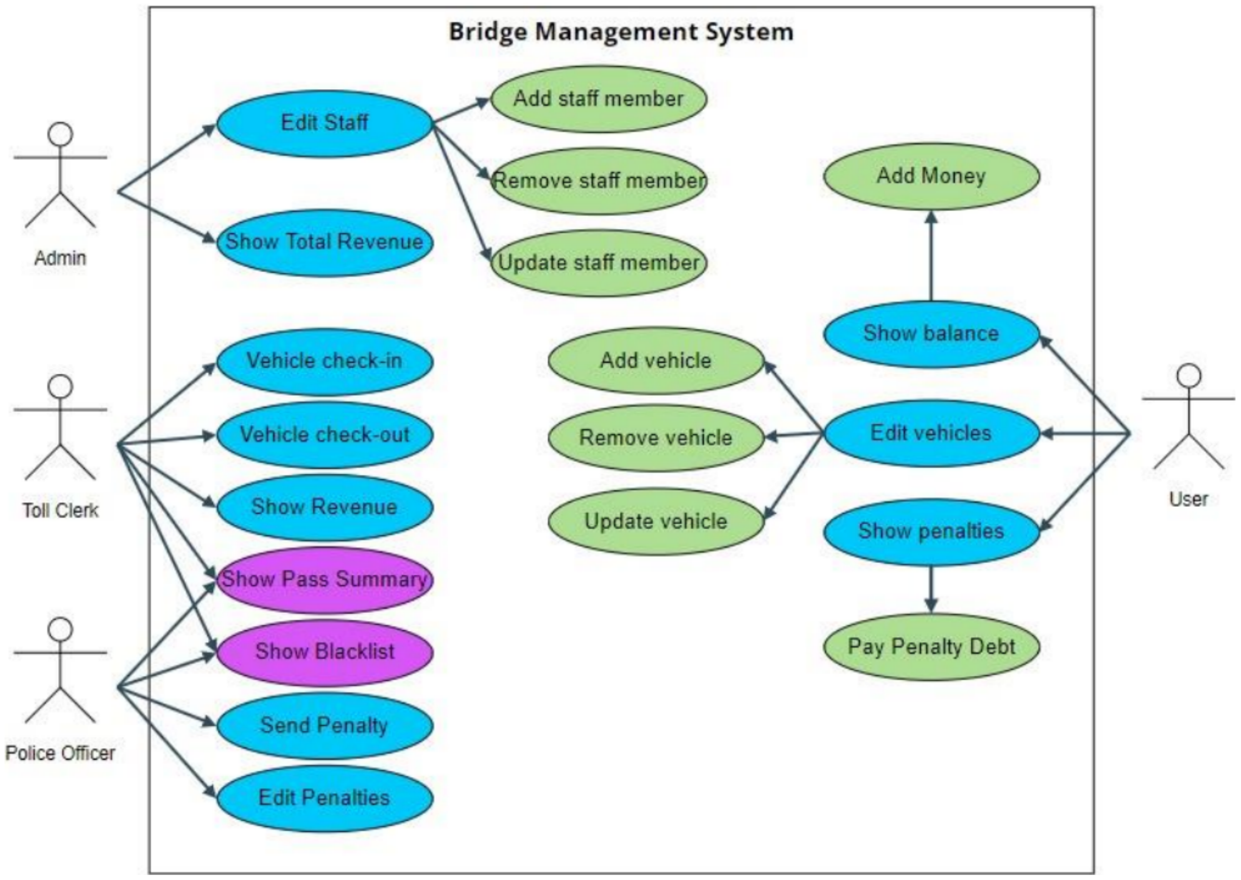
FR15	Admin should be able to add / remove / update users.
FR16	Admin should be able to add / remove / update toll clerks.
FR17	Admin should be able to add / remove / update officers.
FR18	Admin should be able to view the passing summary and total revenue.
FR19	All users in the system should be able to log in/log out to/from the system.
FR20	All users in the system should be able to update their password.

NON-FUNCTIONAL REQUIREMENTS :

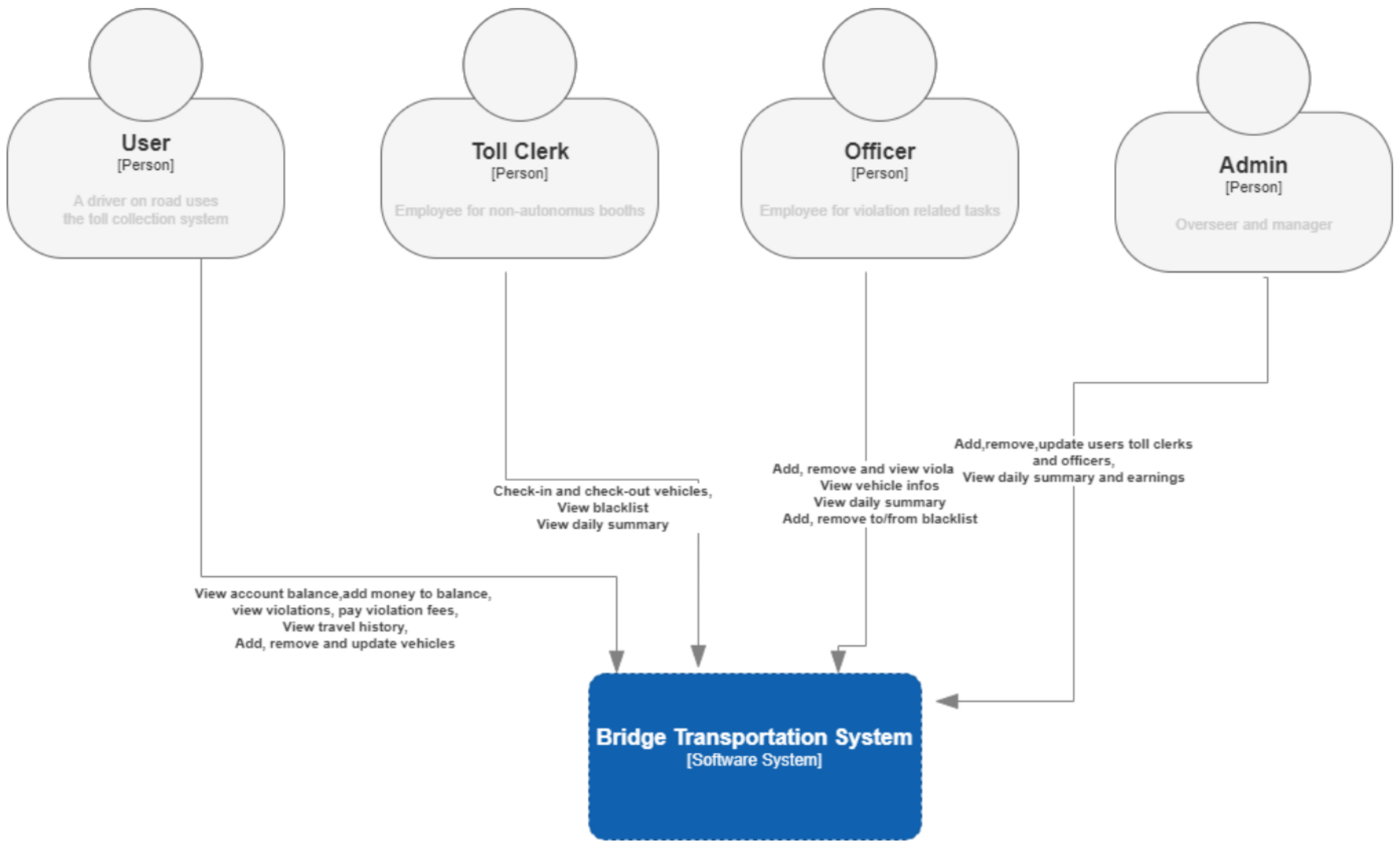


Requirement ID	***Non-Functional Requirements***
NFR1	The system should work on any hardware.
NFR2	The system should have a simple user interface and perform user's request quickly and accurately.
NFR3	No one sees vehicle and user information except officer, toll clerk and admin in terms of security.
NFR4	Registration of a new user into the system must require a confirmation of personal information. (identity number, phone number etc.)
NFR5	Vehicle license should be required for adding new vehicles to the system.
NFR6	User's personal information should be secured should not fall in the wrong hand for any misuse.
NFR7	The login system should be secured and cannot be breached. For this the developer will use some strong encryption system.
NFR8	The penalty should be implemented accurately so that even the authorities cannot misuse the system to penalize someone willingly.
NFR9	Admin can restrict the usage of bridge for maintenance or any unfortunate incident. So user can have an idea about the traffics beforehand.

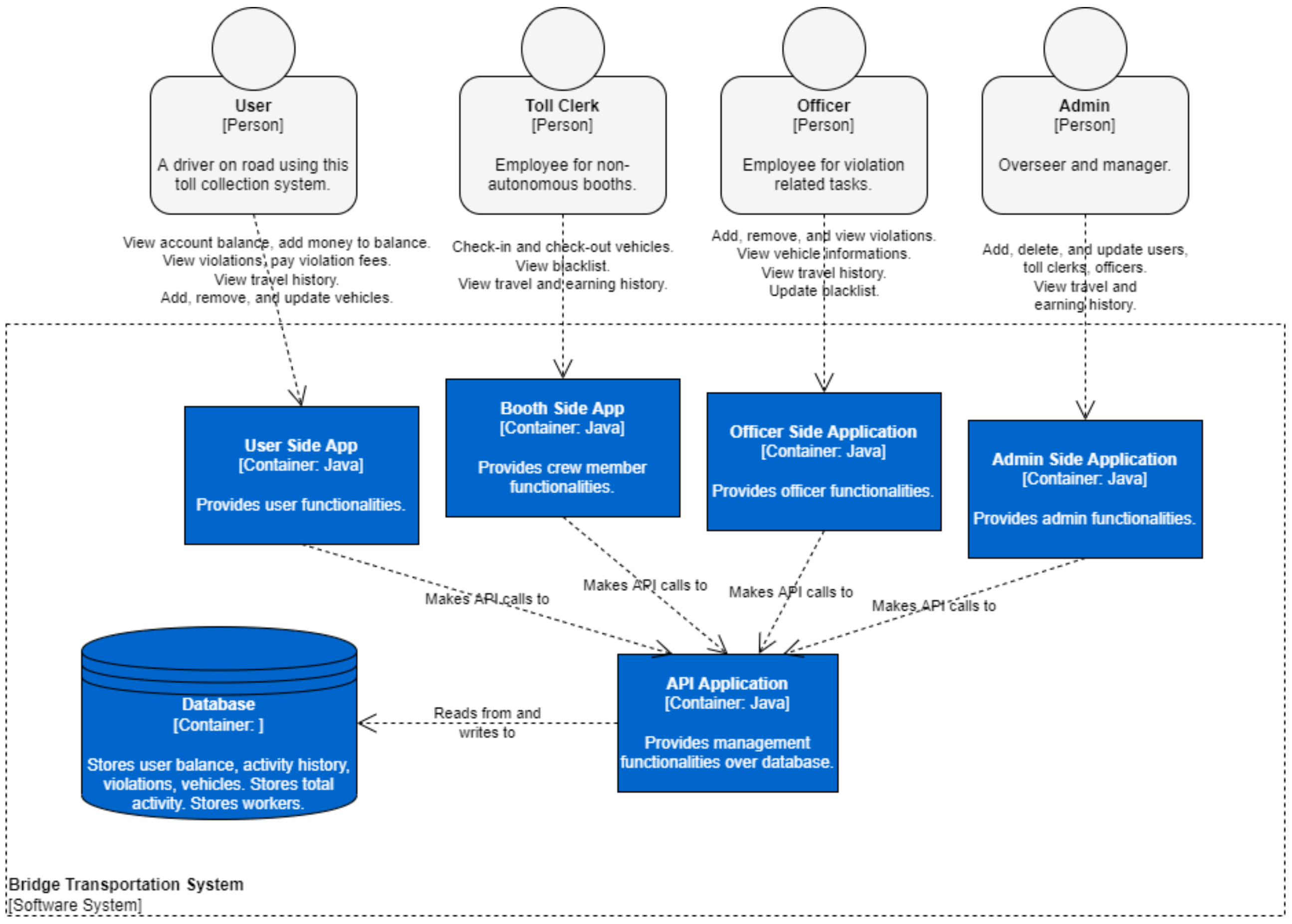
USE-CASE DIAGRAMS



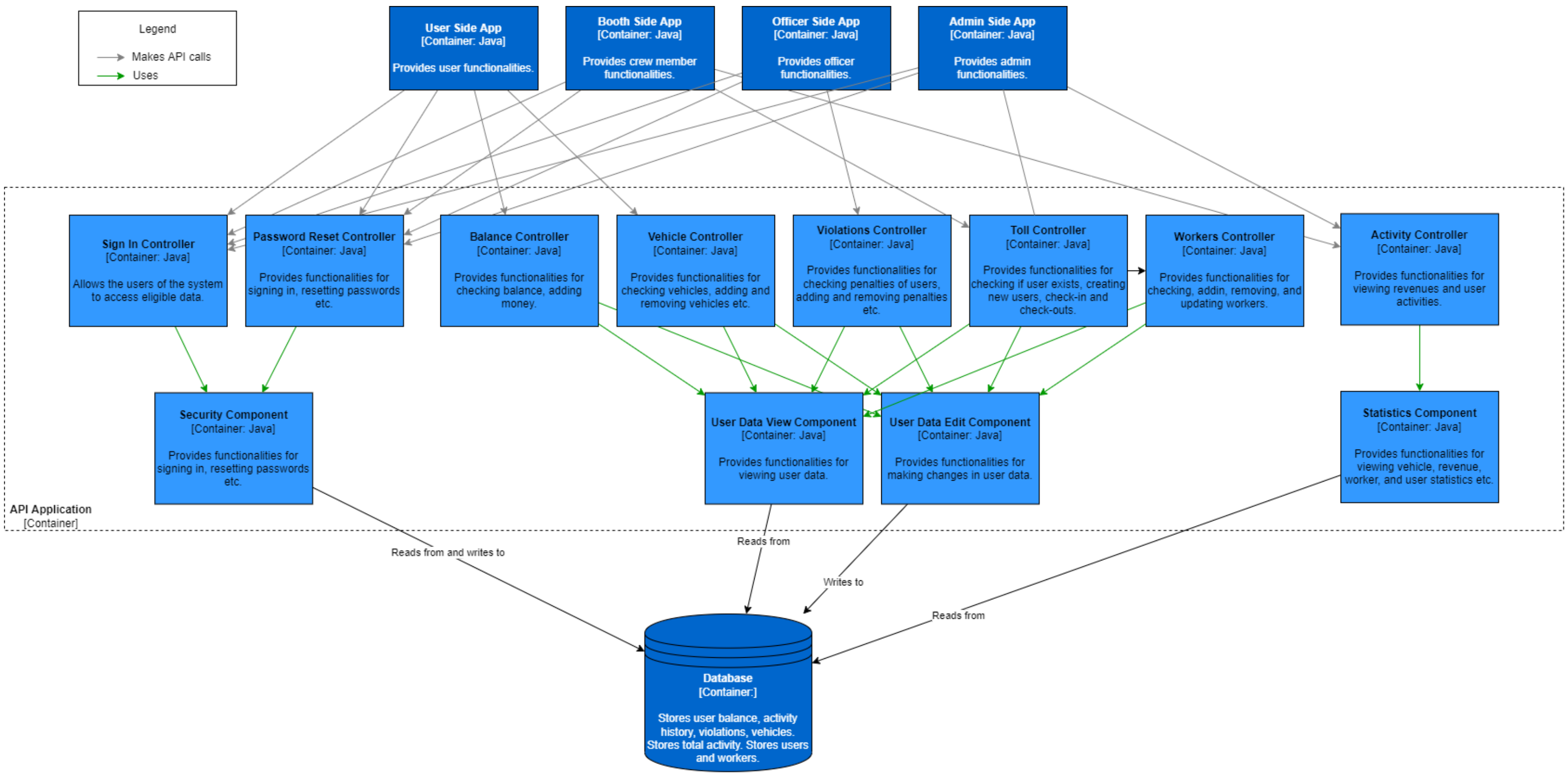
THE C4 MODEL OF THE SYSTEM **only the first three levels*


User
[Person]
A driver on road uses the toll collection system
Toll Clerk
[Person]
Employee for non-autonomus booths
Officer
[Person]
Employee for violation related tasks
Admin
[Person]
Overseer and manager
Check-in and check-out vehicles, View blacklist View daily summary
Add, remove and view viola View vehicle infos View daily summary Add, remove to/from blacklist
Add,remove,update users toll clerks and officers, View daily summary and earnings
View account balance,add money to balance, view violations, pay violation fees, View travel history, Add, remove and update vehicles
Bridge Transportation System
[Software System]

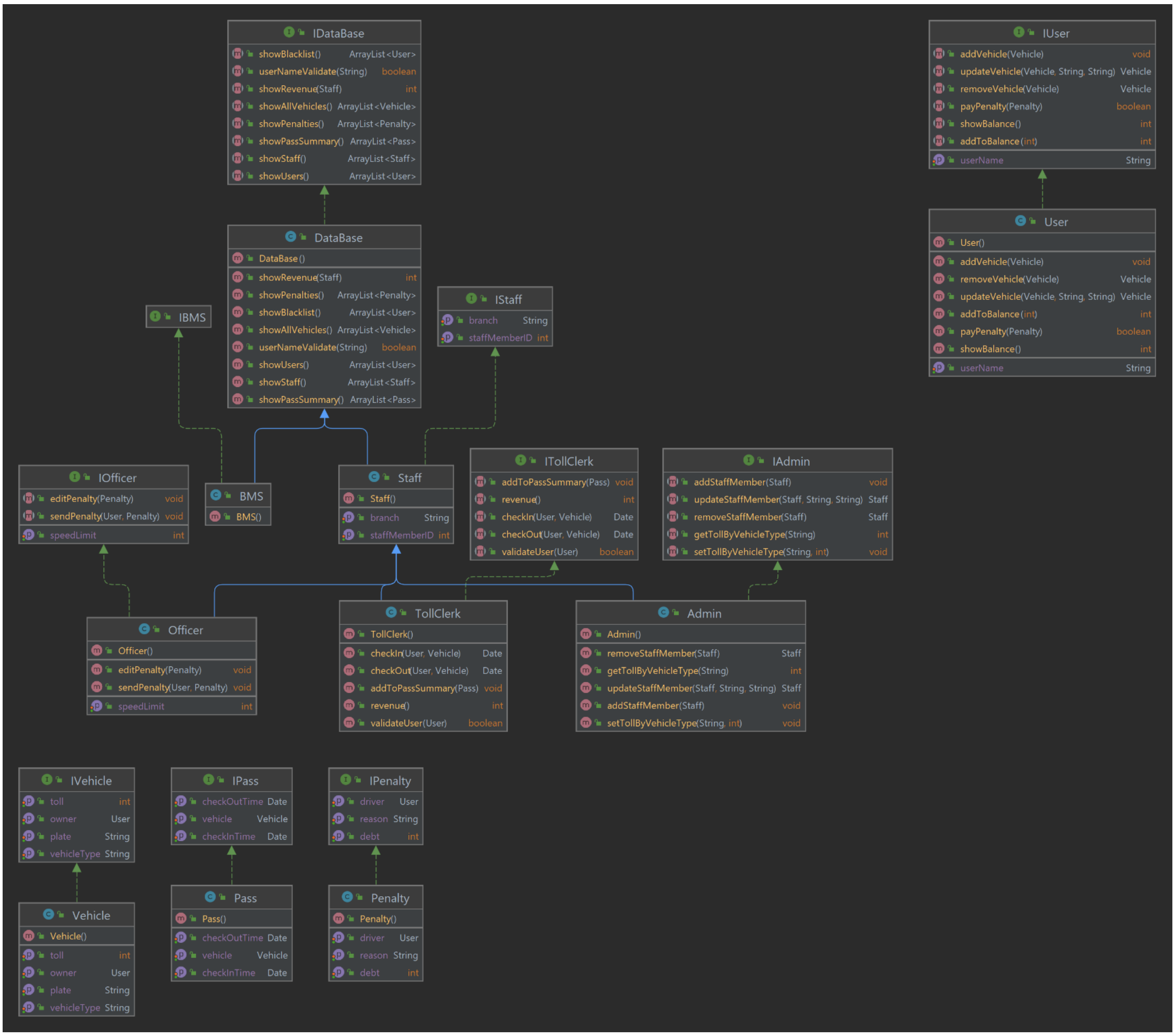
Level 2


User
[Person]
A driver on road using this toll collection system.
Toll Clerk
[Person]
Employee for non-autonomous booths.
Officer
[Person]
Employee for violation related tasks.
Admin
[Person]
Overseer and manager.
View account balance, add money to balance. View violations, pay violation fees. View travel history. Add, remove, and update vehicles.
Check-in and check-out vehicles. View blacklist. View travel and earning history.
Add, remove, and view violations. View vehicle informations. View travel history. Update blacklist.
Add, delete, and update users, toll clerks, officers. View travel and earning history.
User Side App
[Container: Java]
Provides user functionalities.
Booth Side App
[Container: Java]
Provides crew member functionalities.
Officer Side Application
[Container: Java]
Provides officer functionalities.
Admin Side Application
[Container: Java]
Provides admin functionalities.
Makes API calls to
Makes API calls to
Makes API calls to
Makes API calls to
Database
[Container:]
Stores user balance, activity history, violations, vehicles. Stores total activity. Stores workers.
Reads from and writes to
API Application
[Container: Java]
Provides management functionalities over database.
Bridge Transportation System
[Software System]

Level 3

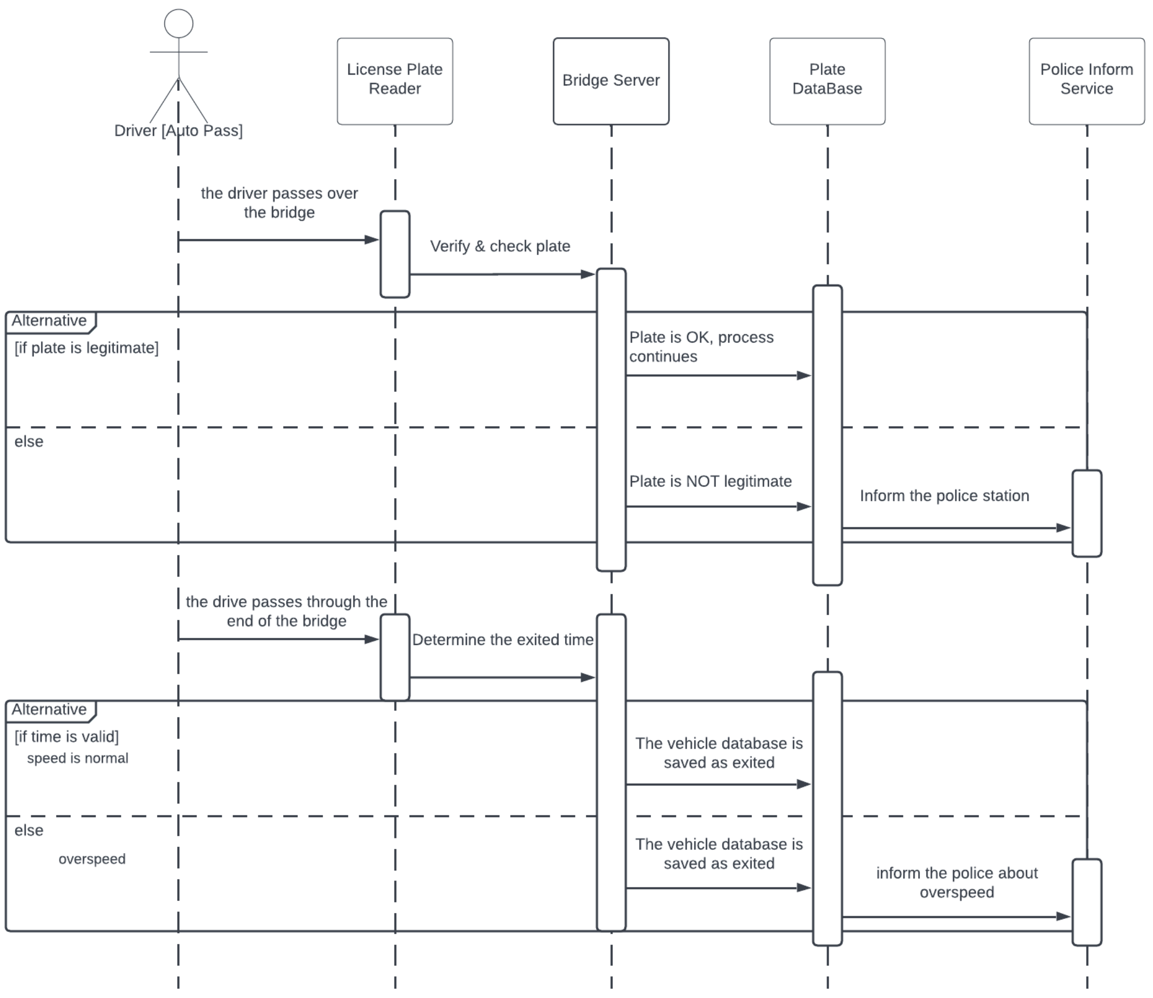

Legend
Makes API calls
Uses
User Side App
[Container: Java]
Provides user functionalities.
Booth Side App
[Container: Java]
Provides crew member functionalities.
Officer Side App
[Container: Java]
Provides officer functionalities.
Admin Side App
[Container: Java]
Provides admin functionalities.
Sign In Controller
[Container: Java]
Allows the users of the system to access eligible data.
Password Reset Controller
[Container: Java]
Provides functionalities for signing in, resetting passwords etc.
Balance Controller
[Container: Java]
Provides functionalities for checking balance, adding money.
Vehicle Controller
[Container: Java]
Provides functionalities for checking vehicles, adding and removing vehicles etc.
Violations Controller
[Container: Java]
Provides functionalities for checking penalties of users, adding and removing penalties etc.
Toll Controller
[Container: Java]
Provides functionalities for checking if user exists, creating new users, check-in and check-outs.
Workers Controller
[Container: Java]
Provides functionalities for checking, addin, removing, and updating workers.
Activity Controller
[Container: Java]
Provides functionalities for viewing revenues and user activities.
Security Component
[Container: Java]
Provides functionalities for signing in, resetting passwords etc.
User Data View Component
[Container: Java]
Provides functionalities for viewing user data.
User Data Edit Component
[Container: Java]
Provides functionalities for making changes in user data.
Statistics Component
[Container: Java]
Provides functionalities for viewing vehicle, revenue, worker, and user statistics etc.
API Application
[Container]
Reads from and writes to
Reads from
Writes to
Reads from
Database
[Container:]
Stores user balance, activity history, violations, vehicles. Stores total activity. Stores users and workers.

CLASS DIAGRAMS

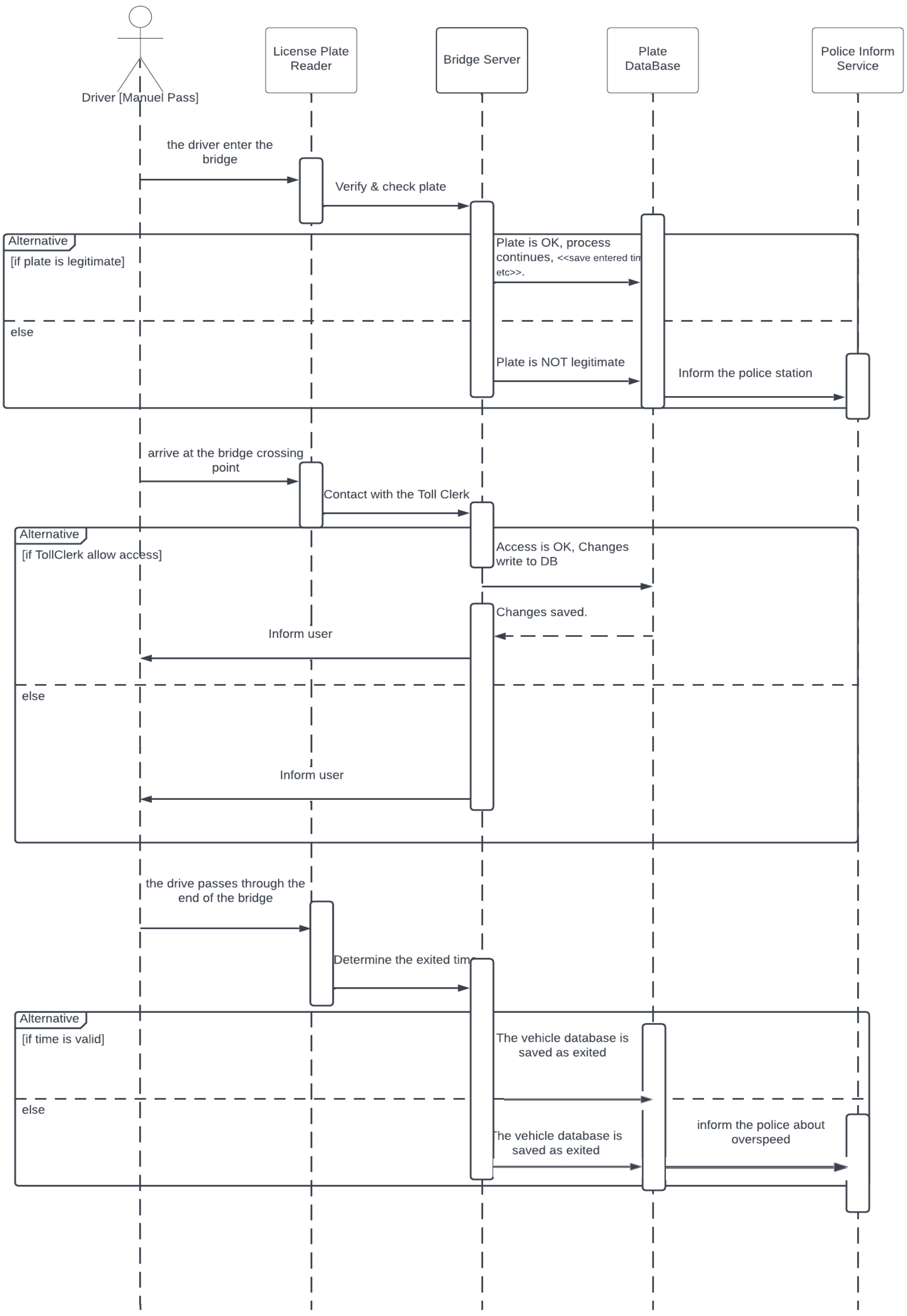

IDataBase
showBlacklist() ArrayList<User>
userNameValidate(String) boolean
showRevenue(Staff) int
showAllVehicles() ArrayList<Vehicle>
showPenalties() ArrayList<Penalty>
showPassSummary() ArrayList<Pass>
showStaff() ArrayList<Staff>
showUsers() ArrayList<User>
DataBase
DataBase()
showRevenue(Staff) int
showPenalties() ArrayList<Penalty>
showBlacklist() ArrayList<User>
showAllVehicles() ArrayList<Vehicle>
userNameValidate(String) boolean
showUsers() ArrayList<User>
showStaff() ArrayList<Staff>
showPassSummary() ArrayList<Pass>
IBMS
IStaff
branch String
staffMemberID int
IUser
addVehicle(Vehicle) void
updateVehicle(Vehicle, String, String) Vehicle
removeVehicle(Vehicle) Vehicle
payPenalty(Penalty) boolean
showBalance() int
addToBalance(int) int
userName String
User
User()
addVehicle(Vehicle) void
removeVehicle(Vehicle) Vehicle
updateVehicle(Vehicle, String, String) Vehicle
addToBalance(int) int
payPenalty(Penalty) boolean
showBalance() int
userName String
IOfficer
editPenalty(Penalty) void
sendPenalty(User, Penalty) void
speedLimit int
BMS
BMS()
Staff
Staff()
branch String
staffMemberID int
ITollClerk
addToPassSummary(Pass) void
revenue() int
checkIn(User, Vehicle) Date
checkOut(User, Vehicle) Date
validateUser(User) boolean
IAdmin
addStaffMember(Staff) void
updateStaffMember(Staff, String, String) Staff
removeStaffMember(Staff) Staff
getTollByVehicleType(String) int
setTollByVehicleType(String, int) void
Officer
Officer()
editPenalty(Penalty) void
sendPenalty(User, Penalty) void
speedLimit int
TollClerk
TollClerk()
checkIn(User, Vehicle) Date
checkOut(User, Vehicle) Date
addToPassSummary(Pass) void
revenue() int
validateUser(User) boolean
Admin
Admin()
removeStaffMember(Staff) Staff
getTollByVehicleType(String) int
updateStaffMember(Staff, String, String) Staff
addStaffMember(Staff) void
setTollByVehicleType(String, int) void
IVehicle
toll int
owner User
plate String
vehicleType String
IPass
checkOutTime Date
vehicle Vehicle
checkInTime Date
IPenalty
driver User
reason String
debt int
Vehicle
Vehicle()
toll int
owner User
plate String
vehicleType String
Pass
Pass()
checkOutTime Date
vehicle Vehicle
checkInTime Date
Penalty
Penalty()
driver User
reason String
debt int

SEQUENCE DIAGRAMS

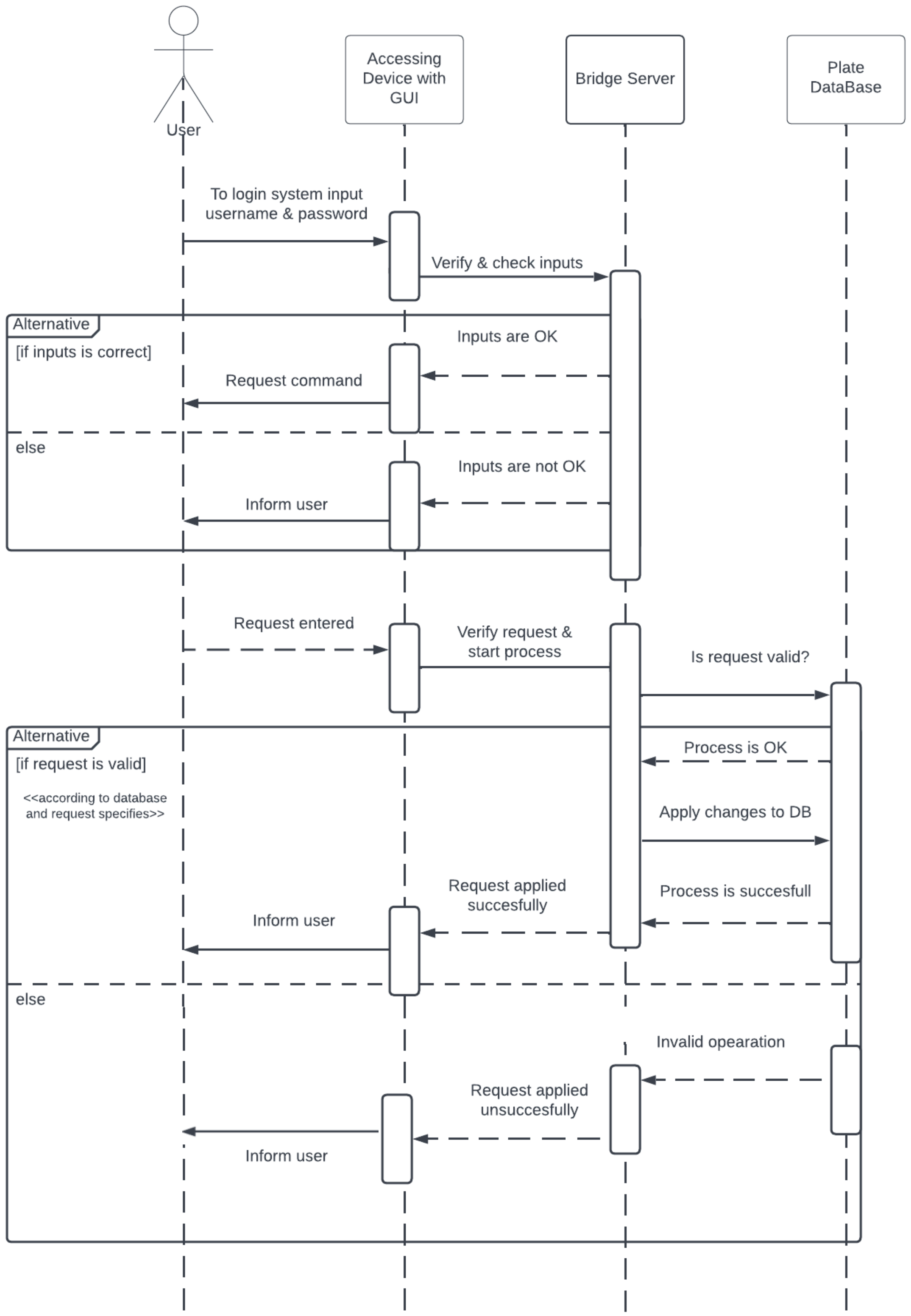
Driver (Auto Pass) Sequential Diagram


Driver [Auto Pass]
License Plate Reader
Bridge Server
Plate DataBase
Police Inform Service
the driver passes over the bridge
Verify & check plate
Alternative
[if plate is legitimate]
Plate is OK, process continues
else
Plate is NOT legitimate
Inform the police station
the drive passes through the end of the bridge
Determine the exited time
Alternative
[if time is valid]
speed is normal
The vehicle database is saved as exited
else
overspeed
The vehicle database is saved as exited
inform the police about overspeed

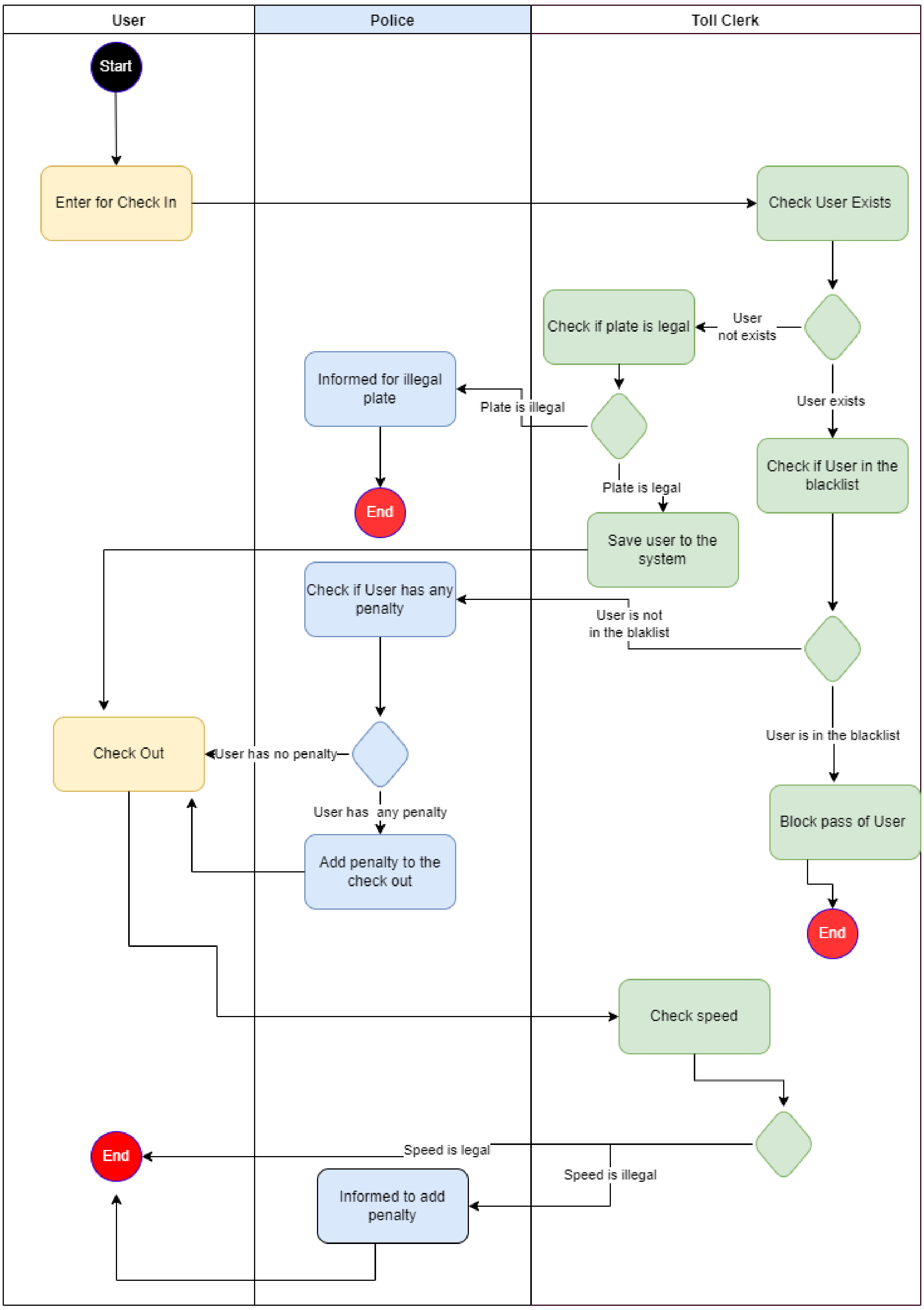
Driver (Manuel Pass) Sequential Diagram


Driver [Manuel Pass]
License Plate Reader
Bridge Server
Plate DataBase
Police Inform Service
the driver enter the bridge
Verify & check plate
Alternative
[if plate is legitimate]
Plate is OK, process continues, <<save entered tim etc>>.
else
Plate is NOT legitimate
Inform the police station
arrive at the bridge crossing point
Contact with the Toll Clerk
Alternative
[if TollClerk allow access]
Access is OK, Changes write to DB
Changes saved.
Inform user
else
Inform user
the drive passes through the end of the bridge
Determine the exited tim
Alternative
[if time is valid]
The vehicle database is saved as exited
else
The vehicle database is saved as exited
inform the police about overspeed

User Sequential Diagram



ACTIVITY DIAGRAMS



THE NON-TRIVIAL IMPLEMENTATION DETAILS

Let’s start with the definition of non-trivial. Requiring real thought or significant computing power. Often used as an understated way of saying that a problem is quite difficult or impractical, or even entirely unsolvable (“Proving P=NP is nontrivial”). The preferred emphatic form is decidedly nontrivial.

In general ,Nontrivial is a for describing any task that is not quick and easy to accomplish. It may mean "extremely" difficult and time consuming.

Speeding:

We aimed to solve the problem of speeding while crossing the bridge by the duration the vehicle took to cross the bridge. It’s a non-trivial implementation. Since The driver can go back and forth with high speed and the system won’t be able to determine if it’s violating the speed rules. The way to solve this problem, by using some external mechanism. But the automation system alone won’t be able to handle this problem.

Weight system :

Another problem which is not possible to solve will be the weight of the vehicle. We must use external mechanisms like weight measurement systems to solve this problem. But as an automation system, the software alone wouldn’t be able to solve this problem.

Encryption:

One of our non-functional requirements is, the login system should be secured and cannot be breached. For this the developer will use some strong encryption system. This falls under non-trivial implementation. As all the electronic systems can be breached more or less. Technically speaking, as of now we can make the system more secure with the latest possible technologies. But with time we may need to upgrade the system to a newer encryption system and this cycle will go on.

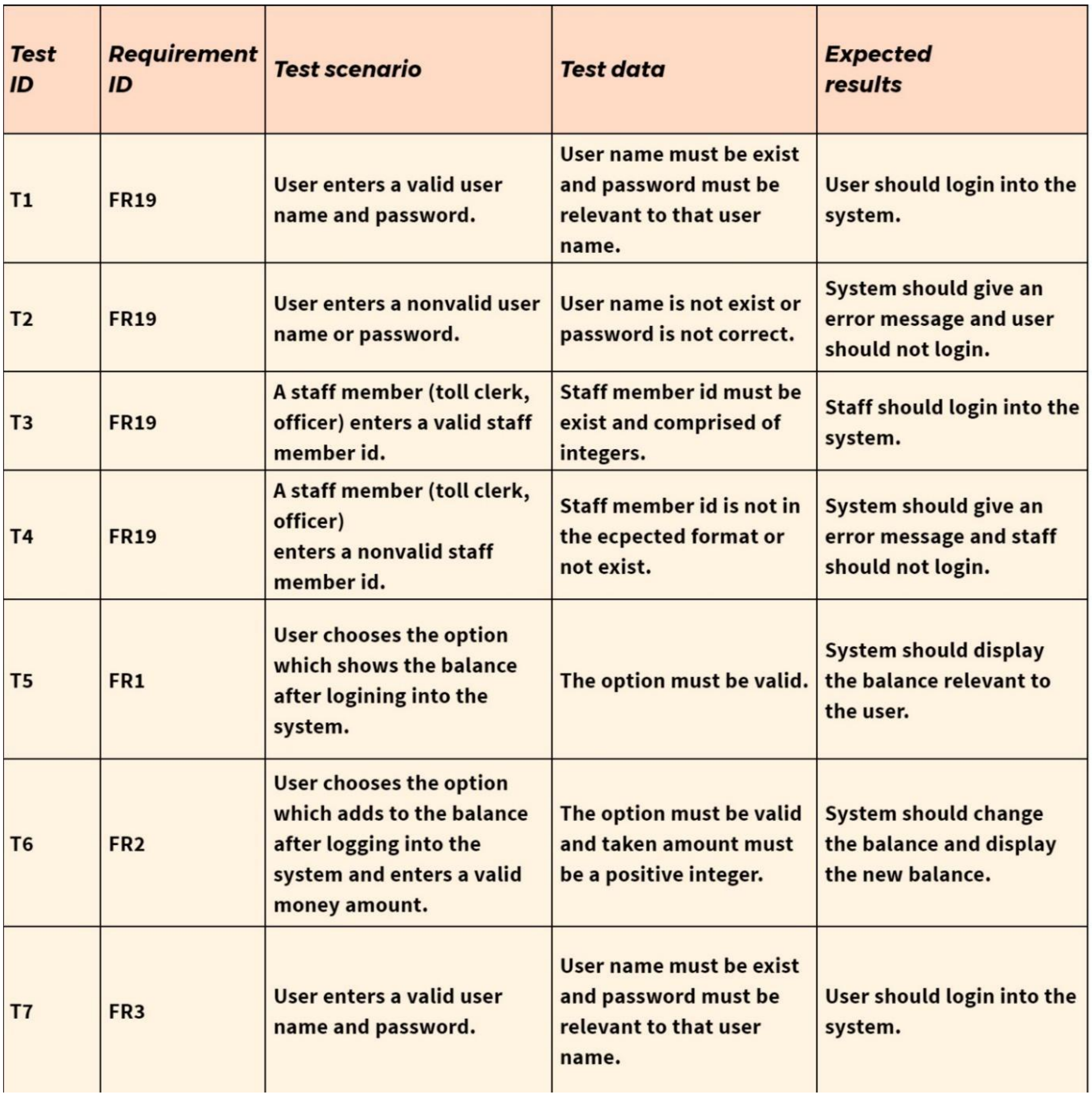
Penalty System:

The penalty system should be automated irrespective of everyone. But as an automation system we can’t fully control each level of the user. At some point the authorities will have some advantage, being the super user of the system. The only way to solve this problem is to choose the right persons and pray so that they don’t misuse the system. Basically the system can’t enforce the system authorities and users to forcefully do something but can monitor their activities only.

Hardware and Testing:

We are aiming for the system to work on any hardware. But in this era of technologies, it’s quite impossible to achieve this. From smart phones to smart watches all these electronic devices have different sets of requirements. We can customize our program to work on any device but can’t test on all the devices. So bugs will keep appearing and we have to continuously fix them.

TEST CASES



Test ID	*Requirement ID*	*Test scenario*	*Test data*	*Expected results*
T1	FR19	User enters a valid user name and password.	User name must be exist and password must be relevant to that user name.	User should login into the system.
T2	FR19	User enters a nonvalid user name or password.	User name is not exist or password is not correct.	System should give an error message and user should not login.
T3	FR19	A staff member (toll clerk, officer) enters a valid staff member id.	Staff member id must be exist and comprised of integers.	Staff should login into the system.
T4	FR19	A staff member (toll clerk, officer) enters a nonvalid staff member id.	Staff member id is not in the ecpected format or not exist.	System should give an error message and staff should not login.
T5	FR1	User chooses the option which shows the balance after logining into the system.	The option must be valid.	System should display the balance relevant to the user.
T6	FR2	User chooses the option which adds to the balance after logging into the system and enters a valid money amount.	The option must be valid and taken amount must be a positive integer.	System should change the balance and display the new balance.
T7	FR3	User enters a valid user name and password.	User name must be exist and password must be relevant to that user name.	User should login into the system.

T8	**FR19**	**User enters a nonvalid user name or password.**	**User name is not exist or password is not correct.**	**System should give an error message and user should not login.**
T9	**FR19**	**A staff member (toll clerk, officer) enters a valid staff member id.**	**Staff member id must be exist and comprised of integers.**	**Staff should login into the system.**
T10	**FR19**	**A staff member (toll clerk, officer) enters a nonvalid staff member id.**	**Staff member id is not in the ecpected format or not exist.**	**System should give an error message and staff should not login.**
T11	**FR1**	**User chooses the option which shows the balance after logging into the system.**	**The option must be valid.**	**System should display the balance relevant to the user.**
T12	**FR2**	**User chooses the option which adds to the balance after logging into the system and enters a valid money amount.**	**The option must be valid and taken amount must be a positive integer.**	**System should change the balance and display the new balance.**
T13	**FR3**	**User chooses the option which shows the penalties after logging into the system.**	**The option must be valid.**	**System should display the user's penalties and warnings.**
T14	**FR4**	**User chooses the option which pays the penalty debts after logging into the system.**	**The option must be valid and taken amount must be a positive integer.**	**System should display a message and remove the penalty.**
T15	**FR5**	**User chooses the option which shows the passing history after logging into the system.**	**The option must be valid.**	**System should display the user's passing history.**

T16	FR6	User chooses the option which adds a new vehicle and enters vehicle information after logging into the system.	The option must be valid. Plate and vehicle type must be valid.	System should adds the vehicle into the user's vehicle list.
T17	FR6	User chooses the option which updates an existing vehicle and enters vehicle information after logging into the system.	The option must be valid. New plate and vehicle type must be valid.	System should update the vehicle information.
T18	FR6	User chooses the option which deletes an existing vehicle after logging into the system.	The option must be valid. Choosen vehicle must be exist.	System should deletes the vehicle from the user's vehicle list.
T19	FR7	Toll clerk checks in the incoming vehicle which is not valid (a non-registered vehicle).	The vehicle or user is not registered.	System should give an error message and vehicle should not be checked in.
T20	FR7	Toll clerk checks in the incoming vehicle which is valid.	The vehicle and user must be valid.	System should record the date of that vehicle's checking in.
T21	FR7	Toll clerk checks out the outgoing vehicle which is already valid.	N/A	System should record the date of that vehicle's checking out.
T22	FR8	Toll clerk chooses the option which shows the blacklist after logging into the system.	The option must be valid.	System should display the blacklist.
T23	FR9	Toll clerk chooses the option which shows the passing history after logging into the system.	The option must be valid.	System should display the passing history.

T24	**FR10**	**Toll clerk chooses the option which shows the total revenue after logging into the system.**	**The option must be valid.**	**System should display the total revenue.**
T25	**FR11**	**Officer sends a new penalty to a valid user with a valid reason.**	**The user and reason must be valid.**	**System should add the new penalty.**
T26	**FR11**	**Officer edits an existing penalty.**	**The penalty and the new information must be valid.**	**System should updates the penalty.**
T27	**FR16**	**Admin adds a new toll clerk.**	**The toll clerk information must be valid.**	**System should add the new toll clerk.**
T28	**FR16**	**Admin updates an existing toll clerk.**	**The toll clerk information must be valid.**	**System should change the toll clerk information.**
T29	**FR16**	**Admin removes an existing toll clerk.**	**The toll clerk must be exist.**	**System should removes the toll clerk from the system.**
T30	**FR18**	**Admin chooses the option which shows the passing history.**	**The option must be valid.**	**System should display the passing history.**
T31	**FR18**	**Admin chooses the option which shows the total revenue after logging into the system.**	**The option must be valid.**	**System should display the total revenue.**

PERFORMANCE ANALYSIS

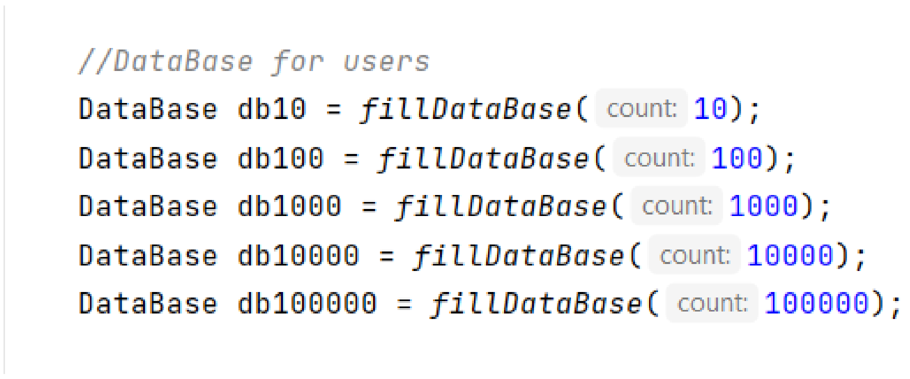
Theoretically Analysis

In DataBase, we are using the BST to store User`s class data because in our bridge automation pass system there are more than million plate so that means if we want to find a user system must be very fast so BST is ideal for that because passing process is not sequential, and time complexity of finding a person in BST is O(logn).

In our DataBase`s User`s BST structure we are expected to logarithmically increase.

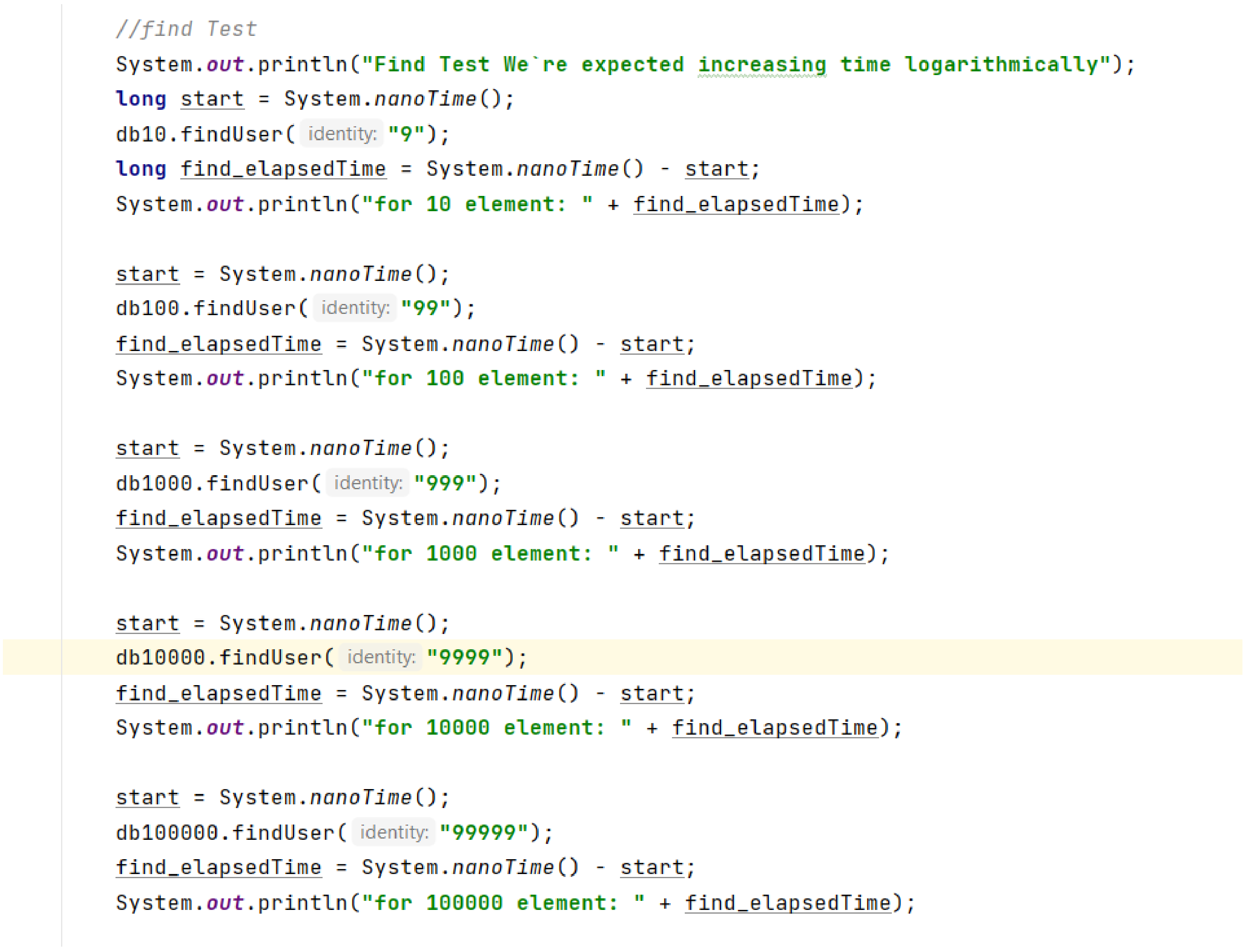
Empirically Analysis

first created data structures & fill with random data



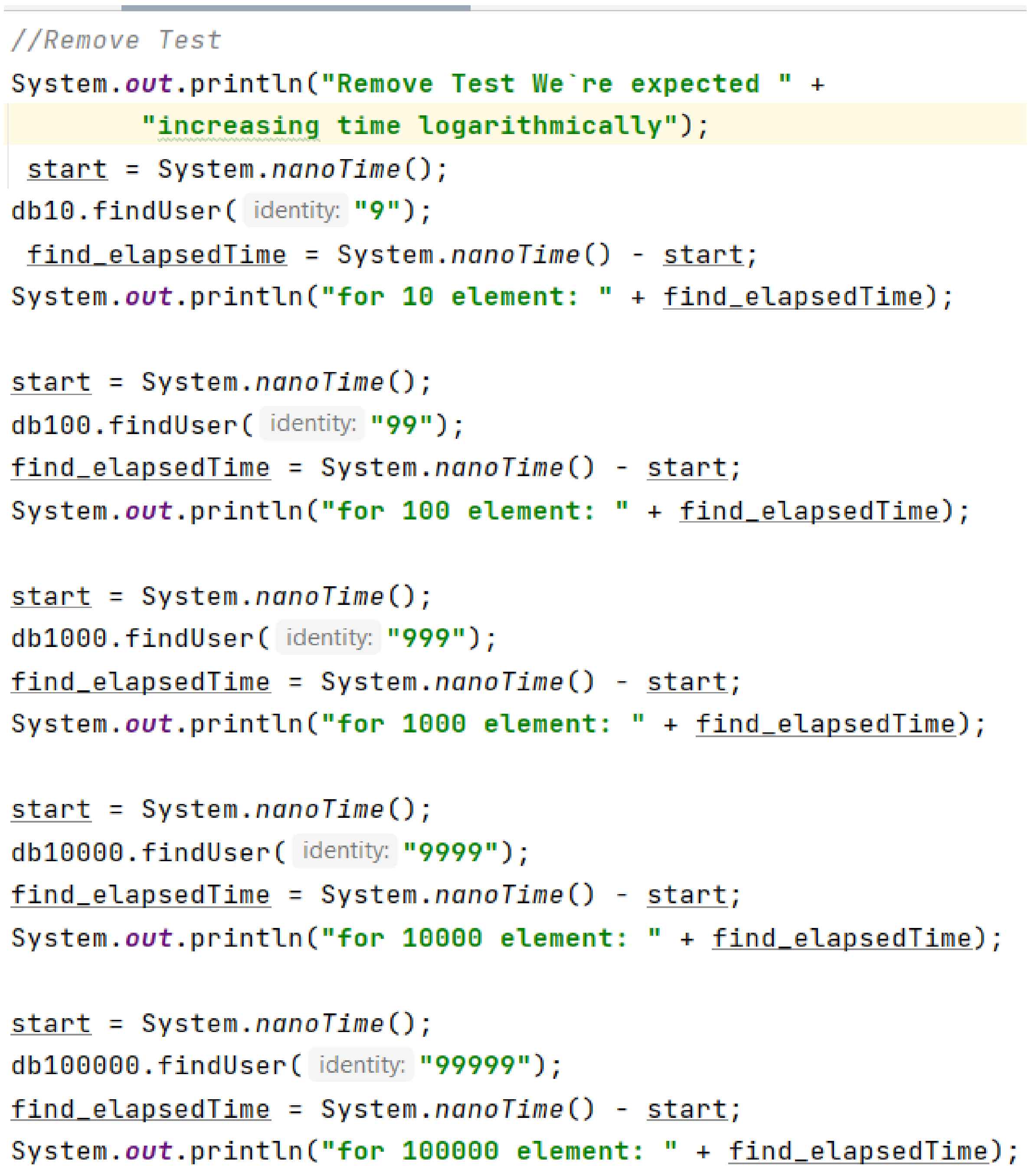
```java
//DataBase for users
DataBase db10 = fillDataBase( count: 10);
DataBase db100 = fillDataBase( count: 100);
DataBase db1000 = fillDataBase( count: 1000);
DataBase db10000 = fillDataBase( count: 10000);
DataBase db100000 = fillDataBase( count: 100000);
```

In this test we tried to find an element in the BST data structure we were expect logarithmically increase



```
//find Test
System.out.println("Find Test We`re expected increasing time logarithmically");
long start = System.nanoTime();
db10.findUser( identity: "9");
long find_elapsedTime = System.nanoTime() - start;
System.out.println("for 10 element: " + find_elapsedTime);

start = System.nanoTime();
db100.findUser( identity: "99");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 100 element: " + find_elapsedTime);

start = System.nanoTime();
db1000.findUser( identity: "999");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 1000 element: " + find_elapsedTime);

start = System.nanoTime();
db10000.findUser( identity: "9999");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 10000 element: " + find_elapsedTime);

start = System.nanoTime();
db100000.findUser( identity: "99999");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 100000 element: " + find_elapsedTime);
```

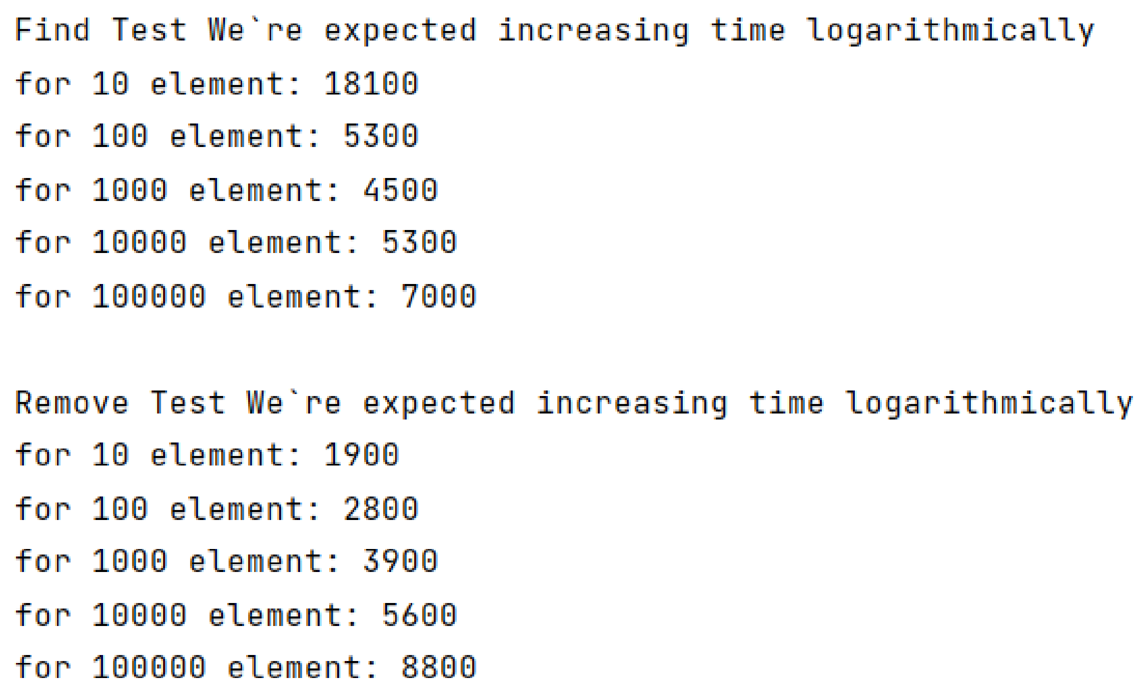
In this test we tried to remove an element in the BST data structure we were expect logarithmically increase



```
//Remove Test
System.out.println("Remove Test We`re expected " +
        "increasing time logarithmically");
 start = System.nanoTime();
db10.findUser( identity: "9");
 find_elapsedTime = System.nanoTime() - start;
System.out.println("for 10 element: " + find_elapsedTime);

start = System.nanoTime();
db100.findUser( identity: "99");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 100 element: " + find_elapsedTime);

start = System.nanoTime();
db1000.findUser( identity: "999");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 1000 element: " + find_elapsedTime);

start = System.nanoTime();
db10000.findUser( identity: "9999");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 10000 element: " + find_elapsedTime);

start = System.nanoTime();
db100000.findUser( identity: "99999");
find_elapsedTime = System.nanoTime() - start;
System.out.println("for 100000 element: " + find_elapsedTime);
```

Output`s of the BST User stored data structure empirically test`s

(Find an element & remove an element)



```
Find Test We`re expected increasing time logarithmically
for 10 element: 18100
for 100 element: 5300
for 1000 element: 4500
for 10000 element: 5300
for 100000 element: 7000

Remove Test We`re expected increasing time logarithmically
for 10 element: 1900
for 100 element: 2800
for 1000 element: 3900
for 10000 element: 5600
for 100000 element: 8800
```

As we see in the output, the increase in the processing time is logarithmically.