Summary:

*) The application is written in Java using Spring boot and JPA.

*) For demo purposes the application uses h2 in-memory database.

*) The build system and dependency management system used is maven.

*) The application has internationalization support with english being supported currently.

*) IDE used is intellij and intellij workspace has been submitted on hackerrank

*) integration tests have been submitted

*) The application has 2 REST API's.

1) GET - /user/{userName}/appointments
Gets the service tier as input from query parameter and returns 5 latest available appointment date and times for the requested tier

Example usage:
```
curl -X GET \
  'http://localhost:8080/v1/user/manu/appointments?service_tier=Tier1' \
  -H 'Content-Type: application/json' \
  -H 'Postman-Token: 7cb46006-df81-4c95-a6ce-83632d6510a6' \
  -H 'cache-control: no-cache'
```

This endpoint is a dumb endpoint as it doesn't do any validations apart from:
*) checking that the user, got from pathparam username, exists in the database
Response code 404 with message explaining error
*) response code 400 if inputted tier is a valid tier (one from Tier1 or Tier2 or Tier3)

Sample Output:
```
["2019-02-11T08:00:00","2019-02-11T08:00:00","2019-02-11T08:00:00","2019-02-11T08:00:00"
,"2019-02-11T08:00:00"]
```

This endpoint ensures that all the date and times being returned are in the future, are between 8am - 5pm and don't lie on a Sunday.

2) POST - /user/{userName}/appointments
Gets the input as raw post body having parameters serviceTier and start
serviceTier - is for the serviceTier (one from Tier1 or Tier2 or Tier3) for which appointment is requested

Start - specifies the start time of the requested appointment

Example usage:
curl -X POST \
  http://localhost:8080/v1/users/nanu/appointments \
  -H 'Content-Type: application/json' \
  -d '{"serviceTier":"Tier3","start":"2099-02-08T12:34:38.073"}
'
This endpoint checks the input in the following ways:

*) user for whom the appointment is being booked for actually exists
*) the requested appointment date and time lies between 8am - 5pm and doesn't lie on a sunday
*) the requested date and time are in the future but not more than 60 days in the future
*) the user is not trying to book more than 1 appointment at a time
*) only returning users can book appointment between 8am - 10am if the appointment date is not in the coming 3 days. If the appointment date is in the coming 3 days then anyone can book any available appointment slot between 8am - 5pm

Sample Output:
{"start":"2099-02-08T12:34:38.073","name":"agent 1"};
Output contains the starttime of the appointment and the name of the agent.

The endpoint saves the appointment details in the database and updates the next available appointments in each tier for the agent who was booked in this reservation.

Compile and Run:
cd reservation_system
mvn spring-boot:run

The server will start at port 8080

Database Schema:

The database schema consists of 4 tables:

NEXT_AVAILABLE_RESERVATION - contains the tier wise latest available appointment for every agent.

RESERVATION - has details of booked reservations

USER - has details of the users

AGENT - has list of all the agents

AGENT

ID	NAME

NEXT_AVAILABLE_RESERVATION

ID	NEXT_AVAILABLE_RESERVATION	SERVICE_TIER	AGENT_ID

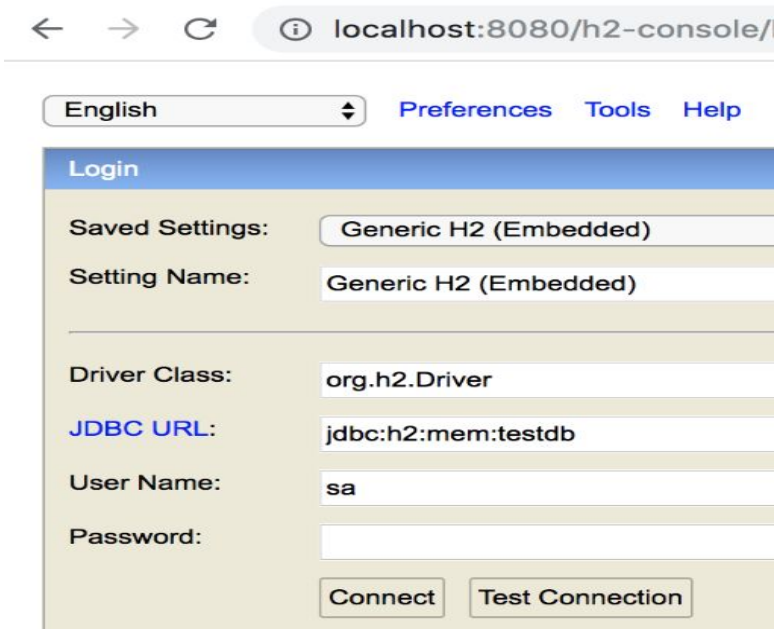
RESERVATION

ID	END	SERVICE_TIER	START	AGENT_ID	USER_ID

USER

ID	IS_NEW_USER	LOCALE	USER_NAME

When the project is running, the database can be browsed from URL - http://localhost:8080/h2-console/



Integration Testing:

Integration tests can be run using the following commands:

```
cd reservation_system
mvn test
```

The integration tests automatically start the server, database instance and then populate the database instance with Agents, Available appointments, 1 returning user and 1 new user.

The tests cover following scenarios:

1) Confirm that appointment booking is successful with valid inputs
2) Confirm that appointment booking fails on sundays
3) Confirm that appointment booking fails if after office hours time is requested
4) Confirm that appointment booking fails if the requested date time is in the past
5) Confirm that appointment booking fails if requested date was not returned by the GET available appoint rest endpoint
6) Confirm that appointment booking fails if requested date time is more than 60 days in the future
7) Confirm that appointment booking fails if multiple reservations are made within a 60 day period
8) Confirm that appointment booking fails for new user if the requested date time is beyond next 3 days and is between 8am - 10am

Future Improvements:

1) Support user timezones instead of using the servers timezone
2) Remove hardcoding for working office hours and days
3) Unit test functionality
4) Use swagger or some other ways for generating API specifications and docs so that it is easier to write clients for the API