2IMN15 - Parking System Assignment

Group 36: Stijn Kamp - 0938450 , Omar Aly - 1586459

I. Introduction

This report shows the system description of a LwM2M-based parking system. LwM2M protocol is an IoT protocol that allows the management of resource-constrained devices through enabling the use of such devices by other applications through lightweight data communication, using CoAP messages.

Our implementation of the parking system follows the system description outlined in the assignment. So as mentioned per the assignment, we used only one parking lot in our system, which was merged with the parking system server. However, due to the changes in the assignment structure this year, some of the resources of the objects provided were deemed useless. For example, the x and the y positions of the car were ignored entirely in the implementation, and thus the object ”Multiple access Joystick” was ignored. Also, the Addressable Text Display object was not used, except for the Text field, which has been used to show an acknowledgement of a car entering a parking spot after a reservation, and to show the parking fee when a vehicle is leaving the parking spot.

II. System Architecture

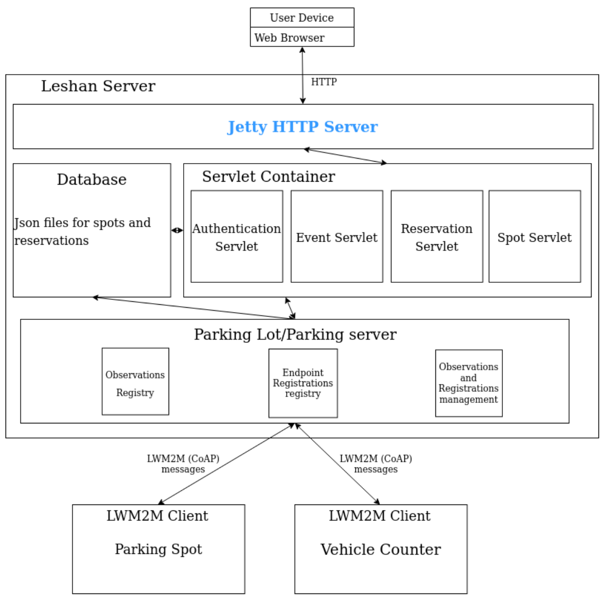

Figure 1: System Architecture of the LwM2M Parking System

Our system consists mainly of 4 main components which are the Web server, the servlets, the database, as well as the local server (which is the parking lot). The system supports LwM2M clients (such as the parking spot or the vehicle counters), which are provided in the assignment. The system provides a user interface to allow the system user to view available parking spots as well as to reserve one or more of them.

III. Functionalities of System Components

A. Web Server

The web server’s task is to provide a user-friendly GUI that shows the available parking spots and allows reserving them. In addition, when logged on using the admin user credentials, the web server shall allow the admin user to change the rate for the parking spots.

B. Servlets

The servlets provide an API to interface with the HTTP messages from the jetty server. The reservation servlet handles the HTTP messages that include the reservation requests, and saves the reservation information, including the license plate, in the database in JSON format. The spot servlet assures the web server can show all the spots that are available in the parking lot.

The authentication servlet assures that only a registered user can be access the parking lot functionalities. In addition, it assures that only the admin user can be able to access the admin user functionalities, such as changing the rate of a parking spot. Finally, the event servlet task is to listen and to handle the parking lot actions, such as the addition of a new parking spot to the lot.

IV. Technologies used

A. Leshan: LwM2M Clients and Servers

In this assignment, we have preferred to use Leshan as the LwM2M server for a number of reasons. First, the LwM2M Clients (the parking spot, vehicle counter, the addressable text display, and the Multiple axis joystick) were already provided as reference implementations., which we would have had to write in C/C++ had we decided to use Wakaama. Since Leshan is based on java (which uses the OOP paradigm), the libraries provided were far more structured and easier-to-use, compared to using Wakaama. Finally, Leshan provided the leshan-server-demo project, which included a full architecture of a system, including the servlets and the web interface, which we used as a guidance in structuring our project.

B. Apache Maven

we have used Apache maven for importing dependencies into our project in an easy-to-use manner. We have used it to import the leshan server libraries, as well as libraries for using JSON messages.

C. *Database: MySQL*

To store all information about users, reservations and entries we used MySQL. The reason for this is the good integration using the JDBC connector of java in order to build the database functionalities. We have used the Hibernate library, to link the different classes in the Java application. In this way it is possible to query database rows using the properties of a specific class. The database structure is listed in appendix A as SQL query file.

D. *Javascript framework VueJS*

In order to speed up the development of the web application the decision has been made to use the VueJS framework. This framework is very useful when there is a need for real-time updating on the website. A VueX store has been created to handle all the data communication with the Jetty server. It makes sure that all the data is up to date and listens to the data-stream for real-time updates. VueJS will then handle that the views are updated appropriately looking at the new updated data. In Figure 2 the communication between the web client and jetty server is shown, using this mechanism we show the status of the parking lot in real time i.e., the state of the parking spots or reservations.

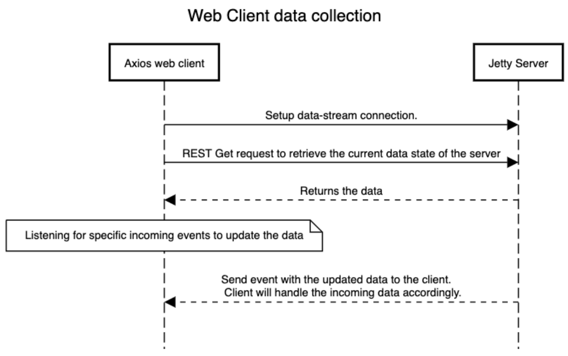
Web Client data collection
Axios web client
Jetty Server
Setup data-stream connection.
REST Get request to retrieve the current data state of the server
Returns the data
Listening for specific incoming events to update the data
Send event with the updated data to the client.
Client will handle the incoming data accordingly.

Figure 2: Data collection by the Axios web client from the Jetty server

E. *CSS framework: Tailwind*

We have used tailwind in order to make the UI of the web application. It is a very flexible framework, and you can design the interface using only classes instead of also writing CSS itself.

V. Scenarios

The general scenarios have been implemented and are shown in this video https://vimeo.com/503557923. The general overview of the scenarios is shown in Figure 3 and Figure 4.

VI. Technical Difficulties

A. *Using Leshan*

Although, using leshan was a far easier and a better documented option compared to using Wakaama, it was the much harder option for both of the team members, since we have never used Java before. Also, we had some hard time understanding the framework through the basic tutorials provided on github. So, our structure depended mainly on the reference leshan server demo, which was not properly documented either. However, good naming conventions made it doable to understand and use the code. It is also worthy to mention that there was no already provided tutorials, which made it significantly harder to recognise the expected system structure as well as the system components.

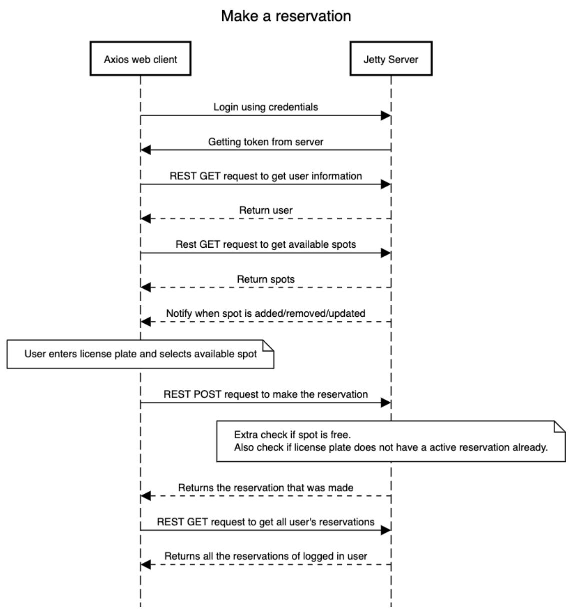
Make a reservation
Axios web client
Jetty Server
Login using credentials
Getting token from server
REST GET request to get user information
Return user
Rest GET request to get available spots
Return spots
Notify when spot is added/removed/updated
User enters license plate and selects available spot
REST POST request to make the reservation
Extra check if spot is free.
Also check if license plate does not have a active reservation already.
Returns the reservation that was made
REST GET request to get all user's reservations
Returns all the reservations of logged in user

Figure 3: Making a reservation on the web user interface.

B. *Syncing system components*

One of the main problems encountered was assuring that all separate components in the system (LwM2M clients, LwM2M server, database and web application) are in sync with each other and do not interfere. For example, when a new parking spot (LwM2M client) is registered, some of its resources has to be observed, and the web app has to be informed of the existence of this new spot, with its state included, and the web app should update the UI to make this spot available for the user to book.

C. *Bugs with LwM2M Clients*

There was a major error in the vehicle counter provided, which was that the counter always increments by one whenever the last plate field is updated, regardless of the direction of the vehicle (whether it being it Enter, or Exit). Therefore, we have not used the value from the vehicle counter, and we have not saved that value either in the database.

VII. Work Division

Omar: Working on the Parking lot class and interaction with the LwM2M clients, and the affiliated models used with the parking lot.
Stijn: Working on the web application, the web server and its associated servlets, the database, and helping out with the parking lot class. Also, he recorded the demos provided in the report.

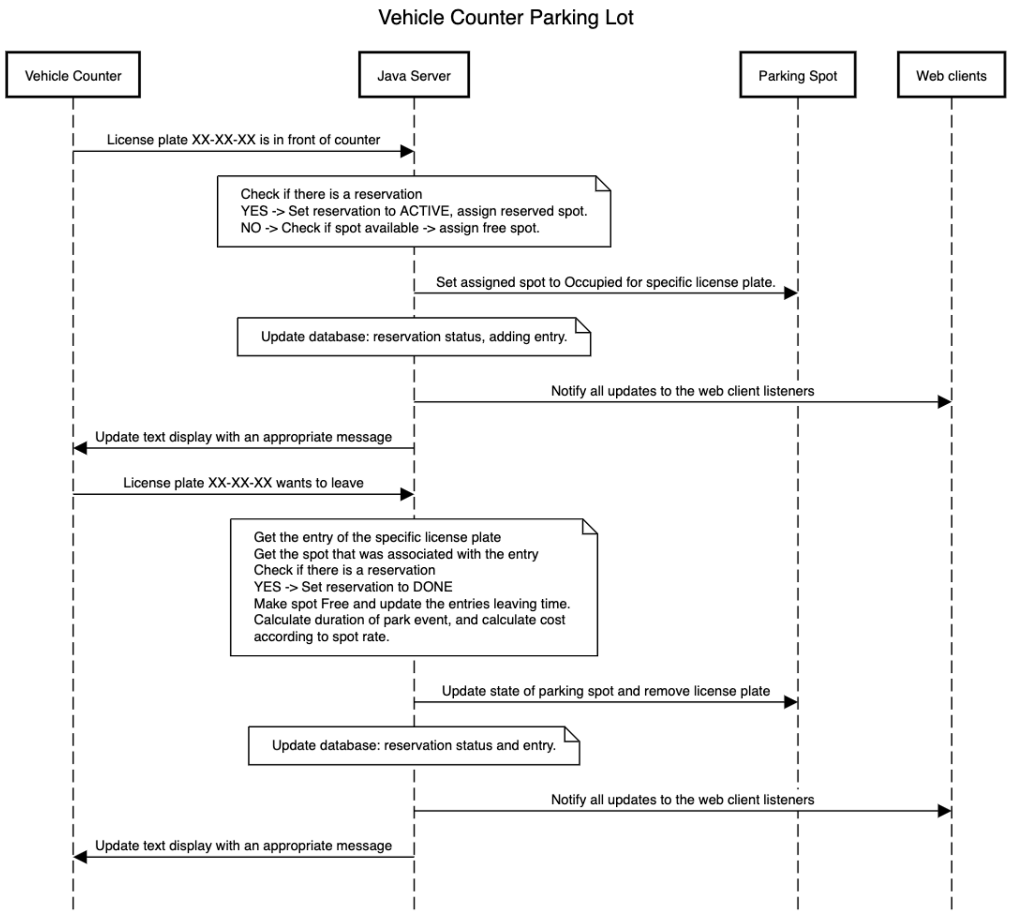
Vehicle Counter Parking Lot
Vehicle Counter
Java Server
Parking Spot
Web clients
License plate XX-XX-XX is in front of counter
Check if there is a reservation
YES -> Set reservation to ACTIVE, assign reserved spot.
NO -> Check if spot available -> assign free spot.
Set assigned spot to Occupied for specific license plate.
Update database: reservation status, adding entry.
Notify all updates to the web client listeners
Update text display with an appropriate message
License plate XX-XX-XX wants to leave
Get the entry of the specific license plate
Get the spot that was associated with the entry
Check if there is a reservation
YES -> Set reservation to DONE
Make spot Free and update the entries leaving time.
Calculate duration of park event, and calculate cost according to spot rate.
Update state of parking spot and remove license plate
Update database: reservation status and entry.
Notify all updates to the web client listeners
Update text display with an appropriate message

Figure 4: Showing the scenario when a vehicle is scanned by the vehicle counter.

VIII. Usage

A. File structure

- parking-app: VueJS web application compiled using nodeJS
- parking-system-server: The Java Server with all functionalities inside.
  - src/main/resources: Configurations files and the final compiled web application
  - src/main/java/com/iot/parking: The actual Java application
    - servlets: All the Jetty Servlets used
    - models: All the model classes, for the LwM2M clients and database entities.
    - utils: All separate classes handling i.e. database connections, authentication and other functionalities.

B. Start up

Run the App.java file. When this is running you can connect to the web application with the following url: http://localhost:8082. There you can login with one of the following credentials:

1) username: test, password: Password → to see the user interface
2) username: admin, password: Password → to see the admin interface

Clients can be added using the already provided java leshan client references.
Adding parking spot + vehicle counter run:

```
java -jar lwm2m-client.jar -u
    127.0.0.1:5683 -pos "3.0:4.0" -n [
    SPOTNAME] -parkingspot -
    vehiclecounter
```

Adding only parking spot

```
java -jar lwm2m-client.jar -u
    127.0.0.1:5683 -pos "3.0:4.0" -n [
    SPOTNAME] -parkingspot
```

Appendix A - MySQL structure

```sql
--
-- Table structure for table `access_tokens`
--

CREATE TABLE `access_tokens` (
  `id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `token` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Table structure for table `entries`
--

CREATE TABLE `entries` (
  `id` int(11) NOT NULL,
  `spot_id` varchar(15) NOT NULL,
  `reservation_id` int(10) UNSIGNED NOT NULL,
  `license_plate` varchar(15) NOT NULL,
  `enter_time` timestamp NOT NULL DEFAULT current_timestamp(),
  `leave_time` timestamp NULL DEFAULT NULL,
  `cost` decimal(8,2) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Add data for table `entries`
--

INSERT INTO `entries` (`id`, `spot_id`, `reservation_id`, `license_plate`, `enter_time`, `leave_time`, `cost`) VALUES
(36, '0', 0, 'AA-00-BB', '2021-01-21␣22:20:17', '2021-01-21␣22:27:41', '35.00');

-- --------------------------------------------------------

--
-- Table structure for table `hibernate_sequence`
--

CREATE TABLE `hibernate_sequence` (
  `next_val` bigint(20) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `hibernate_sequence`
--

INSERT INTO `hibernate_sequence` (`next_val`) VALUES
(70);

-- --------------------------------------------------------

--
-- Table structure for table `rates`
--

CREATE TABLE `rates` (
  `id` int(11) NOT NULL,
  `spot_id` varchar(15) NOT NULL,
  `price_per_minute` decimal(8,2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `rates`
--

INSERT INTO `rates` (`id`, `spot_id`, `price_per_minute`) VALUES
(1, '0', '3.00');

-- --------------------------------------------------------

--
-- Table structure for table `reservations`
--

CREATE TABLE `reservations` (
  `id` int(11) NOT NULL,
  `spot_id` varchar(15) NOT NULL,
  `license_plate` varchar(15) NOT NULL,
  `status` varchar(15) NOT NULL DEFAULT 'PENDING',
  `created_at` timestamp NOT NULL DEFAULT current_timestamp(),
  `user_id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `reservations`
--

INSERT INTO `reservations` (`id`, `spot_id`, `license_plate`, `status`, `created_at`, `user_id`) VALUES
(65, '0', 'AA-00-BB', 'DONE', '2021-01-22␣10:15:32', 9);

-- --------------------------------------------------------

--
-- Table structure for table `users`
--

CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `username` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `name` varchar(255) NOT NULL,
  `role` varchar(15) NOT NULL DEFAULT 'USER'
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `users`
--

INSERT INTO `users` (`id`, `username`, `password`, `name`, `role`) VALUES
(9, 'test', '1000:3b08a3f1fa02f246b5c2ffce121bcbaa:189b84d2b3b3aa02f14552b1ad6ff188067c9ef19d60b98383d9c636349445278dfc0606e6e95e4ea5357e9df3981eba327d4e64597057e951a
(42, 'bigtest', '1000:3b08a3f1fa02f246b5c2ffce121bcbaa:189b84d2b3b3aa02f14552b1ad6ff188067c9ef19d60b98383d9c636349445278dfc0606e6e95e4ea5357e9df3981eba327d4e64597057e

--
```

```
-- Indexes for dumped tables
--

--
-- Indexes for table `access_tokens`
--
ALTER TABLE `access_tokens`
  ADD PRIMARY KEY (`id`),
  ADD KEY `FKjxi0wavfc9xw97x1mhuc8nphm` (`user_id`);

--
-- Indexes for table `entries`
--
ALTER TABLE `entries`
  ADD PRIMARY KEY (`id`);

--
-- Indexes for table `rates`
--
ALTER TABLE `rates`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `spot_id` (`spot_id`);

--
-- Indexes for table `reservations`
--
ALTER TABLE `reservations`
  ADD PRIMARY KEY (`id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `username` (`username`);

--
-- Constraints for table `access_tokens`
--
ALTER TABLE `access_tokens`
  ADD CONSTRAINT `FKjxi0wavfc9xw97x1mhuc8nphm` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`);

--
-- Constraints for table `reservations`
--
ALTER TABLE `reservations`
  ADD CONSTRAINT `reservation_user_id` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE SET NULL ON UPDATE CASCADE;
```