OPnGO test assignment

Design and implement core of a digital parking management system. The main functionality of the system is to let people create an account for a customer (email, password), add their vehicles (some descriptive name and licence plate number) and top-up some money to a virtual wallet. Once this is done when one of the registered vehicles arrives to the parking lot, a camera (parking management system - PMS) will read the vehicle's licence plate number and ask the system to be developed if a vehicle with a given licence plate number is registered in the system and if it is then a bar/gate will be opened so a person can drive in. Once the driver has driven in, parking session is started (current time is stored). When a parked car is leaving a parking lot then **parking session is stopped, price calculated, the resulting amount is withdrawn from a wallet and invoice sent to the user.**

Functional requirements:

- When a user drives into a park a parking session is started. Parking session should include a reference to a vehicle that has driven in as well as time when this happened. Beside reference to a vehicle we must also cache a vehicle's number to prevent situations when a user theoretically can change a vehicle's number after the session has already started.
- Any given vehicle can belong only to one user's account. Vehicle license number should be unique.
- If a user has parked at least once then it should be possible to credit him/her for X eur (value must be fetched from a configuration), meaning that if a wallet has 5 EUR left but upon leaving a car park price calculated for the session is 9 eur, then we will credit the user, his wallet balance in this case is going to be -4 eur. We will allow crediting only in case when user had parked at least once and his the wallet's balance still has some positive amount Y (comes from the configuration as well).
- When a parking session is finished we must store time when the session has been finished as well as total price for the session.
- Whenever money is withdrawn from the user's wallet we need to keep a record of that.
- Every parking lot (domain name - asset) in the system must have at least two things stored:
    - Where asset is located, i.e its address
    - If a given asset is active and vehicles can park car there.

      It shouldn't be possible to add a new a parking lot to the system if it doesn't have those values specified for it.

      In first version of the system logic controlling if an asset is active is very straightforward - just check checking the field, but later it will become complex,

take that into account when designing a model. **Hint**: What can you do to localize this decision making process?
- Create a very basic price engine that for every 15 minutes spent inside the parking lot Z amount (should come from configuration) is added to the resulting total. “Z” is charged for every new 15 minutes slot started, i.e for 31 minutes session time the final cost is going to be 3*Z.
- Send invoice with basic HTML containing:
    - session start time
    - stop time
    - vehicle’s number
    - total cost of the session that user’s wallet has been charged
- The invoice must be sent asynchronously. Eventual consistency is acceptable, it is OK that a system could crash before the invoice has been sent, in this case once the system is restarted it should collect all sessions that do not have invoice sent and re-attempt to send them again.
- If an asset is is not active then it should not be possible to start a parking session.
- Configuration for the X, Y, Z parameters should be coming from a config file (see architectural requirements).

Task
- Design and implement a domain model meeting requirements outlined above.
- Add JPA persistence mapping for the model.
- Create REST endpoints:
    - 2 endpoints described in PMS API below
    - Endpoint for adding an asset - URL, response codes and validation logic is to be decided by you
- Authentication and authorization aspects were intentionally left out from the APIs, but feel free to add those if you have time.

You need to implement only three 3 endpoints, other operations mentioned in the introduction are out of the scope of this assignment.

Architectural requirements:
- System should be implemented using DDD approach.
- It should be possible to recalculate a parking session cost at any time, without triggering invoice to be sent at the same time.
- It should be possible to change implementation of the following components without causing a crippling effect throughout the system:
    - Price engine
    - Invoice sender
    - Configuration provider (for the aforementioned X, Z, Y parameters)
- When the system is started in test environment it should load a bunch of test assets into the database.

Technical stack:

- JDK 1.8+, Java Time API
- Hibernate 4+ with JPA
- Spring Boot 2+, Spring Data JPA 2+
- H2 DB

Development context and business environment

- The system being developed is not going to be under extremely heavy load, so you can rely on JQL and keep logic in the domain layer if you deem so.
- It is fine to reuse existing abstractions provided by Time API in your model.
- Domain model should be persistence layer agnostic, meaning that one should be able to use it even in unit tests and non-persistence aware environments.
- You are not being pushed by stakeholders to deliver the system overnight compromising and making shortcuts in its design and development procedure, in fact it was you who provided the estimation how much time (still reasonable) it is going to take you to implement the described functionality.

PMS API

Below you will see a part of the PMS API specification that you need to take into account when you start working on your implementation.

<table>
<tr><td colspan="2">Once PMS' camera has read a licence plate number of arrived vehicle it will issue request to a $host in order to figure out if access should be granted or a driver should get a regular paper ticket.</td></tr>
<tr><td>Endpoint</td><td>POST https://$host/pms/v1/assets/$asset/sessions</td></tr>
<tr><td>Body</td><td>{ “licensePlateNumber”: “XXX” }</td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>licensePlateNumber</td><td>License plate number of the arrived vehicle</td></tr>
<tr><td>asset</td><td>ID of an asset where PMS sending this request is installed</td></tr>
</table>
</td></tr>
<tr><td>Responses</td><td>
<table>
<tr><td>Code</td><td>200</td></tr>
<tr><td>Description</td><td>Access to be granted. If this code is received then</td></tr>
</table>
</td></tr>
</table>

	bar/gate to be opened.
Body	Empty body.

Code	Anything beside 2xx
Description	Any other code will result in a paper ticket being printed for the driver.

When a driver is about to leave a parking lot, PMS will read a vehicle's licence plate number and issue this request to have a parking session stopped. If successful response is returned then bar/gate will be opened and a driver can leave the premises.

Endpoint	POST https://$host/pms/v1/assets/$asset/vehicle/$licencePlateNumber/session
Body	{ "status": "stopped" }
Parameters	

licensePlateNumber	License plate number of the arrived vehicle
asset	ID of an asset where PMS sending this request is installed

Responses

Code	200
Description	Session is stopped and bar/gate to be opened.
Body	{ "status": "stopped", "total": 12.34, "startedAt": "XXX" "stoppedAt": "XXX" }

Code	Anything beside 2xx
Description	Parking session could not be stopped. The driver will

	need to get in touch with support.