PREMIER PROPERTY

MANAGEMENT GROUP

AUBG Property Management

COS 315 - Software Engineering Sprint 2 Report

Martin Nestorov ....

Tatiana Triukhova ....

Edor Kacerja ....

Monday, 20. November 2017 16:00

Signatures indicate mutual agreement for the content of the delivered report by the guideliness set by Prof. Galletly.

Index

Process Tasks

The following tasks relate to the scrum process.

1. Product Backlog and Sizes

List of the *remaining* user stories, assigned with *priorities* and estimated sizes in *relative points*.

1.1 As an administrator, I want to be able to inform customers that an offer has been accepted or rejected.
—(Medium priority)—
—5 points—

- Add method to send an email to the user who rented/bought a property
- Connect a user account with the property he/she has bought/rented

1.2 As a customer I want to be able to register in the system so that I can make offers for properties.
—(Low priority)—
—3 points—

- Add method to register users when "make an offer" is clicked
- Add method to register at any time during the visit of the app

1.3 As an administrator, I want to be able to divide properties as "for sale" or "for rent".
—(Low priority)—
—2 points—

- Add filtering functionality to administrator for rented/sold properties

1.4 As an administrator, I want to be able to see the profiles of each register customer – name, address, etc.
—(Low priority)—
—2 points—

- Add full reading privileges to admin

1.5 As an administrator, I want to be able to add extra inform, e.g. "No pets" to individual properties.
—(Low priority)—
—1 point—

- Add full editing rights for admin for properties table

2. Sprint Backlog

List of selected user stories to be implemented in Sprint #2.

2.1 As an administrator I want to be able to add new properties for sale or rent to the catalogue. *—(High priority)*
—
—21 points—

- Design the MySQL database **2pts**
- Create MySQL database **3pts**
- Create user and property tables **2pts**
- Fill it with temporary data to test it **2pt**
- Connect java web app to the DB **4pts**
- Authentication for admin **4pts**
- Add functionality for editing the database by admins **4pts**

2.2 As an administrator, I want to be able to view any offers for a given property.
—(High priority)—
—8 points—

- Create user and offers tables **3pts**
- Fill it with temporary data to test it **2pts**
- Connect java web app to the DB **2pts**

- Grant the authority to read the database by the admin **1pt**

2.3 As a customer I want to be able to browse the catalogue for properties for sale or rent within a certain price range or with a certain number of bedrooms.
—(High priority)—
—21 points—

- Add filters for a price range **8pts**
- Add filter for the number of bedrooms **8pts**
- Restrict access to the DB for normal users **3pts**

2.x Plus any other user stories if time sufficient.
This was an agreement made between the customer and the developers, that if time is left, the dev team will continue to work on more user stories, hence the `x` *in the bullet point.*

2.4 Statement of implementation.

Over the course of Sprint 2, our team managed to go beyong the expected Sprint Backlog. Because last sprint we were marginally behind the actual implementation of the stories, this sprint we managed to fully integrate the following User Stories into the product:

2.4.1 As an administrator I want to be able to add new properties for sale or rent to the catalogue.
—(High priority)—
—21 points—
STATUS : COMPLETED

2.4.2 As an administrator, I want to be able to view any offers for a given property.
—(High priority)—
—8 points—
STATUS : COMPLETED

2.4.3 As a customer I want to be able to browse the catalogue for properties for sale or rent within a certain price range or with a certain number of bedrooms.
—(High priority)—
—21 points—
STATUS : COMPLETED

2.4.4 As an administrator I want to be able to delete sold or rented properties from the catalogue.
—(Medium priority)—
—5 points—
STATUS : COMPLETED

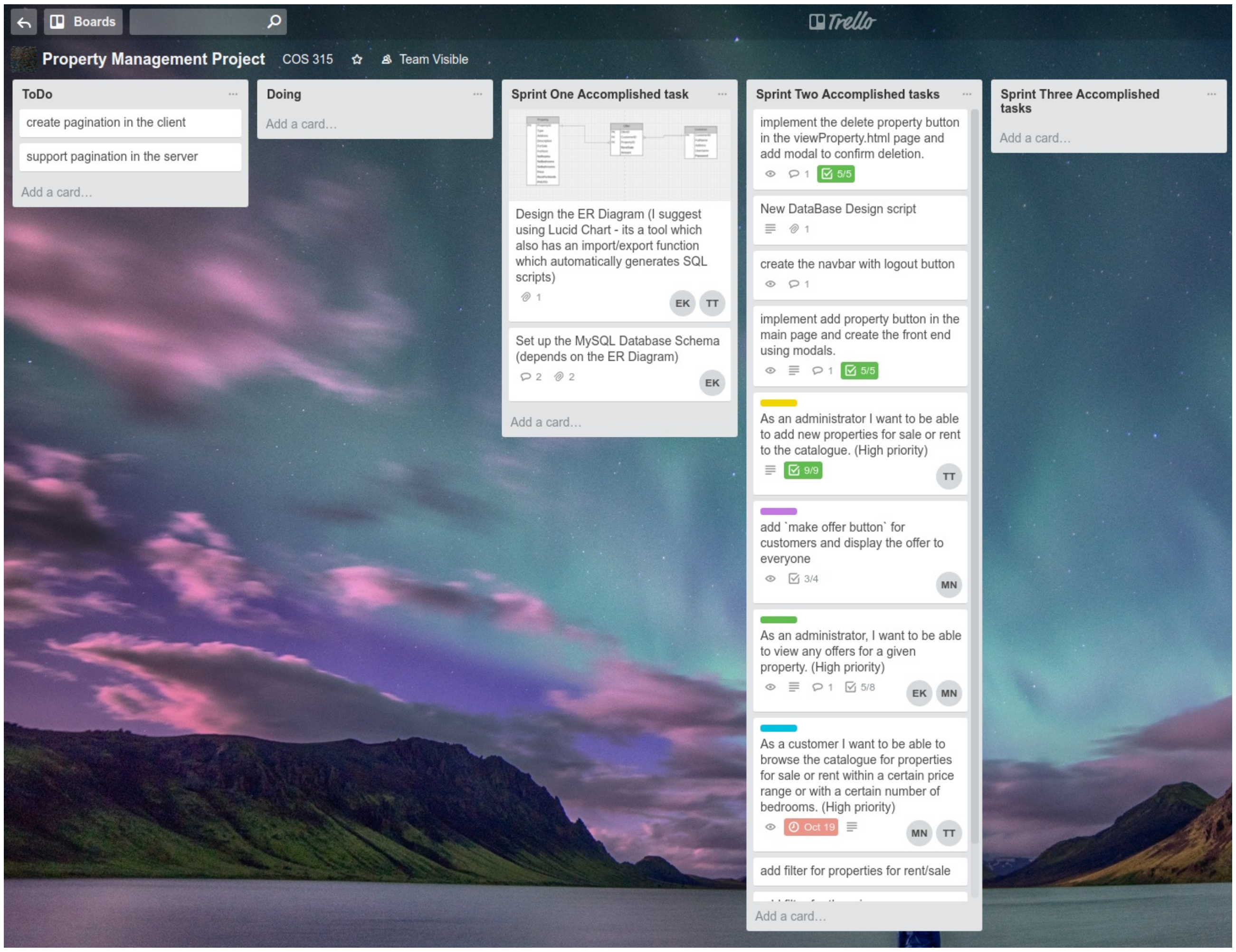
2.4.5 As a customer I want to be able to make an offer for a property.
—(Medium priority)—
—5 points—
STATUS : COMPLETED

NOTE - Because there are **no** user stories left not implemented for this sprint, were are omitting this part of the explanation.

3. Scrum Task Board

Our team uses **Trello** to manage it's Task Board, where we keep track of the current development of the user stories.

This is how our Trello board looks like after the end of Sprint 2.



Boards

Property Management Project COS 315 Team Visible

ToDo
- create pagination in the client
- support pagination in the server
- Add a card…

Doing
- Add a card…

Sprint One Accomplished task
- Design the ER Diagram (I suggest using Lucid Chart - its a tool which also has an import/export function which automatically generates SQL scripts) 1 EK TT
- Set up the MySQL Database Schema (depends on the ER Diagram) 2 2 EK
- Add a card…

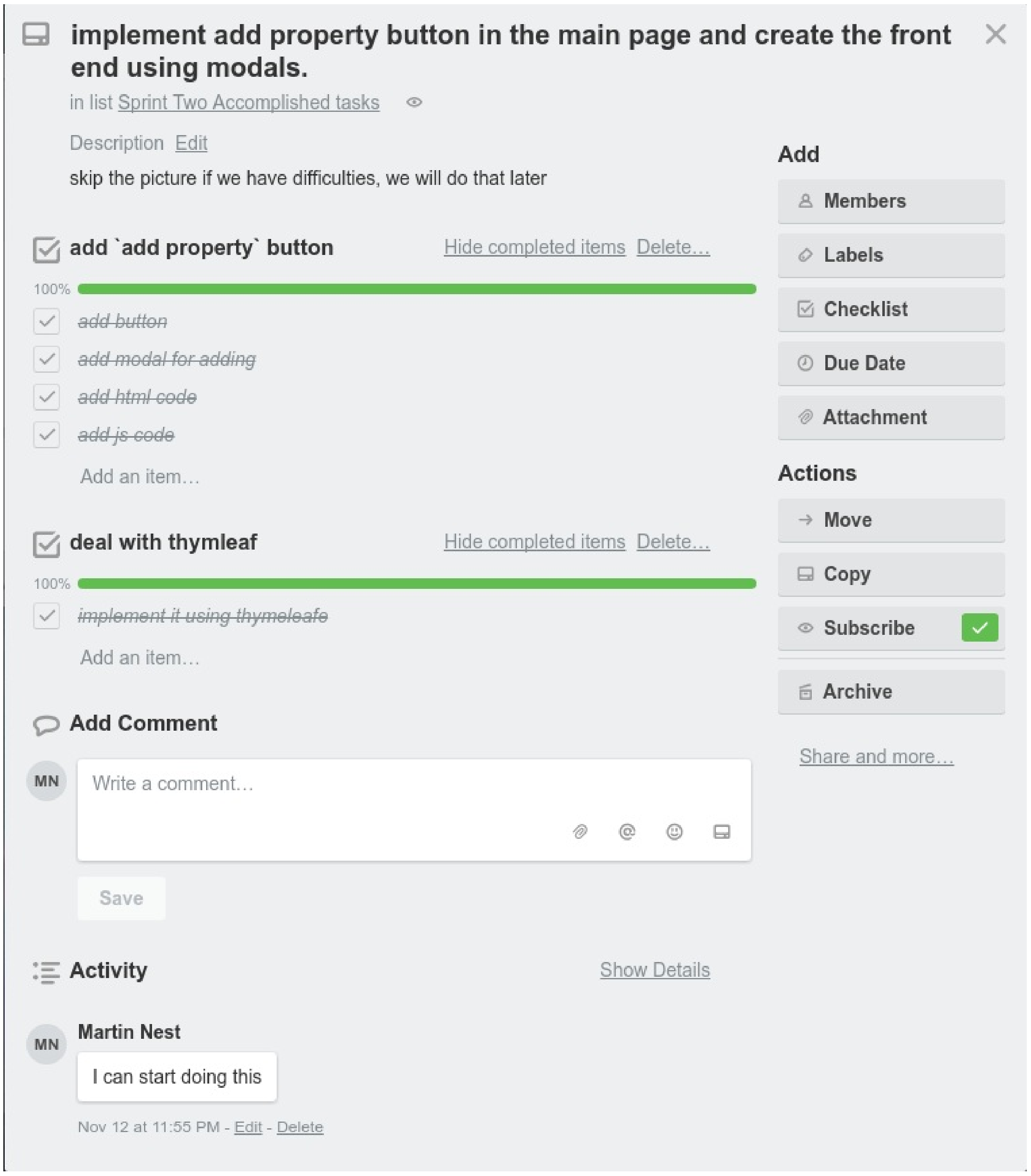
Sprint Two Accomplished tasks
- implement the delete property button in the viewProperty.html page and add modal to confirm deletion. 1 5/5
- New DataBase Design script 1
- create the navbar with logout button 1
- implement add property button in the main page and create the front end using modals. 1 5/5
- As an administrator I want to be able to add new properties for sale or rent to the catalogue. (High priority) 9/9 TT
- add `make offer button` for customers and display the offer to everyone 3/4 MN
- As an administrator, I want to be able to view any offers for a given property. (High priority) 1 5/8 EK MN
- As a customer I want to be able to browse the catalogue for properties for sale or rent within a certain price range or with a certain number of bedrooms. (High priority) Oct 19 MN TT
- add filter for properties for rent/sale
- Add a card…

Sprint Three Accomplished tasks
- Add a card…

NOTE - The above image was unable to be scaled in order for greater readabilty. Apologies for the incovenience.

One by one, going through the major implementations of the sprint:

This screenshot shows how we concentrated on adding new properties to the platform. Here we added buttons and modals in the html code with **Thymeleaf** integration that allow us to access the *Java endpoint controllers*. In the end the new property is saved in the DB and imedaitely displayed on the main page. This was one of the harderst things to do, mainly because **Thymeleaf** is a very picky technology and it has strnage syntax and rules. It's a very constraining technology (not in the good way) and it can take some time to get used to it. In the end it did the job and saved us the hassle to write *JavsScript* code, but it did not make it a breeze to implement. After we were done with this part, *thymeleaf* was much more clear to us and we feel more comforatable to use it later on.


implement add property button in the main page and create the front end using modals.

in list Sprint Two Accomplished tasks

Description Edit

skip the picture if we have difficulties, we will do that later

add `add property` button — Hide completed items Delete…

100%

- [x] ~~add button~~
- [x] ~~add modal for adding~~
- [x] ~~add html code~~
- [x] ~~add js code~~

Add an item…

deal with thymleaf — Hide completed items Delete…

100%

- [x] ~~implement it using thymeleafe~~

Add an item…

Add Comment

Write a comment…

Save

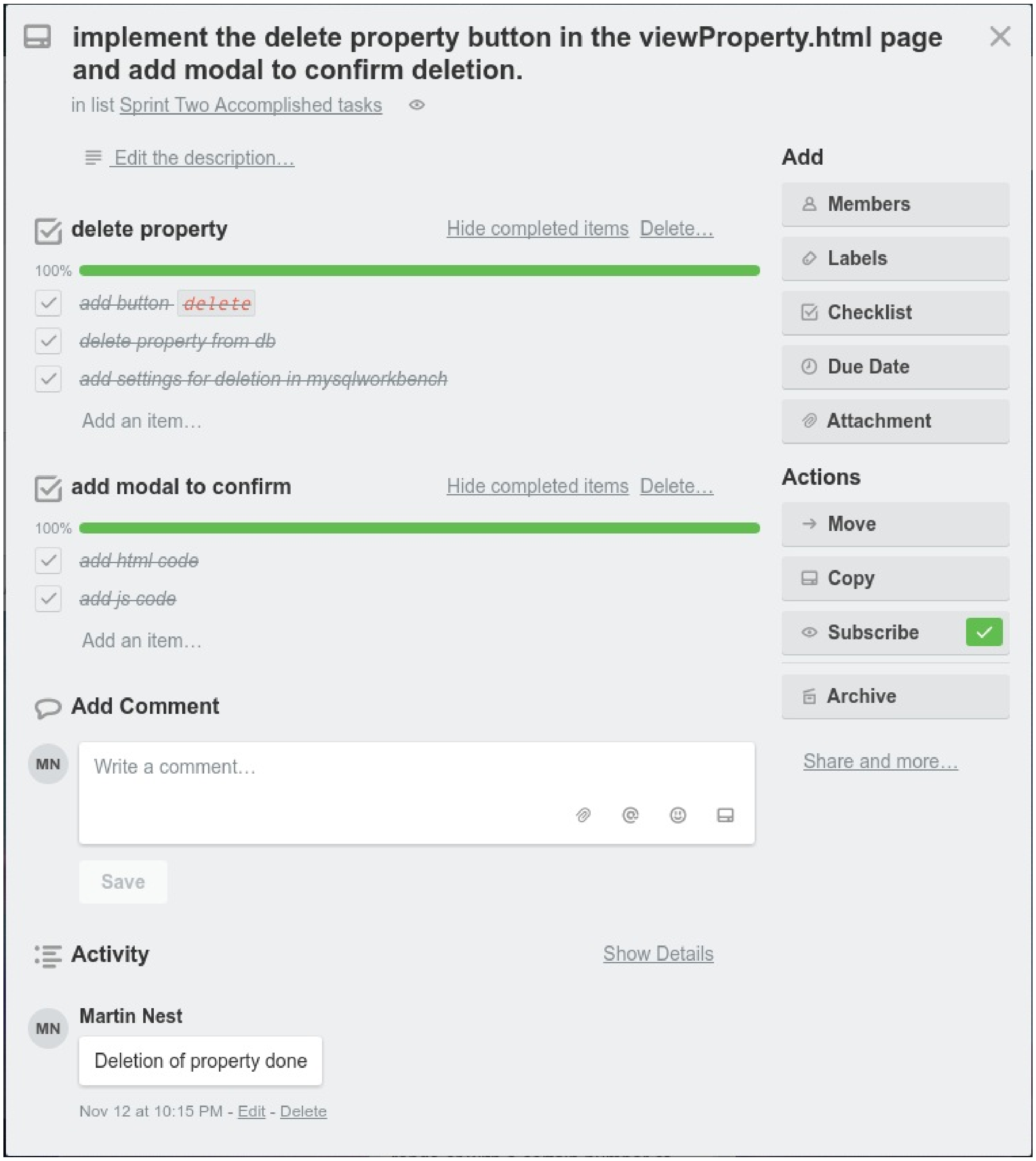
Activity — Show Details

Martin Nest

I can start doing this

Nov 12 at 11:55 PM - Edit - Delete

On this card we can see how concentrated on deleting a property. This was one of the easier tasks to do because the *SQL* script implementation was short and simple. Adding the html button did not cause us any problems at this stage, because autmaticlly the list of properties gets updated.



implement the delete property button in the viewProperty.html page and add modal to confirm deletion.

in list Sprint Two Accomplished tasks

Edit the description…

delete property Hide completed items Delete…

100%

- ✓ ~~add button `delete`~~
- ✓ ~~delete property from db~~
- ✓ ~~add settings for deletion in mysqlworkbench~~

Add an item…

add modal to confirm Hide completed items Delete…

100%

- ✓ ~~add html code~~
- ✓ ~~add js code~~

Add an item…

Add Comment

MN Write a comment…

Save

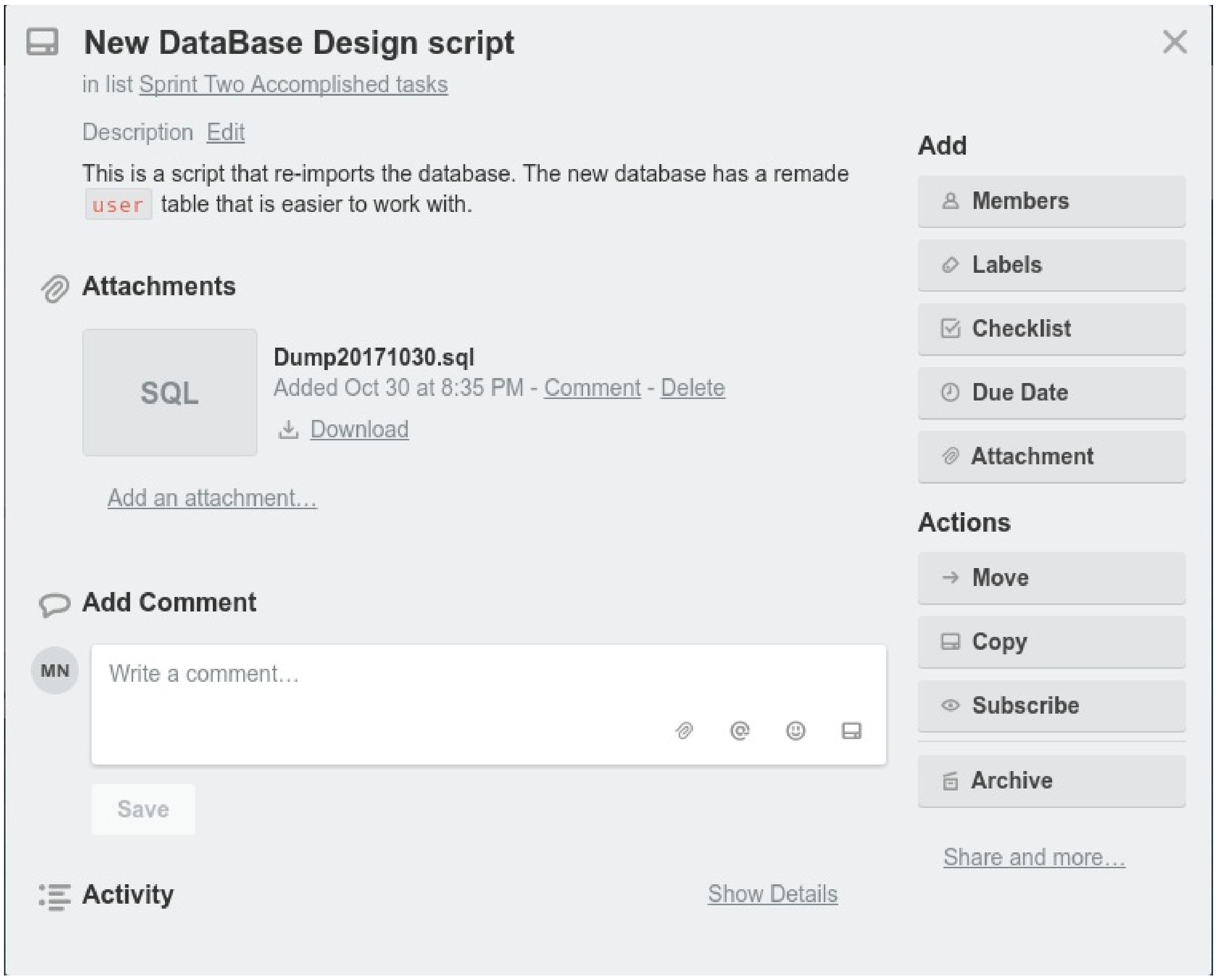
Activity Show Details

MN **Martin Nest**

Deletion of property done

Nov 12 at 10:15 PM - Edit - Delete

One of the tasks we also had to do in order to progress during this sprint is that we hade to think about our database. We hade to redo several of the tuples and tables, because they were not working for us. As far as we know this is a normal practice during development and it was very beneficial to us to make new DB scripts.


New DataBase Design script

in list Sprint Two Accomplished tasks

Description Edit

This is a script that re-imports the database. The new database has a remade `user` table that is easier to work with.

Attachments

Dump20171030.sql
Added Oct 30 at 8:35 PM - Comment - Delete
Download

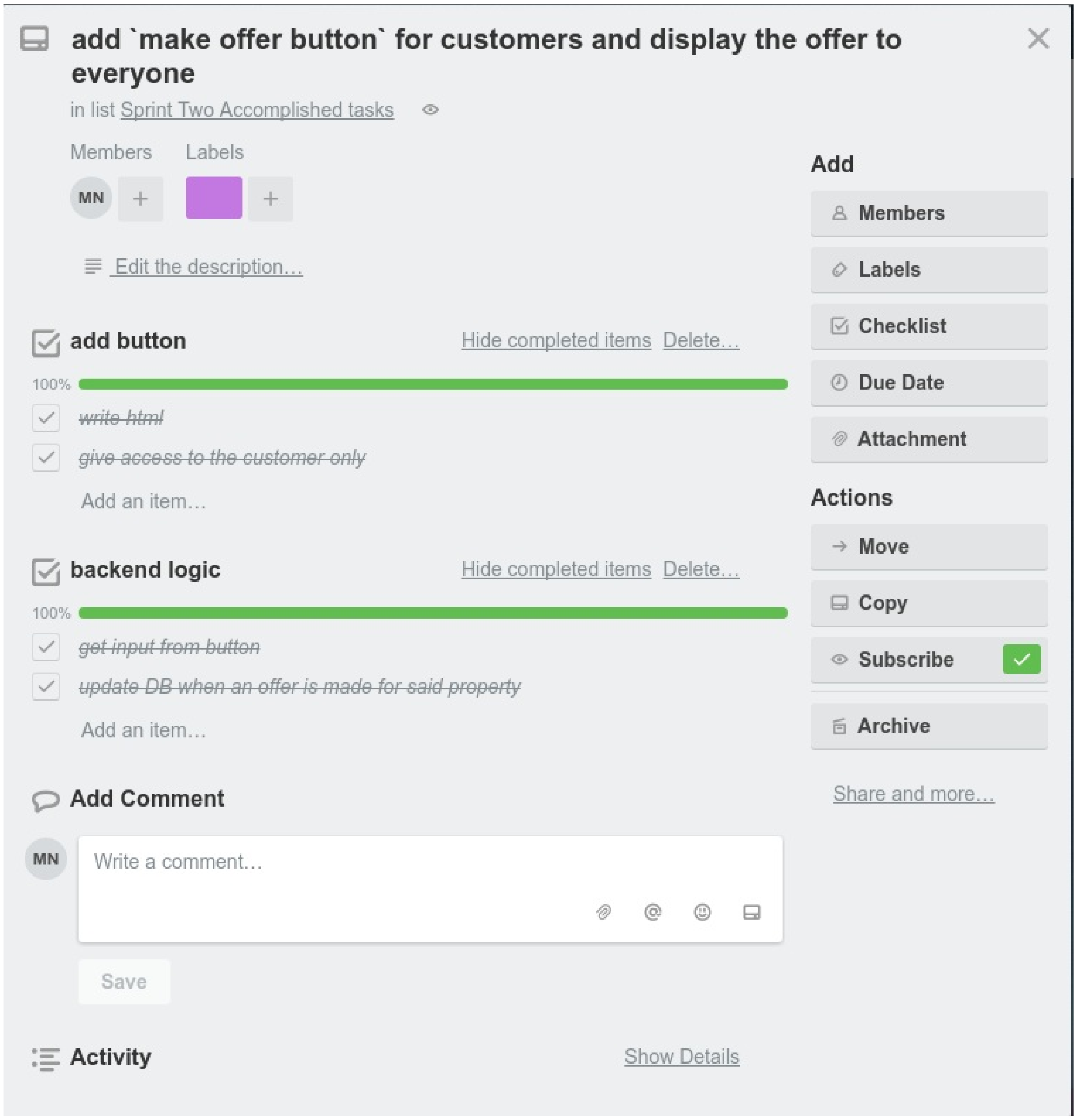
Add an attachment…

Add Comment

Write a comment…

Activity Show Details

At this point we were thinking about how we can make an offer towards a certain property. This turned out easier than expected, mainly because we have already made adding of users and properties, so this was just repeating previous steps. Again the SQL scripts and backend implementation was not difficult, the only hard part is that it takes several hours to make such an implementation.



add `make offer button` for customers and display the offer to everyone

in list Sprint Two Accomplished tasks

Members · MN

Labels

Edit the description…

add button — Hide completed items · Delete…

100%

- ✓ ~~write html~~
- ✓ ~~give access to the customer only~~

Add an item…

backend logic — Hide completed items · Delete…

100%

- ✓ ~~get input from button~~
- ✓ ~~update DB when an offer is made for said property~~

Add an item…

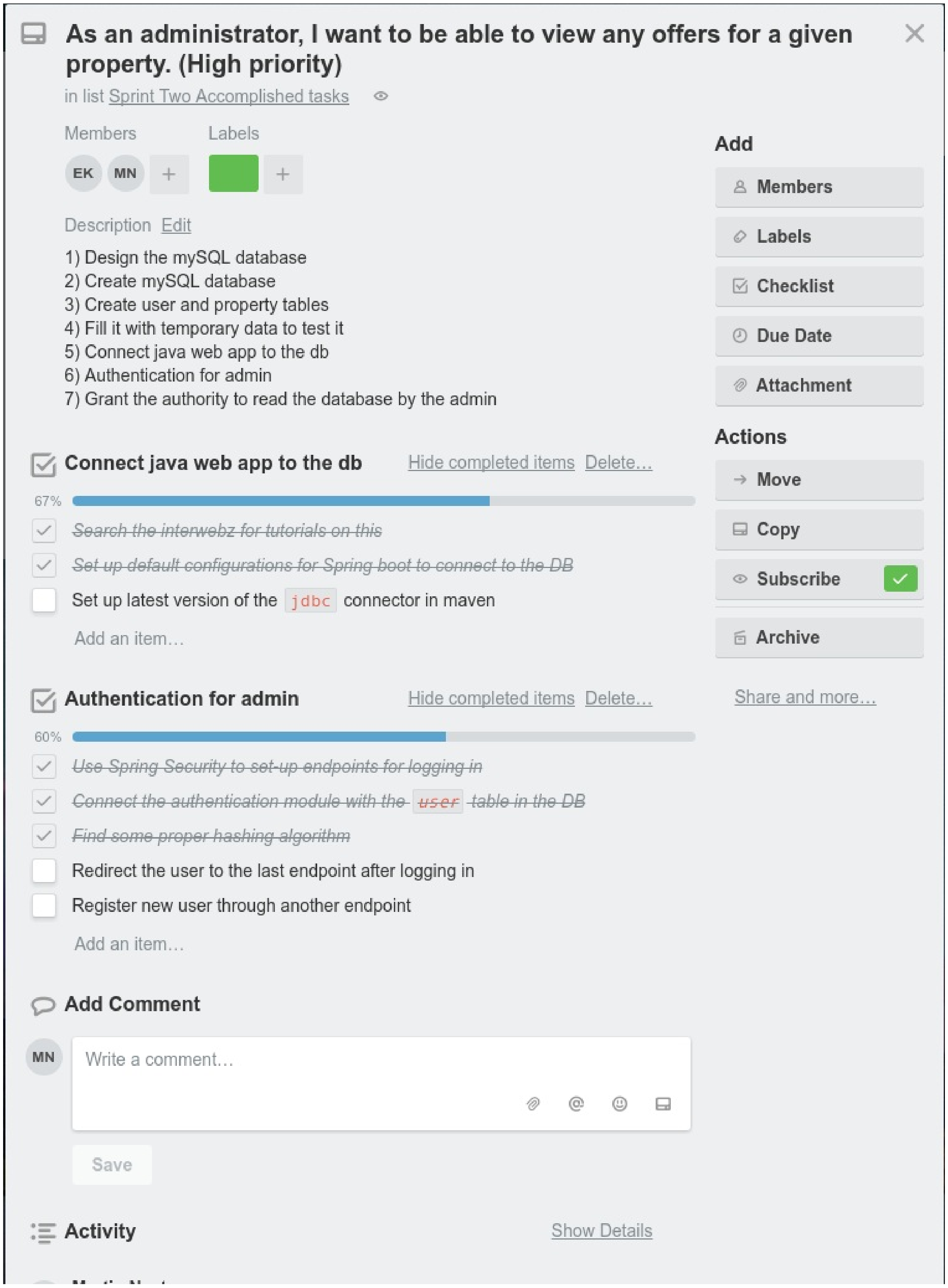
Add Comment

MN · Write a comment…

Save

Activity — Show Details

Following from last step, here we just had to differ a normal *USER* from an *ADMIN*, which was made pretty easy to do with *Spring Security* as it holds the user role during the session, and calls to this role can allows us to give to the user or admin different *read/write* privileges.


As an administrator, I want to be able to view any offers for a given property. (High priority)

in list Sprint Two Accomplished tasks

Members: EK MN

Labels: 🟩

Description Edit

1) Design the mySQL database
2) Create mySQL database
3) Create user and property tables
4) Fill it with temporary data to test it
5) Connect java web app to the db
6) Authentication for admin
7) Grant the authority to read the database by the admin

Connect java web app to the db — Hide completed items Delete…

67%

- ✓ ~~Search the interwebz for tutorials on this~~
- ✓ ~~Set up default configurations for Spring boot to connect to the DB~~
- ☐ Set up latest version of the `jdbc` connector in maven

Add an item…

Authentication for admin — Hide completed items Delete…

60%

- ✓ ~~Use Spring Security to set-up endpoints for logging in~~
- ✓ ~~Connect the authentication module with the `user` table in the DB~~
- ✓ ~~Find some proper hashing algorithm~~
- ☐ Redirect the user to the last endpoint after logging in
- ☐ Register new user through another endpoint

Add an item…

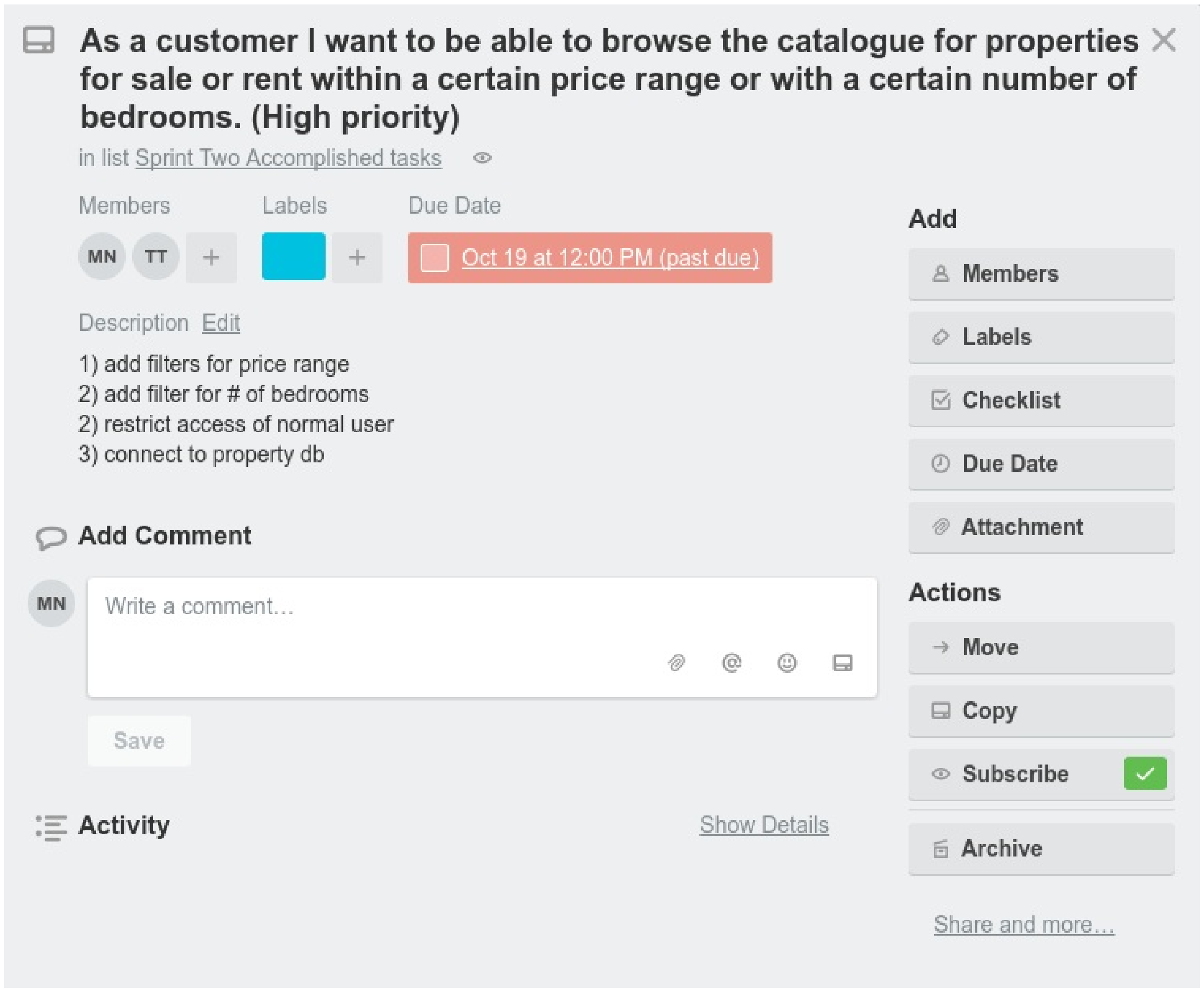
Add Comment

MN Write a comment…

Save

Activity Show Details

And last but not least, filtering was part of the most important features we had to add. This was done using a long integration of *thymeleaf* with the backend since a lot of request had to be made to the server and back. At the end we hade a fill nav bar for filtering working with all of the different property parameters, such as prices, rents, types, locations, room numbers, etc.


As a customer I want to be able to browse the catalogue for properties for sale or rent within a certain price range or with a certain number of bedrooms. (High priority)

in list Sprint Two Accomplished tasks

Members: MN, TT

Due Date: Oct 19 at 12:00 PM (past due)

Description Edit

1) add filters for price range
2) add filter for # of bedrooms
2) restrict access of normal user
3) connect to property db

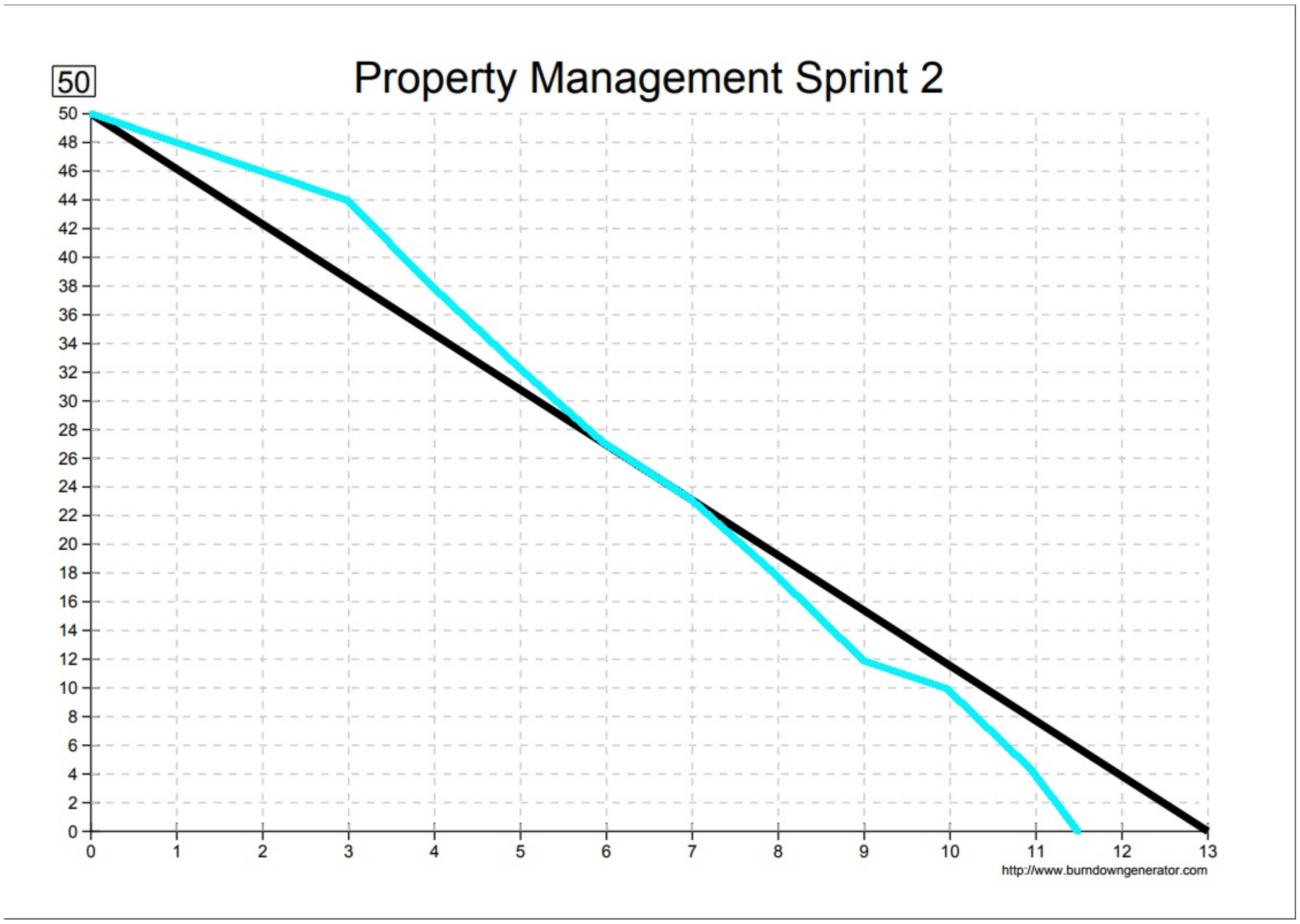
With that we cover the more notible parts of the sprint. Of course more things were added but we consider them either polishings or bug fixes. What we showed here was the meat of the whole Sprint 2.

4. Burn-down Chart

The Burn-down Chart shows the progress over the course of *Sprint #1*. The Chart represents our team velocity, measured in relative points.

$$1 \textit{ relative point} = 1 \textit{ productive day of coding}$$

Our team velocity for *Sprint #2* was **50** points (again due to last sprint lateness). This was calculated by playing Planning Poker and estimating the relative size of the three highest user stories. By adding them up, we estimated that we will manage to do the 50 relative points in our sprint.



**The Burn-down chart was created using the open source website* *[(http://burndowngenerator.com)burndowngenerator.com (http://burndowngenerator.com)]*

This sprints burndown chart is much more different than last one. We can see that this time we are not only very close to our estimation, but we are also surpassing out expectations.

From the start of the sprint we had a little hickup, where we had to redefine some of our understandings for the DB and the way the app should work. This was mainly related to how the UI should look like and how we have to structure the user inputs. But after we cleared that out, we then flooded the Trello board with tasks and started implementing them.

As you can see from the chart, at one point we got even with it, at this time we started getting used to using thymeleaf. After that we got a coding rush and managed to go beyound our expectations and did two more user stories - the making of an offer and the deletion of properties.

One of the highlights here is that we were not held back by the testing phase of our development, mainly because we had a good idea how to test our product.

Product Tasks

1. Testing

For every implemented user story, there is a test plan with a corresponding acceptance test to it and an indication of a pass or fail.

Our tests aim to cover *Functionality*, *Utility*, and *Performance*.

Test 1 — *Editing of DataBase with Admin/User Privileges*

With this we want to evaluate how admins and normal users are able to unteract with the DB. The goal here is to make sure that users do not have the full set of privileges that the admins have. This test is issued every sprint and is tested by the programmers and evaluated by the customer. This test *PASSES*, since users cannot in any way break the security system and mess with the DB.

Results: SUCCESS evaluated by customer.

Test 2 — *Uuser Interface accessibility*

This test covers how accessible the application is, how it allows for easy navigation throught the different menus, how easy it is to register or log in, and how fast information is retrieved from the DataBase. We have a ready user interface that allows for navigation between the different pages of the app. With our UI we try to achive maximal transperancy and not hide any information from the users or admins.

Results: SUCCESS evaluated by customer.

Test 3 — *Making and accpeting offers for a Property*

This test covers the main functionality of our application, where users are able to either buy or rent on specific properties they have selected. By doing so they are able to either imediately access their purchase or be notified if their actions have either been successful or not.

Results: It is too early to evaluate the whole test, since the whole functionality of the app is still under development.

—Acceptance Tests—

These are the tests which proove that the implemented user stories are working and the customer approves of their functionality.
Acceptance tests are either marked with a *PASS/FAIL* tag, indicating the final result after the demo of the tests to the customer.

Feature Test 1: As an administrator I want to be able to add new properties for sale or rent to the catalogue.
—(High priority)—

a. *The administrator can login with valid user id and password. Error if either is invalid.*
PASS
b. *The administrator is then able to navigate to a page that allows him/her to add details of new properties for sale or rent. Error if property already exists.*
PASS
c. *Confirmation message displayed that an update to database has been made.*
PASS
d. *System displays details of newly added property.*
PASS

Feature Test 2: As a customer I want to be able to browse the catalogue for properties for sale or rent within a certain price range or with a certain number of bedrooms.
—(High priority)—

a. *Customer is able to navigate from home page to a page which prompts the user.*

1. For type of property - sale or rent
2. For a price range
3. For a certain number of bedrooms
4. For a location

And then lists the properties with these features. No login required.
Validation of data entered is required, with appropriate error message if invalid.
PASS

Feature Test 3: As an administrator, I want to be able to view any offers for a given property.
—(High priority)—

a. *The admin is able to look at every property and to see what are the offers that are associated with that property.*
PASS

b. *If a property has no exicsting offers to it, then there os nothing dispalyed.*
PASS

Feature Test 4: As an administrator I want to be able to delete sold or rented properties from the catalogue.
—(Medium priority)—

a. *The admin is able to see a delete property button when he/she opens a property page.*
PASS

b. *The delete button prompts for confirmation to the admin before completely deleting the property.*
PASS

c. *The database is automatically updated after this action.*
PASS

Feature Test 5: As a customer I want to be able to make an offer for a property.
—(Medium priority)—

a. *Customers are able to open a property and make an offer to that property.*
PASS

b. *The admin can later see the updated property offers, but the cannot.*
PASS

2. Implementation

Martin Nestorov — For the current Sprint #2 I have worked on mainly the *front end* of the **add a property** user story. Together with this I also had an impact on making an offer to a property.

This was quite a challenge, because for the front end implementation of the app, we used the system `Thymeleaf` and it turned out to be more complcated than expected. This was due to two main reasons - this was both a new technology for us and was also very constraining and we had to follow an odd and strict syntax in the *html* code.

```
<div th:object="${user}" th:if="${user.getRole() == 'ADMIN'}">
    <!-- Button trigger modal -->
    <button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModalNorm">
        Add new Property
    </button>
    <!-- Modal -->
    <div class="modal fade" id="myModalNorm" tabindex="-1" role="dialog"
         aria-labelledby="myModalLabel" aria-hidden="true">
        <div class="modal-dialog">
            <div class="modal-content">
                <!-- Modal Header -->
                <div class="modal-header">
                    <button type="button" class="close"
                            data-dismiss="modal">
                        <span aria-hidden="true">&times;</span>
                        <span class="sr-only">Close</span>
                    </button>
                    <h4 class="modal-title" id="myModalLabel">
                        Add new Property
                    </h4>
                </div>
                <!-- Modal Body -->
                <div class="modal-body">
                    <form action="#" th:action="@{/prop}" th:object="${prop}" method="post">
                        <div class="form-group">
                            <label for="propType">Property type</label>
                            <input type="text" th:name="type" th:filed="*{type}" th:value="${prop.
getType()}"
                                   class="form-control"
                                   id="propType" placeholder="Enter property type"/>
                        </div>
                        <div class="form-group">
                            <label for="address">Address</label>
                            <input type="text" th:name="address" th:field="*{address}"
                                   th:value="${prop.getAddress()}" class="form-control"
                                   id="address" placeholder="Enter address"/>
                        </div>
                        <div class="form-group">
                            <label for="description">Description</label>
                            <input type="text" th:name="description" th:field="*{description}"
                                   th:value="${prop.getDescription()}" class="form-control"
                                   id="description" placeholder="Enter description"/>
                        </div>
                        <div align="left"><br>
                            <label for="sale">Sale
                                <input type="checkbox" th:name="forSale" th:field="*{forSale}"
                                       th:value="${prop.isForSale()}" id="sale"
                                       placeholder="Sale"/><br>
                            </label><br>
                            <label for="rent">Rent
                                <input type="checkbox" th:name="forRent" th:field="*{forRent}"
                                       th:value="${prop.isForRent()}" id="rent"
                                       placeholder="Rent"/>
                            </label><br>
                            <hr>
                        </div>
                        <div class="form-group">
                            <label for="roomsNumber">No. rooms</label>
                            <input type="text" th:name="numberOfRooms" th:field="*{numberOfRooms}"
                                   th:value="${prop.getNumberOfRooms()}" class="form-control"
                                   id="roomsNumber" placeholder="Enter number of rooms"/>
                        </div>
                        <div class="form-group">
                            <label for="bedroomsNumber">No. bedrooms</label>
                            <input type="text" class="form-control"
                                   id="bedroomsNumber" th:name="numberOfBedrooms" th:field="*{numb
erOfBedrooms}"

                                   th:value="${prop.getNumberOfBedrooms()}"
                                   placeholder="Enter number of bedrooms"/>
                        </div>
                        <div class="form-group">
                            <label for="bathroomsNumber">No. bathrooms</label>
                            <input type="text" class="form-control"
                                   id="bathroomsNumber" th:name="numberOfBathrooms" th:field="*{nu
mberOfBathrooms}"

                                   th:value="${prop.getNumberOfBathrooms()}"
                                   placeholder="Enter number of bathrooms"/>
                        </div>
                        <div class="form-group">
                            <label for="price">Price</label>
                            <input type="text" th:name="price" th:field="*{price}" th:value="${pro
p.getPrice()}"

                                   class="form-control"
                                   id="price" placeholder="Enter price"/>
                        </div>
                        <div class="form-group">
                            <label for="rentPerMonth">Rent per month</label>
                            <input type="text" th:name="rentPerMonth" th:field="*{rentPerMonth}"
                                   th:value="${prop.getRentPerMonth()}" class="form-control"
                                   id="rentPerMonth" placeholder="Enter rent per month"/>
                        </div>

                        <button type="submit" class="btn btn-default">Submit</button>
                    </form>
                </div>

                <!-- Modal Footer -->
                <div class="modal-footer">
                    <button type="button" class="btn btn-default"
                            data-dismiss="modal">
                        Cancel
                    </button>
                </div>
            </div>
        </div>
    </div>
</div>
```

This is the input form for the `add property` method. As you can see, this is just a collection of user data, but actually in oder for the whole system to start working, this *html* had to be made into a template. From a back-end perspective, this forced us to do some refactoring to the code base. In the end it worked quite well, but the consequences were such that we had to sped a lot of time reading the documentation of the `Thymeleaf API`.

With this in mind I also made a similar model that sends an offer to a property. The front end behaves the same way as the `add property` model. From a backend perspective this is what the `SQL` scripts and database update looks like:

```
@RequestMapping(value = "/makeOffer")
public String makeOffer(
        @Valid @ModelAttribute(value = "newOffer") Offer newOffer,
        @Valid @ModelAttribute(value = "property") Property property,
        Model model,
        Authentication authentication) {

    LOGGER.info("Making an offer to property " + property.getPropertyId());

    LOGGER.info("The offer is : " + newOffer.getOfferToBuy());

    newOffer.setUserId(userService.getUserByUsername(authentication.getName()).getId());
    newOffer.setOfferId(offerService.getAllOffers().size() + 10);
    newOffer.setPropertyId(property.getPropertyId());

    offerService.addOffer(newOffer);

    List<Property> list = propertyService.getAllProperties();
    model.addAttribute("properties", list);
    model.addAttribute("user", userService.getUserByUsername(authentication.getName()));
    model.addAttribute("prop", new Property());

    return "properties";
}
```

The `jdbcTemplate` makes it easy for use to interact with the DB.

```
@Override
public void updateProperty(int propertyId, double offer) {

    String sql = "UPDATE property SET offer=? WHERE property_id=?";
    jdbcTemplate.update(sql, offer, propertyId);

    LOGGER.info("Updated offer for property " + propertyId);
}
```

Tatiana Triukova — During this sprint I concentrated on working with the *filetring system* for the properties. This was quite a challenge, because `Thymeleaf` proved difficult to learn from the beginning. I also managed to separate the *ADMINS* from the *USERS* so that the app shows different buttons to either one.

Following the implementation of the filters, I had to make a long query towards the Database, in order to actually filter what we needed. This was done manually because `jdbcTemplate` did not have a filtering system integrated in it.

```
    public List<Property> filterProperties(
        String forSale, String forRent, String no_rooms, String price, String no_bedrooms, String
no_bathrooms,
        String type, String address) {

    String sql = "SELECT * FROM property WHERE property_id > -1";
    if (forSale != null) {
        sql = sql + " AND for_sale=" + forSale;
    }
    if (forRent != null) {
        sql = sql + " AND for_rent=" + forRent;
    }
    if (no_rooms != null) {
        sql = sql + " AND no_rooms=" + no_rooms;
    }
    if (price != null) {
        if (price.equals("<30,000")) {
            sql = sql + " AND Price<30000";
        }
        if (price.equals("30,000 - 70,000")) {
            sql = sql + " AND Price BETWEEN 30000 AND 70000";
        }
        if (price.equals(">70,000")) {
            sql = sql + " AND Price>70000";
        }
    }
    if (no_bedrooms != null) {
        sql = sql + " AND no_bedrooms=" + no_bedrooms;
    }
    if (no_bathrooms != null) {
        sql = sql + " AND no_bathrooms=" + no_bathrooms;
    }
    if (type != null) {
        if (type.equals("house")) {
            sql = sql + " AND type = 'house'";
        }
        if (type.equals("apartment")) {
            sql = sql + " AND type = 'apartment'";
        }
    }
    if (address != null) {
        sql = sql + " AND address='" + address + "'";
    }

    RowMapper<Property> rowMapper = new PropertyRowMapper();

    return this.jdbcTemplate.query(sql, rowMapper);

}
```

After this was done, it was a simple listing of the results on the *html* code.

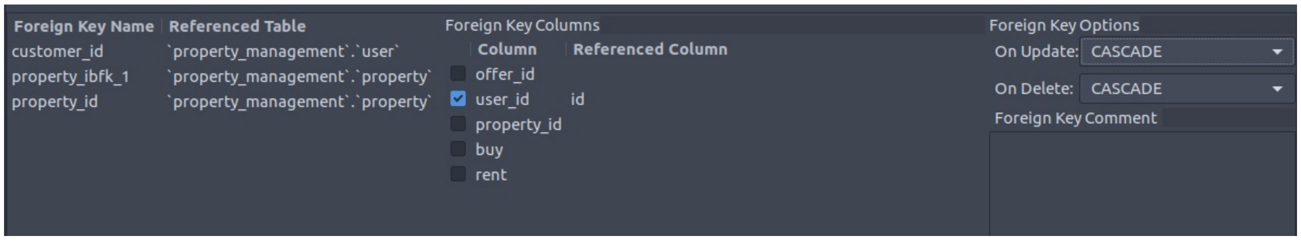
After that I had to deal with how the front-end understood who is an *ADMIN* / *USER*. This was made through `Thymeleaf` only. It was quite easy to do, because in the html code you can make `if` statements and comparisons like so:

```
            <div th:object="${user}" th:if="${user.getRole() == 'ADMIN'}">
                <button class="btn btn-default" data-toggle="modal"
                        data-target="#confirm-delete">
                    Delete
                </button>
            </div>


            <div th:object="${user}" th:if="${user.getRole() == 'USER'}">
                <button class="btn btn-default" data-toggle="modal"
                        data-target="#make-offer">
                    Make Offer
                </button>
            </div>
```

Edor Kacerja — On this sprint I mainly worked on deleting a property and also helping Martin with the back-end to make an offer to a property.

The deletion of a property was tricky for the most part, because we had to think about the DB and it's access levels. Because some of our keys are `foreign keys` and have references to other tables, we hade to tell explicetly that we have to make a `CASCADE DELETION` in oder for a property to be deleted.

```
public String deleteProperty(int propertyId, Model model, Authentication authentication) {

    LOGGER.info("Deleting property by id.");

    propertyService.deleteProperty(propertyId);

    List<Property> list = propertyService.getAllProperties();
    model.addAttribute("properties", list);
    model.addAttribute("user", userService.getUserByUsername(authentication.getName()));
    model.addAttribute("prop", new Property());

    return "properties";
}

    @Override
public void deleteProperty(int propertyId) {

    String sql = "DELETE FROM property WHERE property_id=?";
    jdbcTemplate.update(sql, propertyId);

    LOGGER.info("Deleted property from DB");
}
```



Foreign Key Name	Referenced Table
customer_id	\`property_management\`.\`user\`
property_ibfk_1	\`property_management\`.\`property\`
property_id	\`property_management\`.\`property\`

Foreign Key Columns

	Column	Referenced Column
☐	offer_id	
☑	user_id	id
☐	property_id	
☐	buy	
☐	rent	

Foreign Key Options
On Update: CASCADE
On Delete: CASCADE
Foreign Key Comment

Apart from this, in oder for us to make an offer, we hade to implement a special class `Offer` that interacts on its separate level with the app and DB. Here I implemented the several `Interfaces` and classes in Java, where we had to also make a special table in the DB for the `offers`. This was not hard, because we already knew how to make such classes based on previous experiences with the `Property` and `User` classes.

package com.property.manager.models;

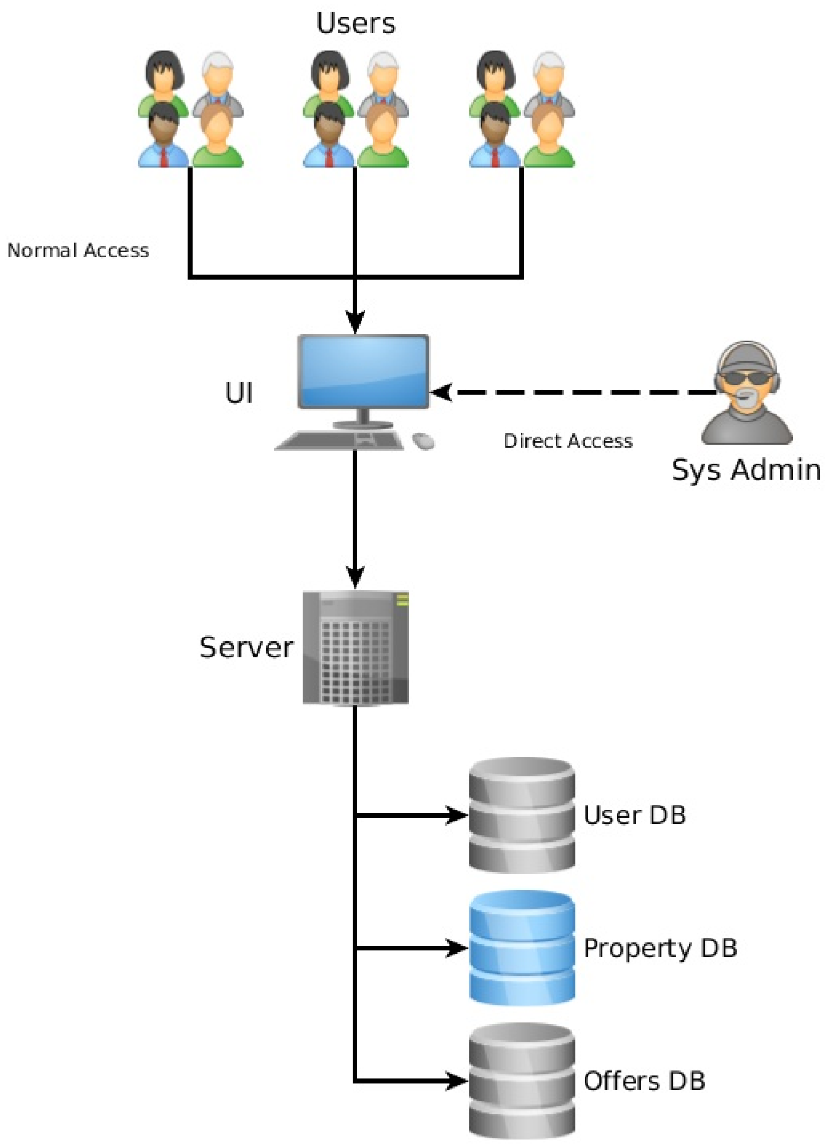
```
public class Offer {

private int userId;
private int offerId;
private int propertyId;
private double offerToBuy;
private double offerToRent;

public Offer() {

}

public Offer(int userId, int offerId, int propertyId, double offerToBuy, double offerToRent) {

    this.userId = userId;
    this.offerId = offerId;
    this.offerToBuy = offerToBuy;
    this.offerToRent = offerToRent;
    this.propertyId = propertyId;
}

public int getUserId() {

    return userId;
}

public void setUserId(int userId) {

    this.userId = userId;
}

public int getOfferId() {

    return offerId;
}

public void setOfferId(int offerId) {

    this.offerId = offerId;
}

public int getPropertyId() {

    return propertyId;
}

public void setPropertyId(int propertyId) {

    this.propertyId = propertyId;
}
public double getOfferToBuy() {

    return offerToBuy;
}

public void setOfferToBuy(double offerToBuy) {

    this.offerToBuy = offerToBuy;
}

public double getOfferToRent() {

    return offerToRent;
}

public void setOfferToRent(double offerToRent) {

    this.offerToRent = offerToRent;
}
}
```

3. Code Quality

Our team relies on a unified coding standard typical for the *Java* and *Spring* Framework. All of us are using the *Camel Case* standard, our code is supplied with JavaDoc comment blocks, and we have separated our classes in the *MVC* standard into the Model, View, and Controller sections.

4. Bonus Diagram

In the following diagram, we can see what the general idea of the operation of the application is.



The users and the admins are both going to access the same user interface that is developed, with the difference being that the admins will have several extra buttons, allowing for more actions. Through the UI, we are able to connect to the server, which is then connected to our DataBase.