Add dynamic pages

Skills Network

Estimated time needed: 2 hour

In the previous module, you created the necessary backend services to manage dealerships, reviews, and cars.
It's time to create user-friendly and aesthetic frontend pages to present these services to the end users.

Environment setup

If your Theia workspace has been reset and you want to continue from what you have done previously, you can `git clone` or pull the latest code from your GitHub repository.

- Set up the Python runtime if your Theia workspace has been reset.

```
1
```

```
python3 -m pip install -U -r requirements.txt
```

Copied!

Note that you may need to perform models migrations for a new Theia environment.

Create a dealership Bootstrap table

In the `get_dealerships` view method you created in the previous lab, we simply returned the dealer names as a simple string in a HTTPResponse.

In fact, a dealer object has multiple attributes and it would be better to be present it in a table with each row representing a dealer and each column representing one attribute.

First, we need to update the `get_dealerships` view to render a Django template to present dealers in a Bootstrap table.

- Open `djangoapp/views.py`, find `get_dealerships(request):` view method and create an empty `context` dictionary.
- Add the `dealerships` list returned by `get_dealers_from_cf` method to the `context`.
- Update the return statement to use `render(request, 'djangoapp/index.html', context)`.

Now we can update `index.html` to display the dealership list appended in the context.

- Open `templates/djangoapp/index.html`, create a Bootstrap table `<table>` under `<nav>` bar.
- For the table head `<thead>`, add a table row `<tr>` with following table header cells `<th>` represents the dealer attribute names:
    - `ID`
    - `Dealer Name`
    - `City`
    - `Address`
    - `Zip`
    - `State`
- For the table body `<tbody>`, add a `<tr>` for each dealer object in `dealership_list`.
- An example table code snippet would look like the following:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

```
<table class="table" id="table">
        <thead>
            <tr>
                <th data-field="id">ID</th>
                ...
            </tr>
        </thead>
        <tbody>
            {% for dealer in dealership_list %}
                <tr>
                    <td>{{dealer.id}}</td>
                    ...
                </tr>
            {% endfor %}
        </tbody>
    </table>
```

Copied!

- For the dealer name table cell, add a link pointing to `dealer_details` view
`<td><a href="{% url 'djangoapp:dealer_details' dealer.id %}">{{dealer.full_name}}</a></td>`.

As such, users can click a dealer name to drill down to the dealer details page with reviews.

A finished dealership table may look like the following:

ID	Dealer Name	City	Address	Zip	State
1	Holdlamis Car Dealership	El Paso	3 Nova Court	88563	TX
2	Temp Car Dealership	Minneapolis	6337 Butternut Crossing	55402	MN
3	Sub-Ex Car Dealership	Birmingham	9477 Twin Pines Center	35285	AL
4	Solarbreeze Car Dealership	Dallas	85800 Hazelcrest Circle	75241	TX
5	Regrant Car Dealership	Baltimore	93 Golf Course Pass	21203	MD
6	Stronghold Car Dealership	Wilkes Barre	2 Burrows Hill	18763	PA
7	Job Car Dealership	Pueblo	9 Cambridge Park	81010	CO
8	Bytecard Car Dealership	Topeka	288 Larry Place	66642	KS
9	Job Car Dealership	Dallas	253 Hanson Junction	75216	TX
10	Alphazap Car Dealership	Washington	108 Memorial Pass	20005	DC
11	Rank Car Dealership	Carol Stream	8108 Dryden Court	60351	IL

Take a screenshot for peer-review:

After you have created the dealerships page, please take a screenshot of your completed dealership list page and name it as `dealerships.jpg` or `dealerships.png` for peer-review.

- Next, let's try to add a filter to the state column so that user can filter dealers by state.
- Update the Bootstrap `<table>` element by adding a `data-filter-control="true"` option to enable filtering.

Now the `<table>` element becomes `<table class="table" id="table" data-filter-control="true">`

- For the `state` head cell in `<thead>`, add a `data-filter-control="select"` option so it becomes `<th data-field="state" data-filter-control="select">State</th>`. This will add a dropdown filter to `state` column.
- Add a JavaScript snippet to turn-on filter control for the `table` element.

```
1
2
3
4
5
6
7
8
9
10
```

```
...
</body>

<script>
  $(function() {
    $('#table').bootstrapTable()
  })
</script>

</html>
```

Copied!

- Now user should be able to filter the dealer list by the `state` column, like the following screenshot:

You will find more detailed references about Bootstrap table with a filter near the end of this lab.

Take a screenshot for peer-review:

After you have added a dealerships filter, please take a screenshot with the filter dropdown list open and name it as `dealerships_filter.jpg` or `dealerships_filter.png` for peer-review.

Create a dealer details/reviews page

By now, you should have finished the dealers table on the index page.

When a user clicks a dealer name on the table, a detailed dealer page should be rendered to show all reviews for the dealer.

Let's start making the dealer details page now.

First, we need to update `get_dealer_details` view to return a list of reviews for a specific dealer.

- Open `djangoapp/views.py` file and find `get_dealer_details` view method you created before.
- Create an empty `context` dictionary and append the reviews list from `get_dealer_reviews_from_cf` method to context.
- Update the return statement to use `render(request, 'djangoapp/dealer_details.html', context)`.

Next, let's display each review as a Bootstrap card on the `dealer_details.html` page.

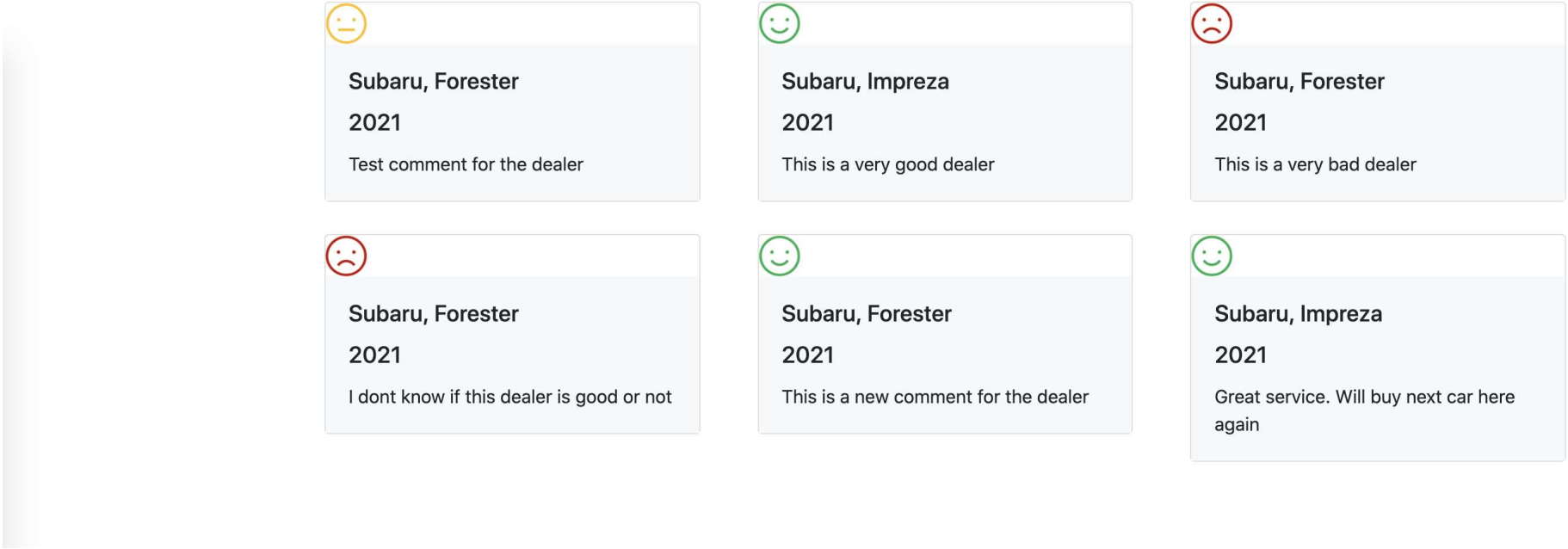
- Open `dealer_details.html`, copy the `<nav>` bar in the `index.html`.
  Then under `{% if user.is_authenticated %}`, add a link pointing to `add_review` view (GET request).

An authenticated user should be able to click on this link and add a review for the dealer. We will create a review submission page and update the `add_review` view later.

- In the `dealer_details.html`, add a `<div class="card-columns">` element to organize review cards.
- For each review in reviews list, create a `<div class="card">` with the following child elements:
  - A `<img class="card-img-left">` to visualize the sentiment using three provided emoji images in `static/media/emoji` folder.
    For example, if the review sentiment is positive, set the `src="{{MEDIA_URL}}/emoji/positive.png"`
  - A `<div class="card-body">` with several `card-title` text labels to show the car model name, make, and purchase year. In addition, add another `card-text` to show review content
- You could use the `add_review` view method created in previous lab to manually post some mockup reviews for a dealer.

The completed dealer detail page may look like the following screenshot:

Reviews for **Sub-Ex Car Dealership**



Take a screenshot for peer-review:

After you have created your dealerships details page, please take a screenshot and name it as `dealership_details.jpg` or `dealership_details.png` for peer-review.

Create a review submission page

Next, let's create a real review submission page to allow user create a review for a dealer. This page works similar to the signup page you created before.

- Open `templates/djangoapp/add_review.html` and add a `<form>` with action pointing to `djangoapp:add_review` (POST request).

Note that you need to add the dealer id as URL parameter here. For example, `action="{% url 'djangoapp:add_review' dealer_id%}"`.
The `dealer_id` could be sent back within the context or you may call get dealer cloud function to get the dealer object and append it into context.

- Add the following child elements to the `<form>`:
  - A `<textarea class="form-control" id="content" name="content" rows="2" required></textarea>` to receive review content.
  - A `<input class="form-check-input" type="checkbox" name="purchasecheck" id="purchasecheck">` to ask if users purchased car from this dealer before or not.
  - A Bootstrap dropdown `<select name="car" id="car" class="form-select" required>` to ask user select a car owned by this dealer. (which you created using Django admin site).

  Each select `<option>` represents a car with make and produce year information. For example:

```
1
2
3
4
5
<select name="car" id="car" class="form-select" required>
    {% for car in cars %}
        <option selected value={{car.id}}>{{car.name}}-{{car.make.name}}-{{ car.year|date:"Y" }}</option>
    {% endfor %}
</select>
```

Copied!

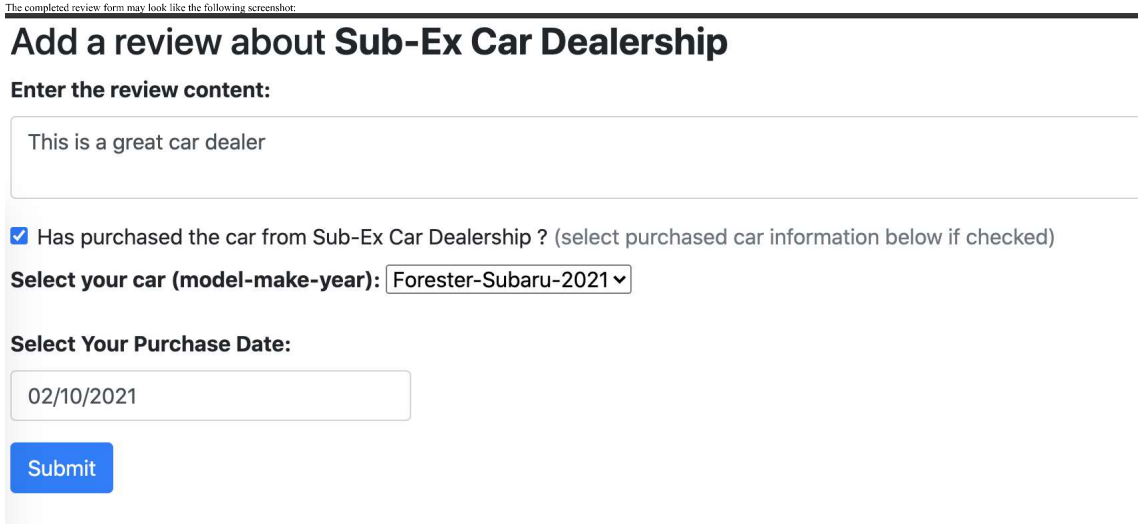
  - A date input `<input class="date-own form-control" type="text" name="purchasedate" id="purchasedate">` to ask users pick a purchase date. You may also add the following script to define the purchase date format.

```
1
2
3
4
5
<script type="text/javascript">
    $('.date-own').datepicker({
        format: 'mm/dd/yyyy'
    });
</script>
```

Copied!

  - A Submit button to post the form data to `add_review` view.

The completed review form may look like the following screenshot:



You can find some detailed references about Bootstrap forms near the end of this lab.

Next, you will need to update `add_review` view to handle both `GET` and `POST` request.

- Open `djangoapp/views.py`, find `add_review` view method.
- When `request.method ==GET` , first query the cars with the dealer id to be reviewed. The queried cars will be used in the `<select>` dropdown.
  Then append the queried cars into context and call render method to render `add_review.html`.
- When `request.method ==POST` , you need to update the `json_payload["review"]` to use the actual values obtained from the review form.
  - For review time, you may use some Python datetime formatting method such as `datetime.utcnow().isoformat()` to convert it into ISO format to be consistent with the format in Cloudant.
  - For purchase, you may use `car.year.strftime("%Y")` to only get the year from the date field.
- Update return statement to redirect user to the dealer details page once the review post is done for example.
  `redirect("djangoapp:dealer_details", dealer_id=dealer_id)`

Now test your templates and updated views to make sure you can add a review for a dealer and see the created review on the dealer details page.

Take a screenshot for peer-review:

After you have created your dealerships review submission page, please take a screenshot and name it as `dealership_review_submission.jpg` or `dealership_review_submission.png` for peer-review.

External References

- Table Filter Control
- Bootstrap Cards
- Bootstrap Forms

Summary

In this lab, you have created dealer list and dealer details. You have added dealer review Django templates and updated corresponding views to append service results into the context object and render the dynamic pages.

At this point, you have completed the main app development work. Next, you just need to containerize the app and deploy it on Kubernetes.

Author(s)

Yan Luo

Upkar Lidder

Other Contributor(s)

Lavanya

Priya

Changelog

Date	Version	Changed by	Change Description
2022-09-20	1.1	K Sundararajan	Updated pip (packages installation) command
2021-02-23	1.0	Yan Luo	Completed the initial version
2021-01-28	1.0	Upkar Lidder	Created new instructions for Capstone project