

## Add dynamic pages



Estimated time needed: 2 hours

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

## Environment setup

If your Thelia 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 of your Thelia workspace has been reset.
- ```
1. 1
2. 1. getEnv() -> pip install -d -r requirements.txt
```

[Copy](#)

Note that you may need to perform models migrations for a new Thelia 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/dealers.html", context)`.

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

- Open `templates/djangoapp/dealers.html`, create a Bootstrap table `dealers` under `cars` but.
- For the table head `<thead>`, add a table row `<tr>` with following table header cells `<th>` represents the dealer attribute names:

```
1. <th> ID
2. <th> Dealer Name
3. <th> City
4. <th> Address
5. <th> Zip
6. <th> State
```

- For the table body `<tbody>`, add a `<tr>` for each dealer object in `dealerships_list`.

- An example table code snippet would look like the following:

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

```
1. <table class="table" id="table">
2.   <thead>
3.     <tr>
4.       <th data-filter="id">ID</th>
5.     </tr>
6.   </thead>
7.   <tbody>
8.     <tr>
9.       <td data-cs="6" data-kind="parent">
10.        <div for dealer in dealerships_list %>
11.          <tr>
12.            <td data-cs="6" data-kind="parent">
13.              <div for dealer in dealerships_list %>
14.                <div for dealer in dealerships_list %>
15.                  <div for dealer in dealerships_list %>
16.                    <div for dealer in dealerships_list %>
```

[Copy](#)

- For the dealer name table cell, add a link pointing to `dealer_details` view `<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  | <a href="#">Holdiamis Car Dealership</a>   | El Paso      | 3 Nova Court            | 88563 | TX    |
| 2  | <a href="#">Temp Car Dealership</a>        | Minneapolis  | 6337 Butternut Crossing | 55402 | MN    |
| 3  | <a href="#">Sub-Ex Car Dealership</a>      | Birmingham   | 9477 Twin Pines Center  | 35285 | AL    |
| 4  | <a href="#">Solarbreeze Car Dealership</a> | Dallas       | 85800 Hazelcrest Circle | 75241 | TX    |
| 5  | <a href="#">Regrant Car Dealership</a>     | Baltimore    | 93 Golf Course Pass     | 21203 | MD    |
| 6  | <a href="#">Stronghold Car Dealership</a>  | Wilkes Barre | 2 Burrows Hill          | 18763 | PA    |
| 7  | <a href="#">Job Car Dealership</a>         | Pueblo       | 9 Cambridge Park        | 81010 | CO    |
| 8  | <a href="#">Bytecard Car Dealership</a>    | Topeka       | 288 Larry Place         | 66642 | KS    |
| 9  | <a href="#">Job Car Dealership</a>         | Dallas       | 253 Hanson Junction     | 75216 | TX    |
| 10 | <a href="#">Alphazap Car Dealership</a>    | Washington   | 108 Memorial Pass       | 20005 | DC    |
| 11 | <a href="#">Rank Car Dealership</a>        | 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_list.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="state" data-cs="6" data-kind="parent" data-bbox="61 650 253 655" data-kind="parent">` option to enable filtering.

Now the `<table>` element becomes `<table class="table" id="table" data-filter="state" data-cs="6" data-kind="parent" data-bbox="61 656 234 661" data-kind="parent">`

- For the state head cell in `<thead>`, add a `data-filter="state" data-cs="6" data-kind="parent" data-bbox="61 673 274 678" data-kind="parent">` option so it becomes `<th data-filter="state" data-cs="6" data-kind="parent" data-bbox="61 678 274 683" data-kind="parent">`. This will add a dropdown filter to state column.
- Add a JavaScript snippet to turn-on filter control for the table element.

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

[Copy](#)

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

|                       |       |  | State |
|-----------------------|-------|--|-------|
| Address               | Zip   |  | IA    |
| 21425 Bartelt Pass    | 50936 |  | IA    |
| 627 Cottonwood Circle | 50335 |  | IA    |

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.png` 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 `cars` bar in the `index.html`.
- Then under `<div id="cars">`, add a link pointing to `dealer_reviews` 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 `dealer_reviews` 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 `div class="card-img-top"` 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/001/1/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



Subaru, Forester

2021


Test comment for the dealer



Subaru, Forester

2021


I dont know if this dealer is good or not



Subaru, Impreza

2021


This is a very good dealer



Subaru, Forester

2021


This is a new comment for the dealer



Subaru, Forester

2021

This is a very bad dealer



Subaru, Impreza

2021

Great service. Will buy next car here again

### Take a screenshot for peer-review:

After you have created your dealership details page, please take a screenshot and name it `as-dealer-shop_details.jpg` or `dealer-shop_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/etjagquery/add_review.html` and add a `ctown` with action pointing to `etjagquery:add_review` (POST request).

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

- Add the following child elements to the `ctown`:

```
<div class="form-control" id="content" name="content" class="form-control" required="required">
</div>
<div class="form-check" id="car" name="car" class="form-check" required="required">
</div>
<div class="form-check" id="year" name="year" class="form-check" required="required">
</div>
```

Each select options represents a car with make and produce year information. For example:

```
<div class="form-control" id="car" name="car" class="form-control" required="required">
</div>
<div class="form-control" id="year" name="year" class="form-control" required="required">
</div>
<div class="form-control" id="content" name="content" class="form-control" required="required">
</div>
```

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

The completed review form may look like the following screenshot:

## Add a review about Sub-Ex Car Dealership

### Enter the review content:

This is a great car dealer

☒ Has purchased the car from Sub-Ex Car Dealership ? (select purchased car information below if checked)

Select your car (model-make-year): Forester-Subaru-2021

### Select Your Purchase Date:

02/10/2021

Submit

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 `etjagquery/views.py`, find `add_review` view method.
- When request method is `GET`, first query the cars with the dealer id to be reviewed. The queried cars will be used in the context's dropdown. Then append the queried cars into context and call render method to render `add_review.html`.
- When request method is `POST`, you need to update the `form_data['review']` to use the actual value obtained from the review form.
  - For review time, you may use some Python datetime formatting method such as `datetime.datetime.strptime()` to convert it into ISO format to be consistent with the format in `Client`.
  - For purchase, you may use `car_year.strptime("%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:

```
return redirect('etjagquery: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 dealership review submission page, please take a screenshot and name it `as-review-submit_details.jpg` or `review-submit_details.png` for peer-review.

### External References

- [Add a Review](#)
- [Bootstrap Forms](#)
- [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 Lakkar

Other Contributor(s)

Laranya

Piya

### Changelog

Date	Version	Changed by	Change Description
2022-08-28 1.1		K. Sundararajan	Updated pip (packages installation) command
2021-02-23 1.0		Yan Luo	Completed the initial version
2021-02-23 1.0		Upkar Lakkar	Created new instructions for Capstone project

© IBM Corporation 2021. All rights reserved.