2. Answer the following questions in a separate word processor document, regarding the applications/projects you made in Question 1, as well as the theories more generally as learnt throughout the workshops:

a. [5 marks] Regardless of the approach you took, list the components that you have built in the frontend and indicate whether they could be implemented as functional components (rather than class-based components). If this is not possible, explain why the component cannot be a functional component (this may relate to a parent or child component).

🡪 I create 5 components in my assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Type** | **Could use functional component?** | **If not, why?** |
| NavbarComponent | Functional Component | Yes |  |
| Shop | Class Component | Yes |  |
| Cart | Class Component | Yes |  |
| Orders | Class Component | No | “Orders” is the parent component of “Order” and needs to maintain the lifecycle, state and callback function. |
| Order | Class Component | Yes |  |

b. [5 marks] Compare and contrast the two approaches to web application development which were discussed in this unit: the ‘traditional’ (Django-style) approach and single-page view or progressive web applications (Express.js and React.js). Provide at least three examples to support your argument.

🡪 The Django-style traditional approach advantage is that it provides relatively complete database integration and security permission management, but its web page generation performance is relatively poor.

1. Django is very easy to integrate ORM databases and provide data migration. We can define database config in settings.py, define data class in models.py and migrate for the data model changed. But in Express.js, we need an outside database to save data, like MongoDB.

2. Django integrates many middlewares to handle security, authentication and CSRF. However, progressive web applications are not integrated.

3. Django integrates the admin interface, easy to maintain roles and users, and data in the database. However, progressive web applications are not integrated.

4. In webpage generation, the traditional approach generates the full web page and sends it to the client browser. However, in progressive web applications, the data will be sent to the client side using REST API, which generates different components and composes them in one view. After the first loading is complete, the page will be generated/updated very efficiently.

c. [5 marks] Do you think it would be appropriate to use the Django admin interface for users to create, update, and delete data on a Django web application? Support your argument with at least two reasons.

🡪 No, the admin interface is very useful for administrators to manage users and data, but it is inappropriate for the end user.

1. The user experience is very bad. Just like the JustTweet app, if we request an end user to post a new tweet or delete one old tweet through the admin interface, it will be difficult for end users to accept.

2. The admin page has a default template, which is difficult to customise. Nowadays, web applications are developing very fast, and there are many different UI/UX design principles, such as material design. We can’t customise the admin UI as we designed it.

d. [5 marks] Why should the response to a POST request redirect the user to another web page, rather than serve the web page directly back to the user? (Hint: consider other types of REST requests).

🡪 There are 4 basic requests in REST requests: GET, POST, PUT, DELETE. GET is used to retrieve resources, POST is used to create new resources, PUT is used to update a specific resource, and DELETE is used to remove a specific resource. POST is safer than GET because the data are not stored in the logs or browser history, and the transport data has no length limitation. We use POST requests to create new resources. However, after submitting data to the server, the server should respond to the client with the new data or the result of submission. This response is just like a response for a GET request, so we redirect to a new page, just like the client sends another GET request to get the submitted result.

e. [5 marks] Express.js and Django (with a plugin such as Tastypie) both allow the creation of a backend API which a frontend such as that built in React could be implemented within. If the client interface was served via a Django ‘view function’, state a downside in terms of deployment within an AWS architecture, when compared to an ‘independent’ client interface (i.e. how we do it with Express.js and React.js together).

🡪 When deploying “Express.js” and “React.js” within an AWS architecture, the React frontend project will be compiled and optimised and copy the output artifacts to the S3 bucket. We can use CloudFront to cache it worldwide to improve the loading performance for the end user. The Express backend project will be run on EC2, with the help of Lambda, using a serverless solution could reduce the time and resources.

If we deploy the Django project as a client interface, we should run the project on EC2 all the time to accept the request at any time. When the client submits a request, the Django project will generate a full HTML page and send it to the client; this consumes a lot of bandwidth resources and more data transferred.

In conclusion, deploying the Django project as a client interface within an AWS architecture needs more computing time, network bandwidth, data transfer, and poor user loading performance.

f. [5 marks] How would you build the data model if this were a Django project? Describe (in terms of what **models.py** would contain) what it would look like if this project were implemented in Django.

🡪 I will create 4 models to store the data: ItemData, StorageData, CartData and OrderData. The attribute info is shown in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Attribute** | **Data type** | **Explanation** |
| ItemData | name | CharField | Used to record the basic information of each item/food/dish. |
| description | CharField |
| price | FloatField |
| StorageData | amount | IntegerField | The storage amount of each item/food/dish. |
| item | ForeignKey |
| CartData | amount | IntegerField | The item data added to the cart. The amountMax is used to control the maximum value the user could add. |
| amountMax | IntegerField |
| item | ForeignKey |
| OrderData | orderId | IntegerField | The past orders information. |
| items | ArrayField |
| amounts | ArrayField |

Then insert initial data following ItemData and StorageData models into the database.

Next, create some HTML pages using Jinja template:

1. Display the food list to the user and let the user select food and send a POST request to create data and save it into the database following the CartData model.

2. Display the cart page to the user and let the user adjust the food amount, send a DELETE request to delete from the cart, send a POST request to create an order, and save it into the database following the OrderData model.

3. Display the past order page to the user and let the user review the order list and select the post order to delete (send a DELETE request).

Last, implement the logic in views.py and set the route in urls.py.