Video 2 : frontend

1. Create virtual environment by using command : pipenv shell
2. Cd into leadmanager and creating folder named frontend by using command : python manage.py startapp frontend
   1. Inside frontend we created folder with name src and inside this folder we created another one called components. The source folder is where all of our components, all of our redux , everything that we do in terms of our react application goes in here .
   2. Inside frontend we created folder with name static and inside this folder we created another one called frontend. Static will be the compiled js .
   3. Inside frontend we created folder with name templates and inside this folder we created another one called frontend. Templates is going to handle the index.html file that gets loaded, basically our single page .
3. So we will have webpack take our react application we are gonna have an index js as and entry point in the source folder, its gonna look at that and its gonna compile it down to a file called main.js inside static.
4. So if you have used create react out before when you run npm run build , it creates a static folder with all your static assets that’s gonna live in the static folder.
5. Go out to the root by cd ..
6. Installing nodejs from nodejs.org
7. Creating json file with al the dependencies by the command npm init -y.
8. Installing webpack : npm I -D webpack webpack-cli
9. Installing babel : npm I @babel/core babel-loader @babel/preset-env @babel/preset-react babel-plugin-transform-class-properties –save-dev
10. Installing react: npm I react react-dom prop-types
11. In order to use the plugin and the dependencies we need to create file in the root folder with the name .babelrc
    1. It a json object with the presets and the plugin
12. Another file to create in the root with the name webpack.config.js, this file used whenever we use webpack, with this file we load the babel loader, that will allow us to use babel to transpile our code.
13. Making changes in the package.json file to compile our react application that lives in the frontend app. Compiling the js file in the src folder and output it to the static folder
14. Now we create the file index.js in source folder: with this file we load the main app component.
15. And in the components folder that in the source we create app.js file, here we inserting the main app in the element with the id ‘app’.
16. In templates/frontend we create a file index.html where the app gonna be loaded at
17. Get style from bootstrap watch
18. Get js from webbootstrap
19. Add frontend to the sittings in leadmanager
20. Point to the template by using the file views.py in the frontend folder, but it wont do anything without linking it to the urls.py file and then including this url to the main usls file.
21. Linking url to the view.py file, by creating urls.py file in frontend dir.
22. Ad the url to urls.py file in the leadmanager dir
23. Run npm run dev from the project root directory , this create main.js, the compiled file for the app , and place it in static/frontend. And this file loaded to the index.html file that in the templates folder.
24. Then we restart the command line by control c
25. Go to the leadmanager folder by cd leadmanager and run the server by python manage.py runserver
26. ------------------------------------------------------
27. After checking that everything fine
28. ------------------------------------------------------
29. Create folder with the name layout in src/components and inside this folder we create js file with the name Header.js
30. Installing **ES7 React/Redux/GraphQL/React-Native snippets**  for the VSC
31. In Header.js we build the navbar using bootstrap
32. Import the header in the app.js file
33. Add watch to the package.json file in order not needing to rerun the dev command and compile new main.js file in each time we update new thing in the code
34. Create folder with the name leads in src/components folder and inside we create three js files : Dashboard.js Form.js and Leads.js
35. Install **Prettier – Code formatter**  for VSC : that’s gonna handle the spacing

Problems:

1. Instead –output , we should write –output-path in the file package.json file
2. The main.js file will be created in static/frontend/main.js/main.js : in order to do the thing in a right way and not create the file main in the folder main , in the commands dev and build inside the package.json file we should delete the last folder in the path , so we delete “/main.js”

Video 3 : react-redux

1. Download chrome extension devtools
2. Install redux and some other libraries by the command : npm I redux react-redux redux-thunk redux-devtools-extension
3. In folder src we create a file with a name : store.js
   * We define the store in this file in order to provide it to the app.js (the main component)
   * The store is responsible for the reducers, so the reducers will make the actions in the store.
   * Also responsible for the initial state and the log files (something like that)
4. In folder src we create a folder for the reducers
   * In reducers we create file with name index.js, this file combines all the reducers in this folder witch they are a js files.
   * Each reducer responsible for set of actions to perform.
   * In reducers we create file with name leads.js, this file responsible for the actions happening on the leads and related to them
5. Making change in app.js in the folder components, her we add the provider to provide the store in order to use the react-redux.
6. In folder src we create a folder with a name : actions
   * This folder this folder contain the actions files
   * In folder actions we create a file with a name : types.js this file, this file contain the actions types in a variables
   * In folder actions we create a file with a name : leads.js, this file contain all the actions that the reducer will make when we dispatch action to him.
7. Install axios with the command : npm I axios

Video 4 : handling errors

1. Install couple of libraries in react by the command : npm I react-alert react-alert-template-basic react-transition-group
2. In the App.js file we add the alert provider and the alert component.
3. When error occur, it will be cached in one of the functions in actions/leads.js, when that’s happened, then action will dispatched to the error reducer and change the state, also the messages in the same way.
4. The alert component: will be in the folder layout with the name Alert.js.
   * alerts here are messages or errors, so we get errors and messages from state (the state of the two files in this folder: errors.js and messages.js) and provide them to the props.
   * componentDidUpdate is a function called with any update in this component’s state.

Problem : in file components/layout/Alert.js in the last line should be withAlert()(Alerts) instead of withAlert(Alerts) : in the red is the difference

Video 5

1. adding owner field to the leads as a foreignkey and give it a property to delete all the leads that linked to this owner if we delete this owner.
2. makemogrations and migrate to refresh the database.
3. change in the api.py file :
   * in order to get only the leads for that owner instead all the leads, and change the permissions to be IsAuthenticated instead of AllowAny.
   * Adding a function to save the lead owner when we create the lead.
4. Change in frontend/stc/actions:
   * Instead of making message in each catch() for every error when error occur, we can make function in the measages.js file to deal with this.
5. Changes in leadmanager/settings.py :
   * Add knox to the installed apps.
   * Add REST\_FRAMEWORK section to let the system that we use token authentication.
6. Creating a new app with name accounts:
   * Add it to the settings
   * Creating serializer.py file:
     1. Serializers:
     2. Creating the user serializer
     3. Creating the register serializer: define the fields, define the create function to create the user and return the user.
     4. Creating the login serializer : defining the fields and the validation function.
   * Creating the apy.py file, this file responsible for dealing with the client requests:
     1. API:
     2. Creating the registration API: define the serializer class and the post function, in post function we create the serializer , check if its falid, save it in the database and return response.
     3. Creating the login API.
     4. Creating the GET user API: we define the permissions, define the serializer class and the get pbject function that returns the user, and sense its with the authentication permission , then just will get the user with its authentication token.
   * Creating the urls.py file to interact with the API:
     1. URLS:
     2. Defining the links and API class that responsible for that link.
7. Include the accounts to the leadmanager usrls.py file.

Video 6

1. Install react dom with the command:
   * npm i react-router-dom
2. in file frontend/src/components/App.js we make changes in order to route the right page:
   * we add switch tag in order to route the first matching url, and then stop search for another component with the same url to route.
   * route tag is for route the component with the url we want to route.
   * Call loadUser action by the function componentDidMount whenever this component is displayed in the browser
3. In components we create folder with the name accounts.
4. In accounts we create file with the name Register.js:
   * Give it the state
   * Add the html tags to make the registration page.
   * Add the variables in the render function
   * Create onSubmit function
   * Create onChange function
5. Create the Login.js file and do the same thing as we did for the Register.js file
6. Add the login and the register to the Header.js file
7. Add auth to the reducers/index.js to combine it with all the reducers
8. Create auth.js in the reducers folder :
   * Add the initialState
   * Add the function to handle the actions
9. In components we create folder with the name common for the files that does not fit in with the other components
10. In folder common we create file with the name PrivateRouters.js, this file for displaying the user that logged in
    * PrivateRoute function will directly return the Route tag that include parameters and one of them is the function render that will render component depends on the authentication.
11. Add the types to actions/types.js three types of actions
12. In actions folder we create auth.js file to dispatch the actions.
    * Function to check the user is authenticated:
      1. First it will dispatch that the user is in loading mode.
      2. Second we get the auth token from the state.
      3. Then we set the header that we send with the request.
      4. After that we check the token and if its valid we send it with the header.
      5. Then we make the get request :
         1. In case it passed and got error then we dispatch that the user is loaded
         2. Otherwise, if error is catched then we dispatch the error response and dispatch the auth error

Video 7

1. Change In the file types.js
2. In actions/auth.js we add another functions:
   1. Login
   2. Logout
   3. register
3. In reducers/auth.js we handle the dispatched data
4. In components/accounts/Login.js we add the functionality login
5. In Header.js :
   1. we add logout in the header
6. in Alerts.js we add alerts to show login logout errors.
7. In components/accounts/Register.js we add the functionality register and createMessage to create error or success message
8. Add tokenConfig function tofunctions in the file actions/leads.js in order to get, add and delete leads for the current logged in user.