Summer Fellowship Report

On

Covid19 Learning Simulator (LeSi)

Django Web Application

Submitted by

Himanshu Kr. Mishra

Under the guidance of

Dr. Sahely Bhadra
Asst. Professor, CSE
IIT Palakkad

Dr. Mrinal Kanti Das
Asst. Professor, CSE
IIT Palakkad

July 7, 2020

Acknowledgment

The internship opportunity I had with FOSSEE, IIT Bombay was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period.

I am using this opportunity to express my deepest gratitude and acknowledge my sense of gratitude to Dr. Sahely Bhadra and Dr. Mrinal Kanti Das who mentored me during my project and kept me encouraged throughout, which helped me in the successful completion of my project. It is my radiant sentiment to place on record my best regards to my mentor for their careful and precious guidance which were extremely valuable for my study both theoretically and practically.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

Sincerely
Himanshu Kr. Mishra

Contents

Chapter 1

Introduction

Covid19 Learning Simulator (LeSi)

Covid19 Learning Simulator is an application that can be used efficiently to track the current situation of the population affected by the virus. This software is built as a project for the fellowship under FOSSEE 2020, IIT Bombay.

In short, LeSi is a python based web application which internally uses some machine learning algorithms and techniques to forecast the results for a population based in some specific location as well as the whole country.

Major task provided was to create the whole LeSi software using Django web framework with various features.

Whole system was built while following the given tasks

1. First task was to read and understand the LeSi on paper model as well as to understand the working of already built LeSi website.

2. Another main task was to forecast the results on a dynamic and running graphs instead of static one. There are total four 2-D graphs which shows the relation between the amount of population and the time period and displays the results for four major areas, infected people, hospitalized people, critical staged people and total deceased cases.

3. One very interesting task was to take the inputs of parameters for predicting the results from users through scrollable input bars for each parameters alongwith displaying the current value of certain parameters in a bubble element for the ease of setting the exact values.

4. Out of these, the important task was to make an authentication system for the website. So that if a user wants to use the simulator then he/she must be authorized one. Authentication system was supposed with the features like login, sign up, logout, reset password etc.

5. Challenging task was to implement the logic for forecasting in the backend. It includes certain important and vital operations such as calling the model file in the routes file, calling a python file/script responsible for plotting the graphs, passing all the necessary files such as CSV files for all location and PKL file for default values.

Challenges

While building the application several challenges came into the path, all of them are listed below

1. Understanding the model of Learning simulator alongwith it's functionality and architecture was bit tough for me as I was not having any prior machine learning knowledge and such pandemic models.

2. Initially it was difficult for me to understand the flow of using the model from initializing the inputs to plotting the graphs. Since a lot of operations was being done one by one.

3. Loading the PKL file and using the values as default for the model was also tough task because in the backend the system comprises more than one location and several parameters.

4. Implementing scrollable input bars was not that tough but dynamically rendering the current values was painful as it requires knowledge of JQuery and JavaScript.

5. Out of these, the most difficult and challenging task was to plot the dynamic graphs, which is still not created because of lesser exposure to the other smart technologies and lesser time to learn everything.

Project Source Code

https://drive.google.com/drive/folders/1sFh5TD5-gihBMvPeokIlbapx7sDPnbnf?usp=sharing
(Recommended)

OR

https://github.com/hkm007/Covid19-Learning-Simulator (Private Repository)

Chapter 2

Building LeSi

2.1 Software Requirements

For buiding LeSi, many softwares are used and required. Following is the detailed report on softwares, installation, technology stack etc.

1. **Python** – It is an interpreted, high-level, general-purpose programming language. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming.

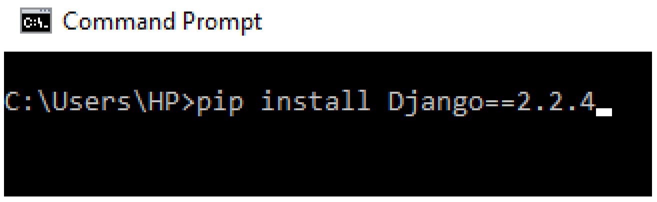
   For building LeSi, python version **3.7.2** is used however a version above or equal to 3.6 is sufficient.

   For installing python, download the interpreter from official website and install it into your system.

2. **Django** – It is a Python-based free and open-source web framework that follows the model-template-view (MVC) architectural pattern. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself.

   Django version **2.2.4** is used for building the software so make sure to install the same version of django.

   For installing django, make sure python is installed in the system then open the terminal and write the following command



```
C:\Users\HP>pip install Django==2.2.4
```

3. **Code Editor** – One can use any code editor of their personal choice. I have used VS code by Microsoft to buil this software.

4. **Other Requirements** – LeSi is built on windows 10 operating system but it can easily be installed and built on Linux as well. For debugging and creating the application it's recommended to use Google Chrome web browser but again one can use any web browser of their own choice.

2.2 Technology Stack

For building the complete application following technologies are used

1. **Python** – It is an interpreted, high-level, general-purpose programming language. Python is dynamically typed and garbage-collected.

2. **Django** – It is a Python-based free and open-source web framework that follows the model-template-view (MVC) architectural pattern.

3. **HTML** – **Hypertext Markup Language** (**HTML**) is the standard markup language for documents designed to be displayed in a web browser. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML is used for building the frontend skeleton of LeSi.

4. **CSS** – **Cascading Style Sheets** (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. CSS is used on a large strength in LeSi to design the frontend.

5. **Bootstrap** – **Bootstrap** is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

6. **JavaScript** – **JavaScript** often abbreviated as **JS**, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. In LeSi, javascript is used on larger scale for dynamically maintaining the website and frontend of LeSi. Major application constitutes of dynamically rendering the year in copyright section and dynamically rendering the scrollable input bars.

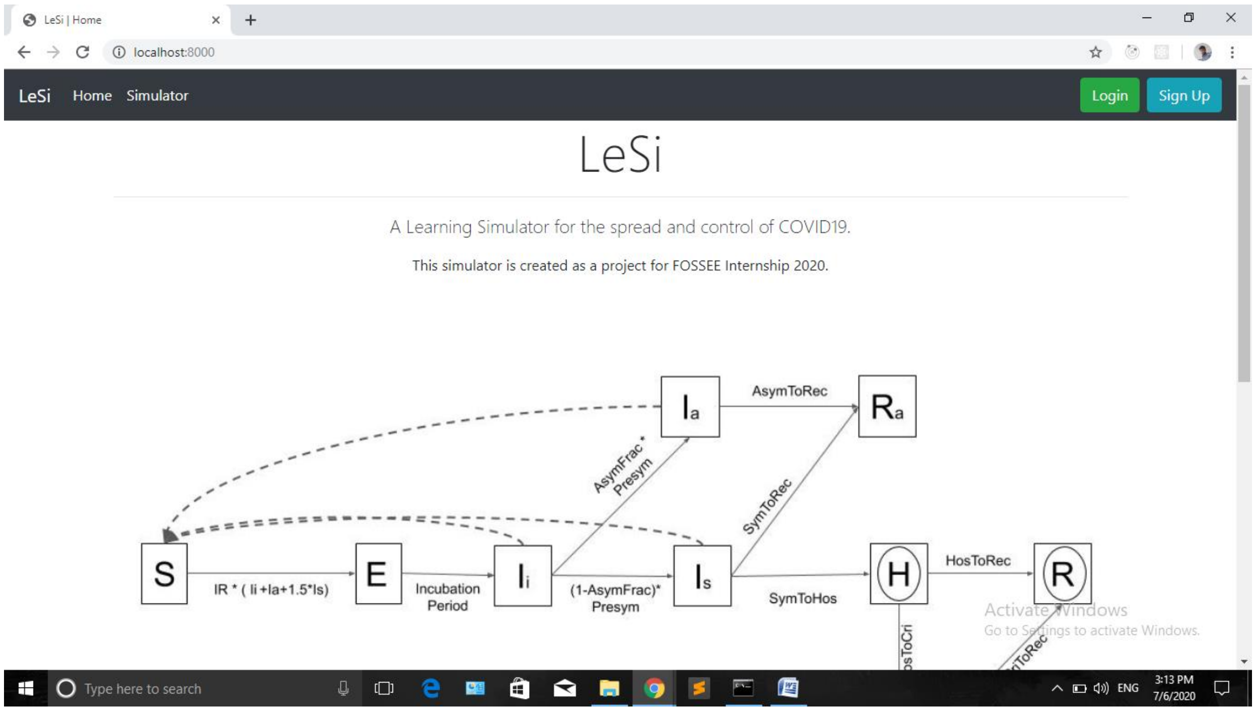
7. **Google reCAPTCHA** – **reCAPTCHA** is a provider of human verification systems owned by Google. The original iteration of the service was a mass collaboration platform designed for the digitization of books, particularly those that were too illegible to be scanned by computers. The verification prompts utilized pairs of words from scanned pages, with one known word used as a control for verification, and the second used to crowdsource the reading of an uncertain word. This service is used in LeSi, for increasing the security of web application or simulator from various cyber attacks by robots and for limiting the unwanted server hits from unauthorized users.

2.3 Architecture of LeSi

Pages

Mainly there are 3 pages in the web application namely **Home, Simulator and Results/Output** pages.

Home Page



LeSi

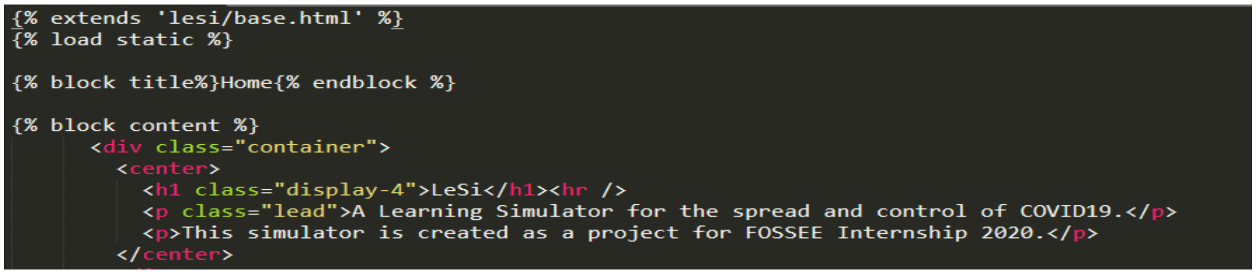
A Learning Simulator for the spread and control of COVID19.

This simulator is created as a project for FOSSEE Internship 2020.

Home page is designed using HTML, CSS, Bootstrap and JavaScript. It contains the details of the LeSi model alongwith it’s diagram which depicts the 10 compartment pandemic model.

This file can be customized while editing the **home.html** file stored at **LeSi>lesi>templates>lesi>home.html**.

Home page is rendering the elements such as footer, navbar etc from another file named **base.html** stored at **LeSi>lesi>templates>lesi>base.html**.

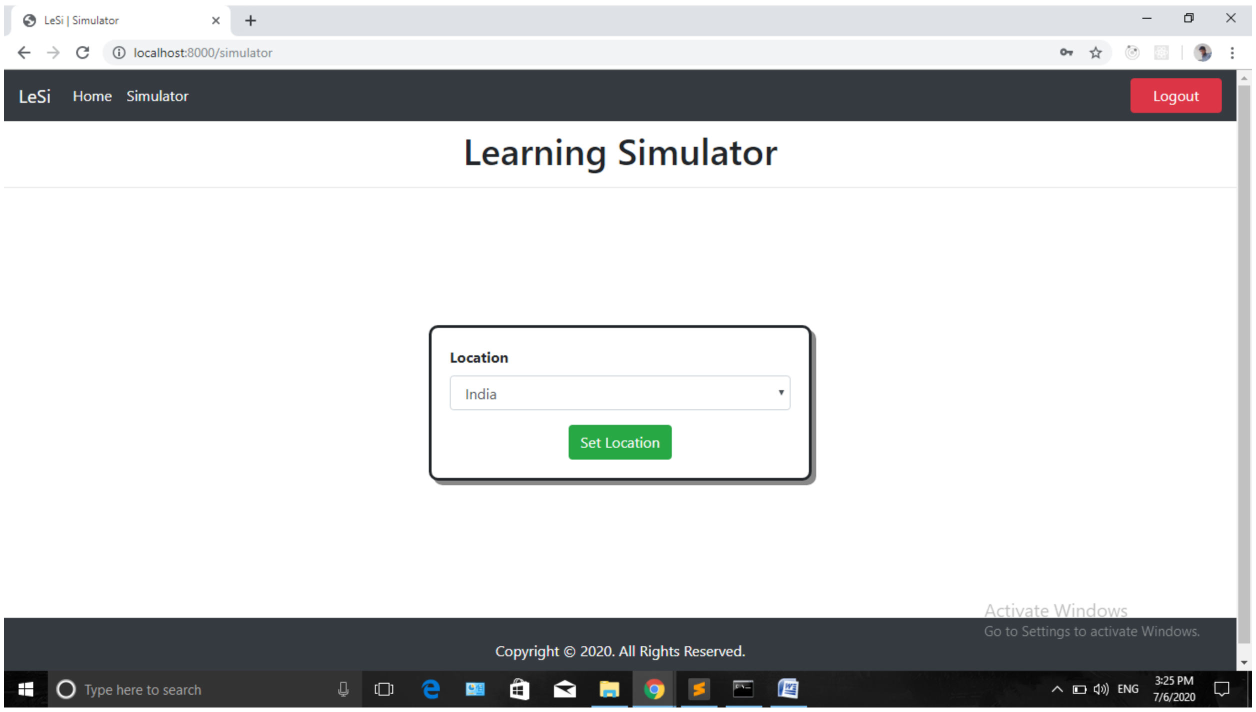


```
{% extends 'lesi/base.html' %}
{% load static %}

{% block title%}Home{% endblock %}

{% block content %}
      <div class="container">
        <center>
          <h1 class="display-4">LeSi</h1><hr />
          <p class="lead">A Learning Simulator for the spread and control of COVID19.</p>
          <p>This simulator is created as a project for FOSSEE Internship 2020.</p>
        </center>
```

Simulator Page



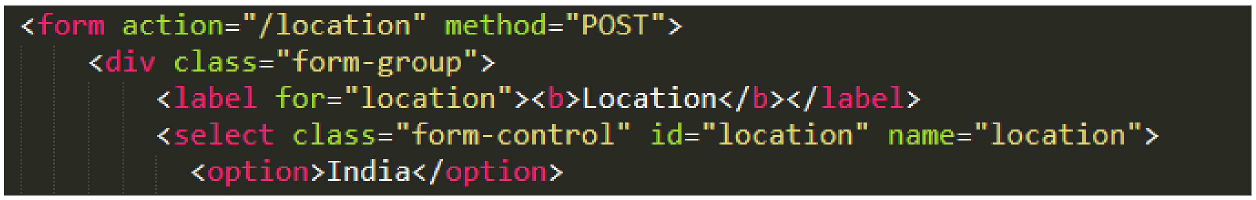
Simulator page is designed using HTML, CSS, Bootstrap, JavaScript etc. This page is saved as **simulator.html** in the given location **LeSi>lesi>templates>lesi>simulator.html**.

This page is rendering the elements such as footer, navbar etc from another file named **base.html** stored at **LeSi>lesi>templates>lesi>base.html**.

This page is secured and in order to use this website one must have an account on LeSi.

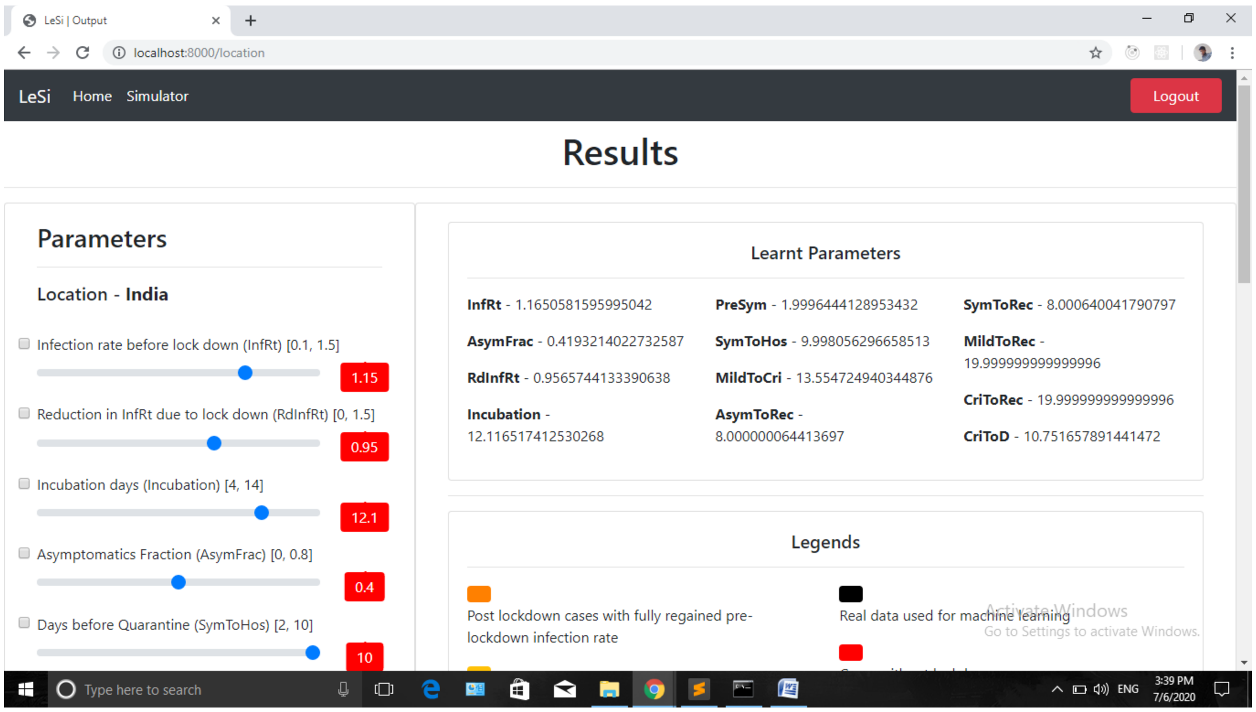
It has a dropdown list which contains many locations or states of India and the country itself. This dropdown list is created using select tag in html and it’s bordering are designed using different classes of boostrap and css.

The card also has a button with text ‘Select Location’ once you select the location one can click the button to predict the results. Basically it’s a form in html which on the click of button makes a request to the server as a POST method to post the value of location to backend where it can be processed for further forecasting related operations.



```
<form action="/location" method="POST">
    <div class="form-group">
        <label for="location"><b>Location</b></label>
        <select class="form-control" id="location" name="location">
          <option>India</option>
```

Output Page



Results

Parameters

Location - **India**

Infection rate before lock down (InfRt) [0.1, 1.5] 1.15

Reduction in InfRt due to lock down (RdInfRt) [0, 1.5] 0.95

Incubation days (Incubation) [4, 14] 12.1

Asymptomatics Fraction (AsymFrac) [0, 0.8] 0.4

Days before Quarantine (SymToHos) [2, 10] 10

Learnt Parameters

InfRt - 1.1650581595995042
AsymFrac - 0.4193214022732587
RdInfRt - 0.9565744133390638
Incubation - 12.116517412530268
PreSym - 1.9996444128953432
SymToHos - 9.998056296658513
MildToCri - 13.554724940344876
AsymToRec - 8.000000064413697
SymToRec - 8.000640041790797
MildToRec - 19.999999999999996
CriToRec - 19.999999999999996
CriToD - 10.751657891441472

Legends

🟧 Post lockdown cases with fully regained pre-lockdown infection rate

⬛ Real data used for machine learning

Output page is the most important and delicate page of LeSi simulator application. It has many child components.

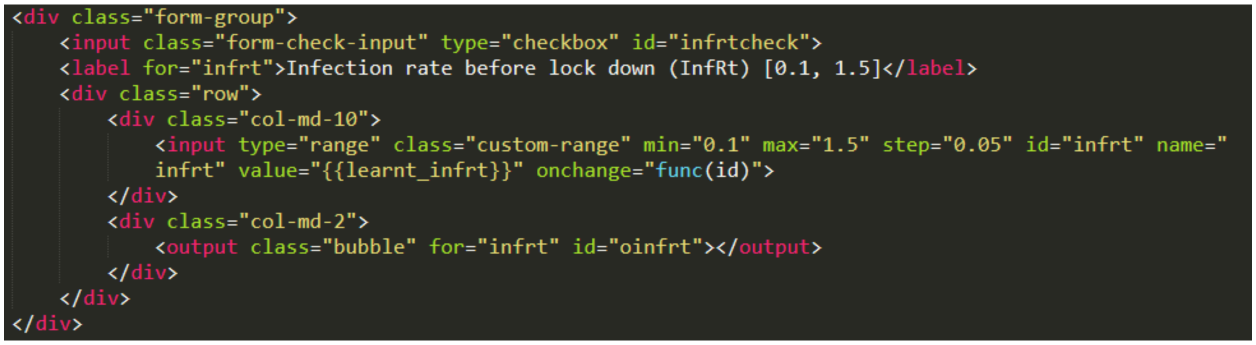
First of all it also renders some components like navbar, footer etc from **base.html** file stored at **LeSi>lesi>templates>lesi>base.html**.

Output page consists of many components like one component for displaying graphs, another for displaying, learnt parameters, one shows legends and out of these most important one is the parameters component. It is basically a form which can be used to run the model on your choice of parameters.

Parameters Section

It is one of the most important section of the website as it allows user to forecast the results on their own while customizing the parameters accordingly. It is basically a form in which various parameters are denoted that can be changed with the scrollable input bars attached with them and also there is a red bubble element in the right side which depicts the current value of the bar.

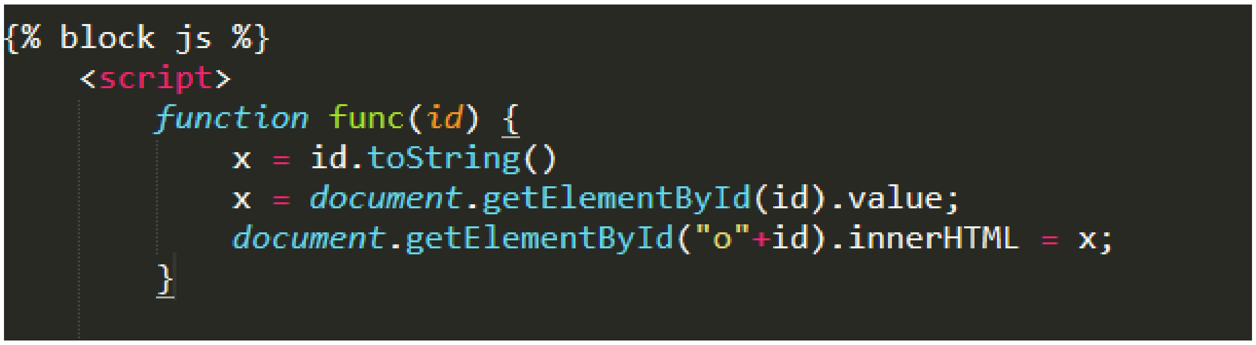
HTML input element is used to design the scrollable bar while output tag is used for showing the bubble for exact current value. But for making it dynamically according to the scroll, javascript is used at larger extent.



```
<div class="form-group">
    <input class="form-check-input" type="checkbox" id="infrtcheck">
    <label for="infrt">Infection rate before lock down (InfRt) [0.1, 1.5]</label>
    <div class="row">
        <div class="col-md-10">
            <input type="range" class="custom-range" min="0.1" max="1.5" step="0.05" id="infrt" name="
            infrt" value="{{learnt_infrt}}" onchange="func(id)">
        </div>
        <div class="col-md-2">
            <output class="bubble" for="infrt" id="oinfrt"></output>
        </div>
    </div>
</div>
```

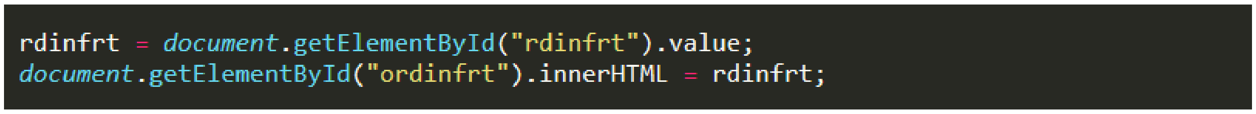
A single scrollable input looks like this in it's code. Each element has it's unique id. Every **input** tag has their own minimum and maximum values and a current value which is dynamically coming from the backend. Ex – {{learnt_infrt}} in the above picture is coming from backend. Scrolling is done with a step size of **0.05**. Now every input element has a JavaScript function attached with it using onchange attribute of input tag with name **func** in which the elements id is passed as an argument.

Now the "func" is executed in the script section below the whole code.



```
{% block js %}
    <script>
        function func(id) {
            x = id.toString()
            x = document.getElementById(id).value;
            document.getElementById("o"+id).innerHTML = x;
        }
```

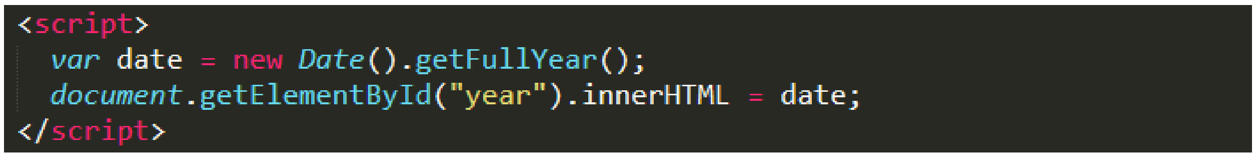
This **block js** section is responsible for executing all the javascript code in the frontend. Now the default values are also displayed in the output tag according to the given code written in javascript.



```
rdinfrt = document.getElementById("rdinfrt").value;
document.getElementById("ordinfrt").innerHTML = rdinfrt;
```

In JavaScript, just the id of each element is taken and manipulated using the functions. And hence the each element has unique id, code works completely fine and manipulates the DOM as we wish.

Footer

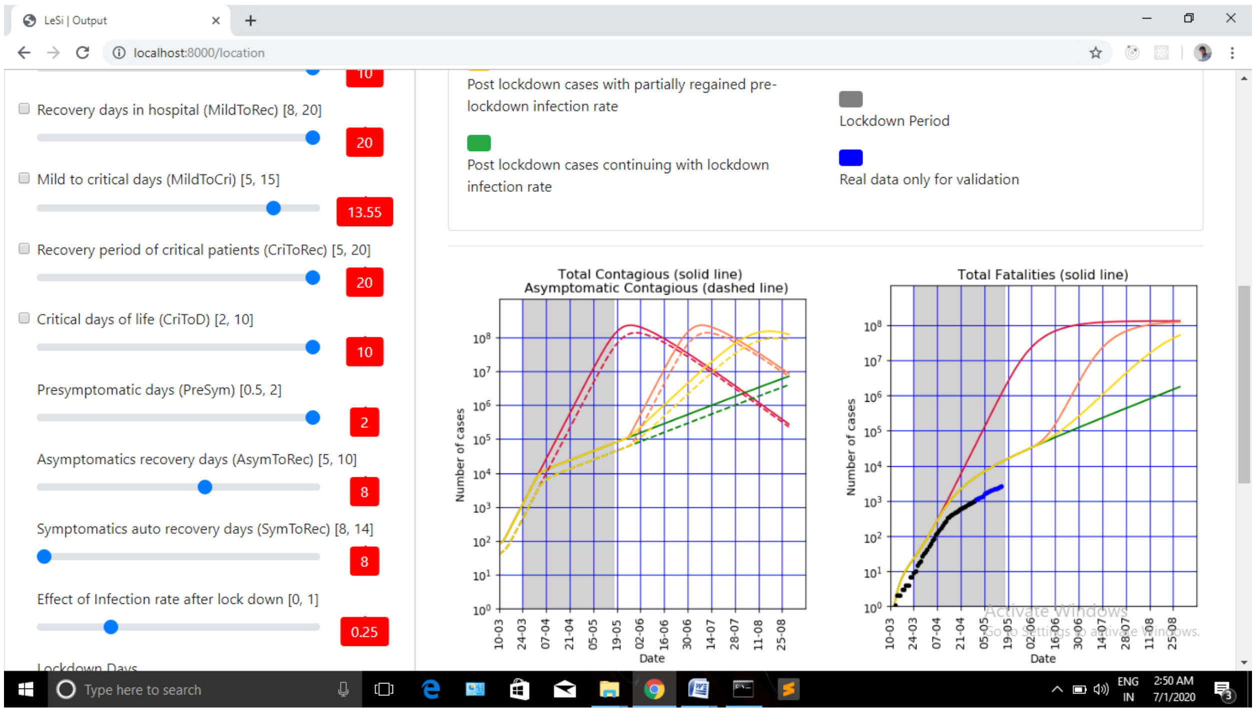


```
<script>
  var date = new Date().getFullYear();
  document.getElementById("year").innerHTML = date;
</script>
```

Year in footer is dynamically rendered using the above JavaScript code written in the end of **base.html** file.

Graphs

Most important component of the website is the graphs part that are being rendered on the frontend as an image saved in a folder as PNG files in the backend.



Recovery days in hospital (MildToRec) [8, 20] 20

Mild to critical days (MildToCri) [5, 15] 13.55

Recovery period of critical patients (CriToRec) [5, 20] 20

Critical days of life (CriToD) [2, 10] 10

Presymptomatic days (PreSym) [0.5, 2] 2

Asymptomatics recovery days (AsymToRec) [5, 10] 8

Symptomatics auto recovery days (SymToRec) [8, 14] 8

Effect of Infection rate after lock down [0, 1] 0.25

Lockdown Days

Post lockdown cases with partially regained pre-lockdown infection rate

🟩 Post lockdown cases continuing with lockdown infection rate

▨ Lockdown Period

🟦 Real data only for validation

All the four images are stored as c.png, d.png, r.png and h.png depicting critical patients, deceased patients, recovered cases and hospitalized ones respectively.

Each image in frontend is displayed using given code written in **output.html** file

```
<div class="row">
    <div class="col-md-6">
        <img src='{% static "lesi/results/c.png" %}' alt="c" style="height: 450px; width: 400px;">
    </div>
    <div class="col-md-6">
        <img src='{% static "lesi/results/d.png" %}' alt="d" style="height: 450px; width: 400px;">
    </div>
    <div class="col-md-6">
        <img src='{% static "lesi/results/h.png" %}' alt="h" style="height: 450px; width: 400px;">
    </div>
    <div class="col-md-6">
        <img src='{% static "lesi/results/r.png" %}' alt="r" style="height: 450px; width: 400px;">
    </div>
</div>
```

These image files are generated by the model in the backend. They are basically a python file which are called in the backend. And since all the files are stored in a static folder and rendered a static files django uses a different syntax for loading the static files as {% load static %}.

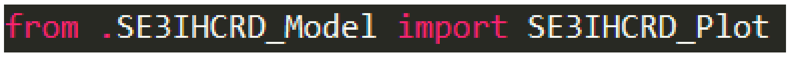
Model Python File

Heart of this application is the model python file that are responsible for generating results. It uses some algorithms to predict or forecast accurately.

Name of this model file is **SE3IHCRD.py**. It is stored at the given location **LeSi>lesi>SE3IHCRD_Model>SE3IHCRD.py**

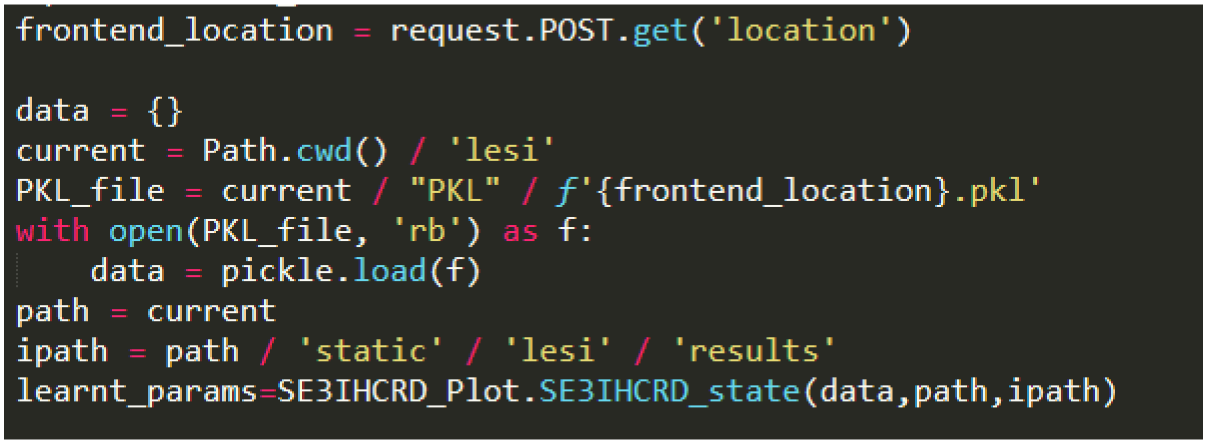
Alongwith this file the same folder contains one more important file named **SE3IHCRD_Plot.py**. This file is responsible for creating a connection between the backend and the model file i.e. **SE3IHCRD.py.** And the same file is responsible for creating the image files as graphs that are saved in the results folder.

SE3IHCRD_Plot.py file internally call the function from **SE3IHCRD.py** and generates the results. And the **SE3IHCRD_Plot.py** is called inside the main logic file of backend i.e. **views.py** file. The code used for calling the file is depicted below



```
from .SE3IHCRD_Model import SE3IHCRD_Plot
```

And then the function inside the files are called in the routes created in views.py file. And since the functions requires various arguments as well the PKL file for a location for the purpose of default parameters are all coded as

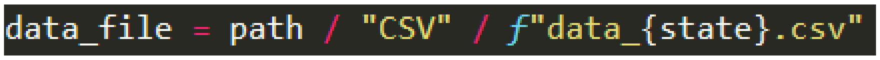


```
frontend_location = request.POST.get('location')

data = {}
current = Path.cwd() / 'lesi'
PKL_file = current / "PKL" / f'{frontend_location}.pkl'
with open(PKL_file, 'rb') as f:
    data = pickle.load(f)
path = current
ipath = path / 'static' / 'lesi' / 'results'
learnt_params=SE3IHCRD_Plot.SE3IHCRD_state(data,path,ipath)
```

SE3IHCRD_state.py is the function written in **SE3IHCRD_Plot.py** file in which three arguments are passed, data is a dictionary which contains default values for certain location loading from the PKL file stored in a specific folder named **PKL**. Now path and ipath are the location or path of directory containing the models and the location for storing the images as PNG files.

SE3IHCRD_Plot.py internally uses the CSV files stored in a separate folder named CSV using following code



```
data_file = path / "CSV" / f"data_{state}.csv"
```

And then it reads the data using pandas library.

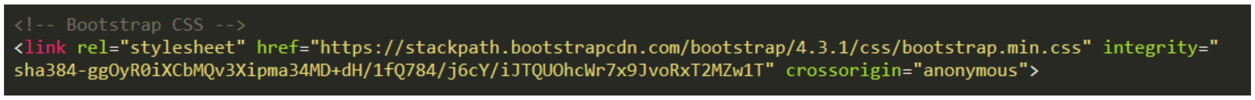
Inputs Logic

Once you reach the output page now you can predict the results on your own. One can change the parameters values using the scrollable inputs whose functioning is described in earlier section. Now there are checkboxes given with some parameters using which you can define specifically which parameters you want to predict. Just click the checkbox and then use the button **“Run LeSi”** in the bottom. A form will post all the data to the backend where the model file generates the graphs and redirects you to the output page. Whole logic is written in **views.py** file inside a function route named **handleInput**.

Responsive Design

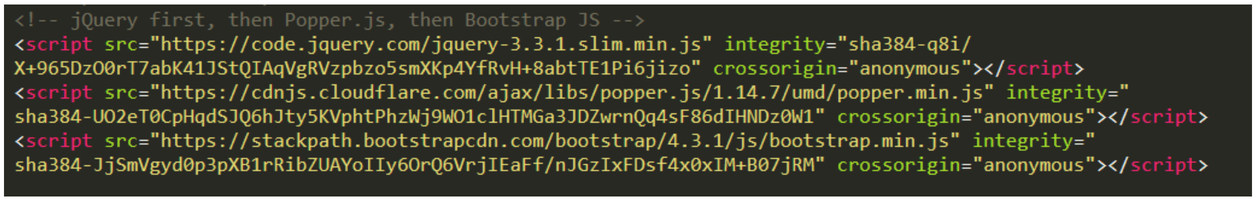
LeSi is made completely responsive for different sized screens such as laptops, mobile phones, tablets etc. It adjusts itself according to the size of screen and gives a better user experience.

Bootstrap library is used for adding the responsive nature of the website. Firstly it’s CDN is linked to **base.html** file in it’s header section as



```
<!-- Bootstrap CSS -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="
sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
```

And then some javascript dependencies and JQuery files are also added in script section in the bottom of the same file as



```
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/
X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="
sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="
sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
```

After adding all this files, everywhere bootstrap styling measures have been used for designing as well as structuring the wesite.

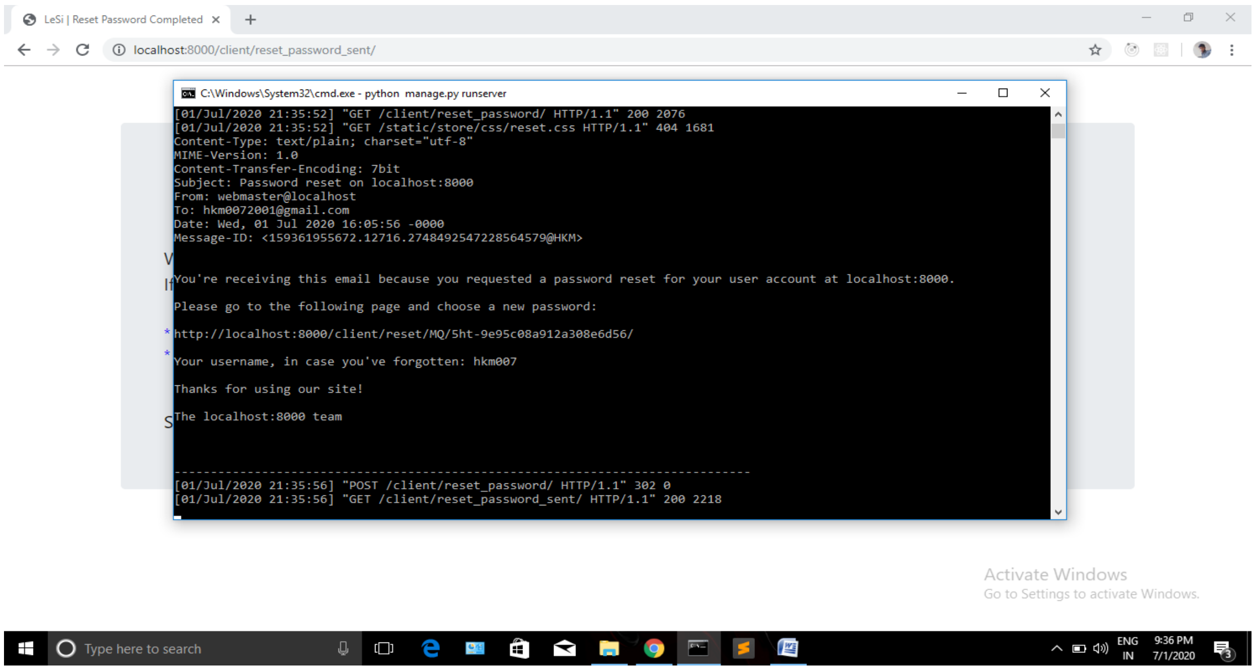
Chapter 3

Limitations & Future Scope

The LeSi web application currently displays the static images or graphs for the forecast. The original task was to create a dynamic graph which plots the running graph.

Since Django is mainly a backend framework and don’t have high capabilities to manipulate the frontend it was a tough task . Due to lesser exposure to other technologies and less time to learn them, this task was not completed. So instead of a dynamic plotting, static graphs are displayed on the frontend.

One major limitation is associated to the authentication system is that if someone wants to reset the password then instead of sending mails to the email id the reset link is sent to the same terminal where the server is running. It was not achieved because to setup an email service and atleast for testing purpose, django server must have a signed email account that can be recognized by Gmail otherwise they doesn’t accept the mail. And since LeSi was in development phase only not launched and doesn’t have an authorized email, terminal based logic is used for now. But it can be changed to email service anytime by adding few lines of code in **settings.py** file.



LeSi | Reset Password Completed
localhost:8000/client/reset_password_sent/
C:\Windows\System32\cmd.exe - python manage.py runserver
Activate Windows
Go to Settings to activate Windows.
Type here to search

```
[01/Jul/2020 21:35:52] "GET /client/reset_password/ HTTP/1.1" 200 2076
[01/Jul/2020 21:35:52] "GET /static/store/css/reset.css HTTP/1.1" 404 1681
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: Password reset on localhost:8000
From: webmaster@localhost
To: hkm0072001@gmail.com
Date: Wed, 01 Jul 2020 16:05:56 -0000
Message-ID: <159361955672.12716.274849254722856457 9@HKM>


You're receiving this email because you requested a password reset for your user account at localhost:8000.

Please go to the following page and choose a new password:

http://localhost:8000/client/reset/MQ/5ht-9e95c08a912a308e6d56/

Your username, in case you've forgotten: hkm007

Thanks for using our site!

The localhost:8000 team



-------------------------------------------------------------------------------
[01/Jul/2020 21:35:56] "POST /client/reset_password/ HTTP/1.1" 302 0
[01/Jul/2020 21:35:56] "GET /client/reset_password_sent/ HTTP/1.1" 200 2218
```

For now LeSi can produce results for six different states namely Delhi, Karnataka, Kerala, Maharashtra, Tamil Nadu and West Bengal and for whole India. But it is designed in such a way that if in future one wants to add other locations, it can easily be achieved with just storing the data files and adding states in frontend dropdown list.

Chapter 4

Running Application

4.1 Starting Server

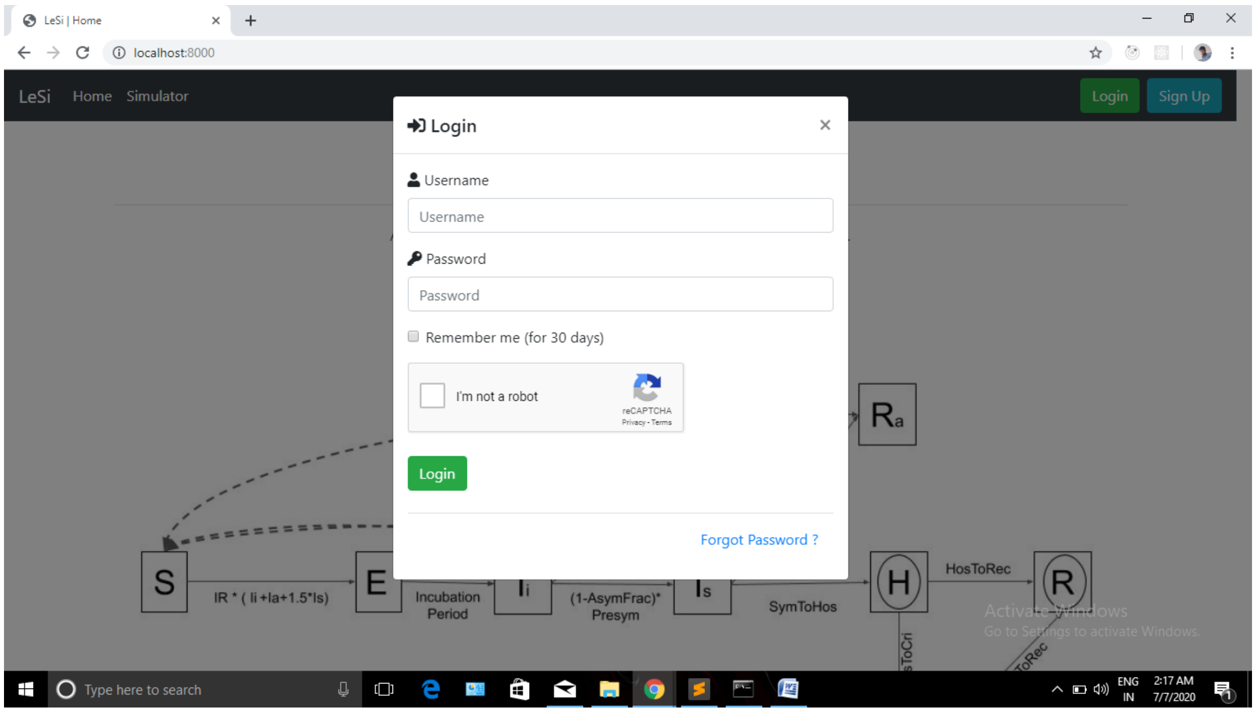
For starting the server just follow these steps :

1. Install python 3.7.2
2. Then run the command in the project directory
   "**pip install –r requirements.txt**"
3. navigate to the project directory containing **manage.py** file
4. open command prompt/terminal in the same directory
5. run the following command
   "**python manage.py runserver**"
6. Open web browser and search the following url
   '**127.0.0.1:8000**' or '**localhost:8000**'

4.2 User Authentication

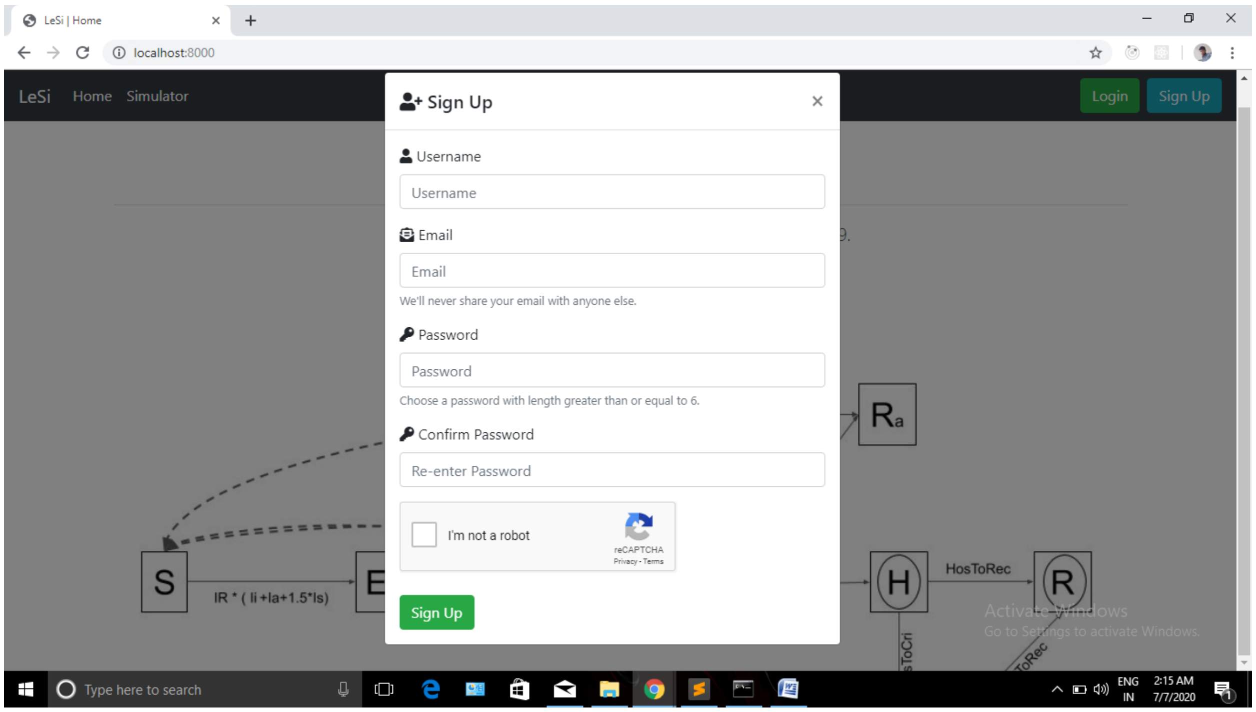
In order to use the simulator, user must create an account on it.

Login



Fill the form correctly and click enter. A success message will be displayed. Now in case if you don't have an account on LeSi, just click Sign Up button and create an account with us.

Sign Up



Sign Up

Username: Username

Email: Email

We'll never share your email with anyone else.

Password: Password

Choose a password with length greater than or equal to 6.

Confirm Password: Re-enter Password

I'm not a robot

Sign Up

Password

Make sure to have a strong password in order to secure your LeSi account and remember it. You can change your password anytime.

Reset Password

What if you forgot your password ?

Don't worry LeSi have the feature to reset your password in a very simple and quickest way.
Just click Forgot Password option on login form and follow the instructions provided on the screen to successfully reset your password.

Note

Currently the reset password system is working as a console based architecture, in future during further development it can be changed anytime to email verification system.

So the reset password link will be sent to your console/terminal on which the server is running.

Logout

Anytime your work is done and you are about to leave, don't forget to click Logout button and come out of the simulator.

4.3 Using Simulator

Parameters

In total 15 parameters are there listed in the frontend.

1. InfRt – Infection rate before lock down. It's range is from 0.1 to 1.5.
2. RdInfRt – Reduction in InfRt due to lock down. It's range is from 0 to 1.5.
3. Incubation – It depicts the incubation days. It's range is from 4 to 14.
4. AsymFrac – It stands for aysmptomatic fraction. It's range is from 0 to 0.8.
5. SymToHos – It stands for total days before quarantine. It's range is from 2 to 10.
6. MildToRec – It stands for total recovery days in hospital. It's range is from 8 to 20.
7. MildToCri – It stands for mild to critical days. It's range is from 2 to 10.
8. CriToRec – It stands for total days before quarantine. It's range is from 2 to 10.
9. CriToD – It stands for total days before quarantine. It's range is from 2 to 10.
10. PreSym – It stands for total days before quarantine. It's range is from 2 to 10.
11. AsymToRec – It stands for total days before quarantine. It's range is from 2 to 10.
12. SymToRec – It stands for total days before quarantine. It's range is from 2 to 10.
13. Effect of infection rate – It's range is from 0 to 1.
14. Lockdown – It depicts for the total lockdown days.
15. Period – It depicts the period of forecasting days. In general can be taken around 50.

Select Location

In the simulator page one can select the location for predicting the results. Once clicked the button, a post request is made to the server. Then in server a function created in **views.py** file named **handleLocation** runs. It grabs the location and according to the location loads the PKL file for default values and then calls the model file's function which generates the results and then redirects the user to the output page. Alongwith redirecting it also sends a dictionary of learnt parameters to the frontend returned from the model function. Which are then displayed in a separate section at the top of the page.

Plots

Plots are basically a 2-Dimensional graph where y-axes represents the total number of cases and x-axes represents the time period of forecasting.

Many colors are used while plotting the graph.

Orange - Post lockdown cases with fully regained pre-lockdown infection rate
Yellow - Post lockdown cases with partially regained pre-lockdown infection rate
Green - Post lockdown cases continuing with lockdown infection rate
Black - Real data used for machine learning
Red - Cases without lockdown
Grey - Lockdown Period
Blue - Real data only for validation

4.4 Security

While creation of project, security of web application cum simulator was taken as the most prioritized thing. Basically the website was secured in different ways

1. Securing it from unauthorized access
2. Security from unwanted form submission
3. Securing the website from cyber attacks or robot attacks

Security from unauthorized access

Web application is completely secured from unauthorized access. If you want to access the simulator one must be logged in. Otherwise he/she will get an warning message and will be redirected to the home page. In order to use the simulator simply log in or create an account.

Security from unwanted form submission

To prevent the server from unwanted and unnecessary web hits via form submission, csrf tokens are used with all the forms. Since django is also much concerned about csrf tokens, it makes the process more secure.

Cross-site request forgery, also known as **one-click attack** or **session riding** and abbreviated as **CSRF** (sometimes pronounced *sea-surf*) or **XSRF**, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.

Securing the website from cyber attacks or robot attacks

Many times we come across a situation when a cyber or robot attack happens on a website which may results into data loss, server crash, high traffic, database losses etc.

One very intuitive way of controlling this activites are achived in the Learning Simulator via using Google reCAPTCHA. This technique can be used to prevent server from such harmful attacks.

All the forms in the website are secured using reCAPTCHA. User must verify the process in order to use the simulator in a correct way.

If a script or robot tries to make requests on the server, it won't be, because it can't pass the reCAPTCHA verification. Thus, only humans can make request on the server.

4.5 Messages

All the warnings and informations are displayed on the screen using alert component on the frontend in which messages are dynamically rendered using django messages framework.

Each category of message is displayed with different color. For example a warning is displayed in yellow color and a success message is displayed in green color.

Chapter 5

References

1. https://covid.iitpkd.ac.in/lesi
2. https://www.python.org/doc/
3. https://docs.djangoproject.com/en/2.2/
4. https://tutorial.djangogirls.org/en/
5. https://www.w3schools.com/
6. https://developer.mozilla.org/en-US/docs/Web/JavaScript
7. https://codewithharry.com/videos/python-django-tutorials-hindi-0
8. https://stackoverflow.com/
9. https://www.youtube.com/playlist?list=PL-osiE80TeTtoQCKZ03TU5fNfx2UY6U4p
10. https://www.latex-tutorial.com/tutorials/