Codebase Structure Document

- We have developed a web application using the **ASP .NET MVC framework** for which we have used **Visual Studio 2019** to develop the web application. We have used **jquery, Javascript, LINQ, C#** and **Partial Views** to handle the front-end as well as back-end functionalities.

- Since we have used the 'MVC' framework, the three main components in our web application are **'Models', 'Views' and 'Controllers'**. Furthermore the scripts and css files are merged into different folders '**Scripts', 'content' and 'context'**. Since we frequently change our database, we have used the 'code-first' approach to develop our web application. The code-first approach requires the models to migrate manually using the 'package manager console' – more has been elaborated in the 'Readme' guide on our github repository. Following points further enlighten our codebase structure:

- **Models:** Models are basically the 'classes' to represent our tables in the database. For this iteration we have mainly 3 models. After these models are developed, use migration to create the actual tables in the database.
    1. **allCouncilPostcode.cs :** This model has all the properties needed to store the details of the postcodes of all councils. It has a one-to-many relationship between the council and the postcodes. This model has been used to map the relationship of the postcode entered by the user and the council to which it belongs.

    2. **councilRate.cs:** This model has the properties that store the ranking, the recycling rate and the name of the council. This model has been used to show the ranking and the recycling rate of the council to which the user postcode belongs.

    3. **Plastic.cs:** This model stores the properties of all the plastic items. This model has been mainly used for our 'Alternatives' feature where we are showing the alternatives to the plastic items adolescents use in their daily life.

- **Controllers:** Controllers are basically a 'medium' between our 'Models' and 'Views'. These controllers have actions that take data from models and pass it to the views using the actions. This data is fetched from the database using **'LINQ'** queries which act kind of like **'SQL'** queries to fetch data. So normally there is 1 controller for every model (however it is not necessary all the time). So in our codebase for the iteration 2, we have the following controllers:
    1. **HomeController.cs:** The HomeController is the one which handles our main Home screen (the screen user views once they open the website). There is no

model associated with this controller since we are not fetching any data from the database to develop the view for the home screen.

2. **AlternativesController.cs:** The AlternativesController is the controller which is mainly responsible for our 'Alternatives' feature. This controller is associated with the 'Plastic.cs' model and fetches data from this model.

3. **councilRatesController.cs:** This controller is associated with the 'councilRate.cs' model. It fetches data from this model and is responsible for passing the ranking and the recycling rate of the respective council.

4. **allCouncilPostcodesController.cs:** This controller is mainly responsible for fetching the mapping of the councils and their respective postcodes. This basically takes the postcode entered by the user and using 'LINQ' it fetches the council related to the given postcode.

- **Views:** Views basically represent the user interface, or we can say the 'layouts' that the users 'see'. For every action that we have in our controller, we have a respective View. Following are the views in our codebase structure:
  1. **Home index.cshtml:** This view is mainly responsible for showing the main home screen to the user.
  2. **Home recyclingsuggestions.cshtml:** This view shows the suggestions to the users regarding how they can improve their recycling rate.
  3. **Alternatives index.cshtml:** This view shows the alternatives for the plastic items to the users.
  4. **councilRates index.cshtml:** This view gets the postcode input from the user and matches it with the respective council and sends the further data needed to a **partial view** called 'RatePartial.cshtml'.
  5. **councilRates RatePartial.cshtml:** This view is responsible for showing the recycling rate and the ranking of the council on a 'gauge'.
  6. **allCouncilPostcodes index.cshtml:** This is responsible for showing the ranking and recycling rate of all the councils.

- **Scripts:** There are various javascripts used in our web application to develop various functionalities.
  1. **jQuery:** jQuery is used to show dynamic data stored in our database instead of showing just the static data. Also, it has been used in our application to call different actions in the controllers on button click.
  2. **Gauge.js:** This javascript has been used to develop the 'guage' feature in our web application.
  3. **Popper.js:** This javascript has been used to show add css effects to the 'popup' on the alternatives page.

- **Content:** This has datatables and the images used in our web application.
    1. **Datatables:** This has been used to show the 'full results' of all the councils in Melbourne.
    2. **Images:** Images folder is used to store all images.
    3. **Bootstrap:** This is used to add amazing css effects to the web application.

- **Context:** Context is used to save the dbcontext to interact with the Microsoft SQL.

- **Migrations:** When the models are migrated then it creates the mandatory files.

- **Shared:** Shared folder has all files which can be shared with all common layouts.

- **App_Data:** This has the database mdf file.

Furthermore, the **mathematical model** was developed in **Jupyter Notebook** can be found in our github repository.