Software Design Document

1. Tools for building the system:

- **Server-side and client-side programming languages:**
    - On the client side we will use Java script (React library), HTML, CSS.
    - On the server side we will use Python (Flask library with WSGI).
- **Databases:**
    - Relational database (SQL).
- **Open source libraries:**
    - React.
    - Flask.
    - WSGI.

- **Algorithms:**
    - The average algorithm.
    - The median algorithm with linear functions.

2. system structure:

- **System components - files, classes, functions:**

    1. Client side: This subsystem will be responsible for rendering the user interface, handling user interactions, and displaying all screen pages that will be presented to the user.
       The client side will be built from components where some will represent a screen in the system and some will be auxiliary functions.
       System screens:
        - Home screen: the display screen of the system which will include a brief detail about the system and routing to the other system screens.

        - Registration/login screen: This screen will allow you to log into the system with identifying information.

        - Voting screen: This screen will display the list of topics that can be voted on and the budget distribution. Each subject will be the root of sub-topics that will contain projects (leaves) while the rest of the user can decide whether to give the entire budget to the general topic (the root) or to an intermediate topic (node) or to specific projects according to his preferences.

The amount that the citizen will distribute is the amount of the budget that the government approved for that year.

- Information screen for the projects: this screen will contain information for the projects, when the user clicks on a certain project/issue, he will receive the explanation of the project (costs, area, population, execution time, etc.). In addition, the admin will have the option to edit existing projects and add new projects.

- Statistics screen: This screen will display statistical data on the distribution of the budget (voting percentages, comparison between the current budget and the budget chosen by the people, etc.).

- Results screen: This screen will display the results of the distribution of the budget chosen by the people according to the two algorithms.

2. Server side: this subsystem will be responsible for handling logic and business rules such as voter verification, and statistical calculations, in addition the budget distribution algorithms will be implemented in this subsystem and will be operated in it.

   Departments:
   1. user:
      - ID(integer).
      - Date of birth(string).
      - First name(string).
      - Last Name(string).
      - Email (string).
      - may vote (boolean).
      - A password (string).
      - Am I a system administrator (boolean)
      - sex (string)

   2. cross section:
      - identifier (integer)
      - name (string)
      - description (string)
      - father in the tree (subject)
      - Boys in the tree (list[] node)
      - Budget amount allocated to him (double)

3. Current budget:
    - The total budget (double)
    - subject (root of tree)

4. Nodes that are projects:
    - All projects (project[])

5. Voting (budget vote):
    - Social Security number of that citizen (integer)
    - projects (the object)

6. Calculations:
    - topics (array[])

Functions:

- is allowed to vote (integer): returns boolean whether the citizen is allowed to vote.
- Calculates how many people voted out of all the citizens of the country (int amount citizen, int amount of votes)
- Dividing the amount of the budget allocated to the subject to its children until reaching the project (subject object)
- Calculation of voting statistics by gender
- Comparison between the current budget and the budget chosen directly by the people according to the division of the system ()
- Wrap function Calculate() to run the division algorithms
- Database retrievals

Algorithms:

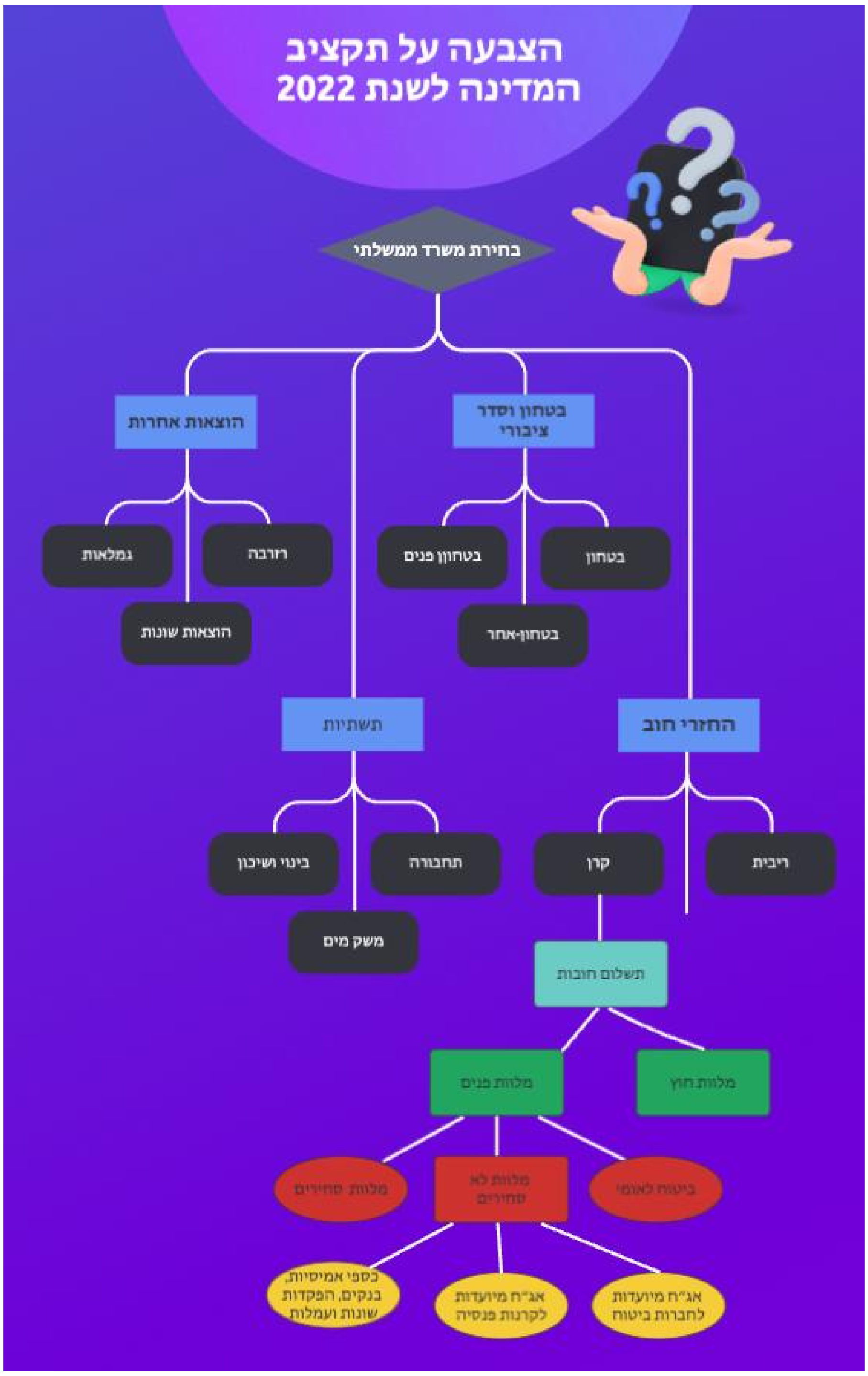
- The average algorithm: calculates the distribution of the budget for each project given by the citizens and divides by the number of voters.
- The median algorithm with linear functions: for each project, we select in advance a group of n-1 (according to the number of voters) linear functions (called continuous and increasing), we will add them to the group of voters' votes and activate the median algorithm (that is, we will arrange the votes in ascending order and choose The n/2 is rounded up).

3. Database: This sub-system will be responsible for storing and managing the information of citizens, projects, offices, voting results, and the total budget.

tables:

- **the existing budget**: a table containing the existing budget for each project.
- **The details of the citizens**: a table containing the details of the citizens stored in the system.
- **cross section**: contains the entire tree under it (topics and projects)
- **Votes**: A table containing all the votes of each project for each citizen.
- **results**: a table containing the results of the algorithms for dividing the budget.

- **Voting screen:**



<u>**legend**</u>:
Sharp rectangle - main subject.
Rounded rectangle - project.
Ellipse - rose (a sub-project under which there are no other projects).
Colors - each color indicates a different level in the tree.

<u>explanation</u>:
By clicking on a vertex (representing an office/project), the user will have two options:

1. Choosing the budget for the office/project. In this case, the budget for all the children of the same vertex will be allocated automatically, based on the ratio they received in the last published original budget.
2. Expanding the next child of the vertex (the subprojects that are under it). The user will be able to repeat this section until he reaches the top, then he will have to choose the budget.

Finally, the user will have to click on resolve the budget submission, after which he will receive a message whether he is sure he wants to submit the budget:

- If he chooses the "yes" option, the user's vote will be saved in the database and he will receive an appropriate message from the system.
- If he chooses the "No" option, the user will return to the budget selection screen.

3. <u>Input-output of each component:</u>

- Sending an object of the list of budget subjects (from the server side to the client side):

```
{
 subjects:
        {
         lvl1:
                {
                 lvl2:
                        {
                        leaf1: value,
                        leaf2: value,
                        total: value
                        },
                        total: value
                },
                total: value
        }
}
```

- Sending the voting object (from the client side to the server side):

```
{
user_id: value,
subjects:
        {
         lvl1:
                {
                 lvl2:
                        {
                        leaf1: value,
                        leaf2: value,
                        total: value
                        },
                        total: value
                },
                total: value
        }
}
```

- Getting the statistics in the dashboard (from the server side to the client side):

```
{
stats:
        {
        total_citizens: value,
        total_votes: value,
        voter_rate: value,
        num_of_men: value,
        num_of_women: value,
        men_ratio: value,
        women_ratio: value,
        votes_by_age:
                        {
                        youngs (18-29): value,
                        adults (30-49): value,
                        elderly (50+): value
                        }
                }
        }
```

- Sending user information after the registration process (from the client side to the server side):

```
{
user_id: value,
first_name: value,
last_name: value,
email: value,
password: value,
gender: value,
birth_of_date: value
}
```

- Sending the results of the budget distribution algorithms:

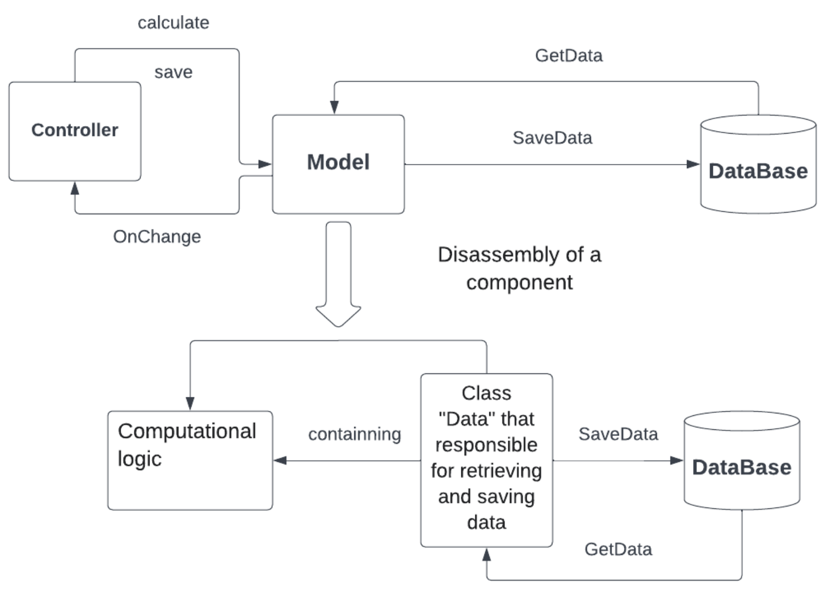
```
{
 subjects:
        {
         lvl1:
                {
                 lvl2:
                        {
                        leaf1:
                                {
                                new_value,
                                diff_from_original: value
                                },
                        leaf2:
                                {
                                new_value,
                                diff_from_original: value
                                },
                        total:
                                {
                                new_value,
                                diff_from_original: value
                                }
                        },
                    total:
                              {
                              new_value,
                              diff_from_original: value
                              }
          },
          total:
                              {
                              new_value,
                              diff_from_original: value
                              }
     }
}
```

4. **architecture**:

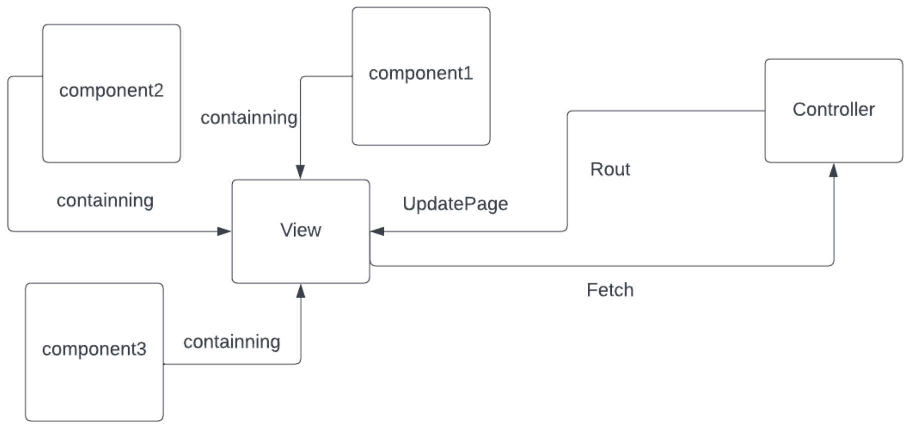
We will work in a server-client architecture where the central design pattern will be MVC as can be seen in the information flow diagram.

The data flow diagram DFD

Backend:



Front End:


component2
containning
component1
Controller
Rout
containning
View
UpdatePage
Fetch
component3
containning