Introduction

Purpose

This software design document describes the architecture and system design of a Fair Coalition Cases Distribution System. The intended audience is the coalition members that want to find a fair way to distribute the cases most equitably.

Scope

The software gets from each party of a potential coalition, its number of mandates and value for each ministry with a limited sum. The software returns a fair and envy-free division of the ministries between the parties. After the division is shown, the parties can edit their values if they are not satisfied. This software is beneficial to potential coalitions as it can show a basis of a fair division which could significantly reduce the time it takes to form a coalition.

Overview

This document describes the software design for the system which is described below .
Reference Material
https://arxiv.org/abs/1908.01669 - Efficient Fair Division with Minimal Sharing by Erel Segal-Halevi and Fedor Sandomirskiy.

Definitions and Acronyms

- Fair division - division when the value from the ministries of each political party is proportional to its size.
- Envy free division - by the values of a given political party, it didn't get less than other parties proportionally to their size.

SYSTEM OVERVIEW

The project should be implemented as a website, with a GUI which provides a user-friendly environment which passes the idea of the usage in an intuitive way. Project algorithm is located on the server side , as an input it gets the information from the user in a CSV or JSON format and calculates it, the output, is being sent to the client in the same format it got the input, which afterwards is being showed in a graphical intuitive way to the user.

Description of tools we will use to build the system

Coding languages on the client and server side:

On the side of the client we will use react.js. On the side of the server we will use Python.

Database:

To store the information we will be using the database MongoDB

Servers:

For the server we use Django.

Open source libraries:

We will be using the open source library Fairpy.

Algorithm:
From the library Fairpy we will be using the algorithms proportional allocation with bounded sharing.

System structure on the client and server sides:
System components - files, classes and functions

The classes:

- List of strings containing the name of all ministries\items.
- Political party which stores
    - The name of the party.
    - Number of mandates the party has.
    - Dictionary with the party values for each ministry.

Function:

- Proportional allocation with bounded sharing - which gets a matrix of the values from each party to each item and a list containing the number of mandates each party has. It returns a division.

- **interfaces and connections between components**

  the political parties will upload their files/input from the user interface on the website, the csv/json files we get as input will be stored in mongodb, an easy to use database, then we will process them with django framework to run python fair.py library on the web server.

- **Formats of I/O of each component**

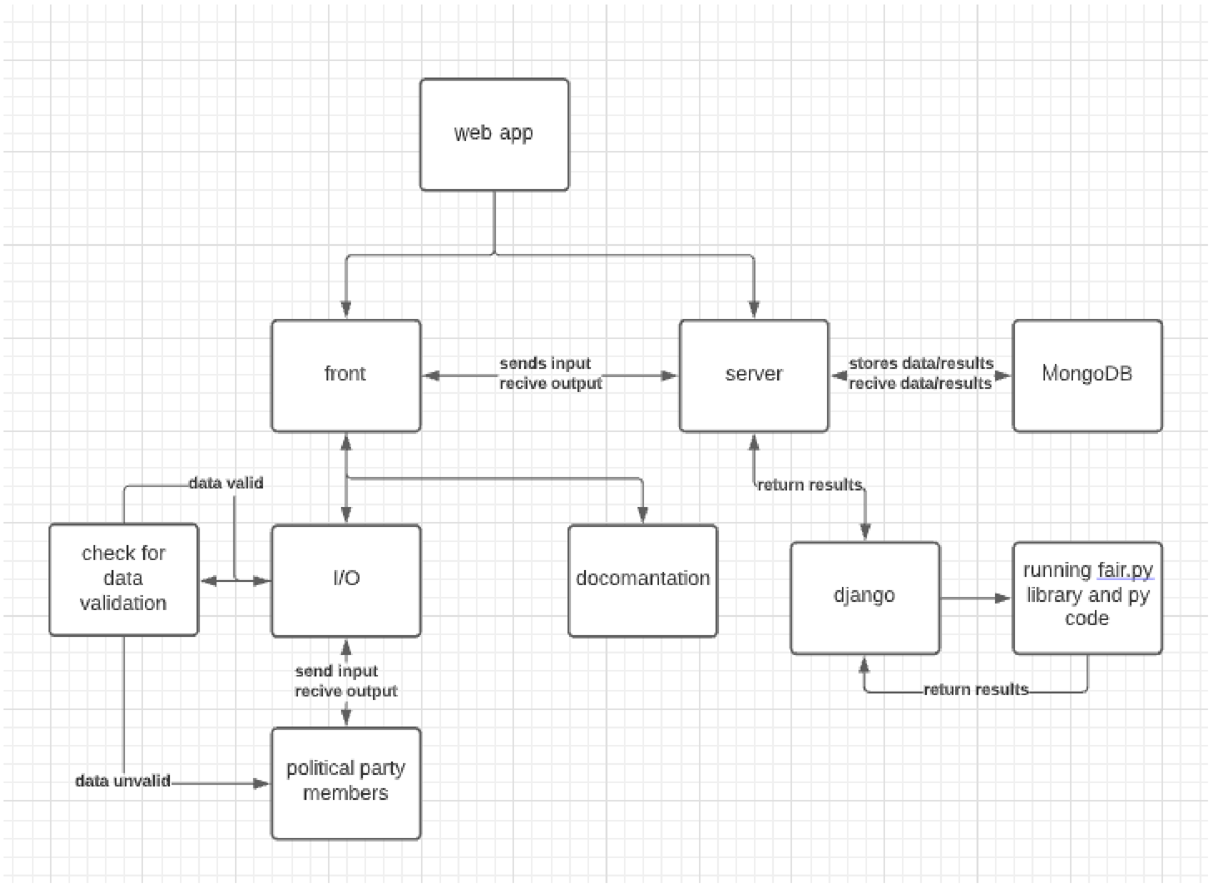
  react : will get input files or text and show output.
  mongodb: will store the inputs from the political parties for us to manipulate them
  django : will run python libs on the web server (fair.py) and output the results

- **Data structures, tables and between**

  Since we use mongodb it will be stored as a hashmap.
  fair.py uses matrix and list. matrix m*n that represent:
  [*][] - political party
  [][*] - each political object that has value
  every index in the list represents a political party and the value represents the number of mandets she achieved.

<u>SYSTEM ARCHITECTURE</u>

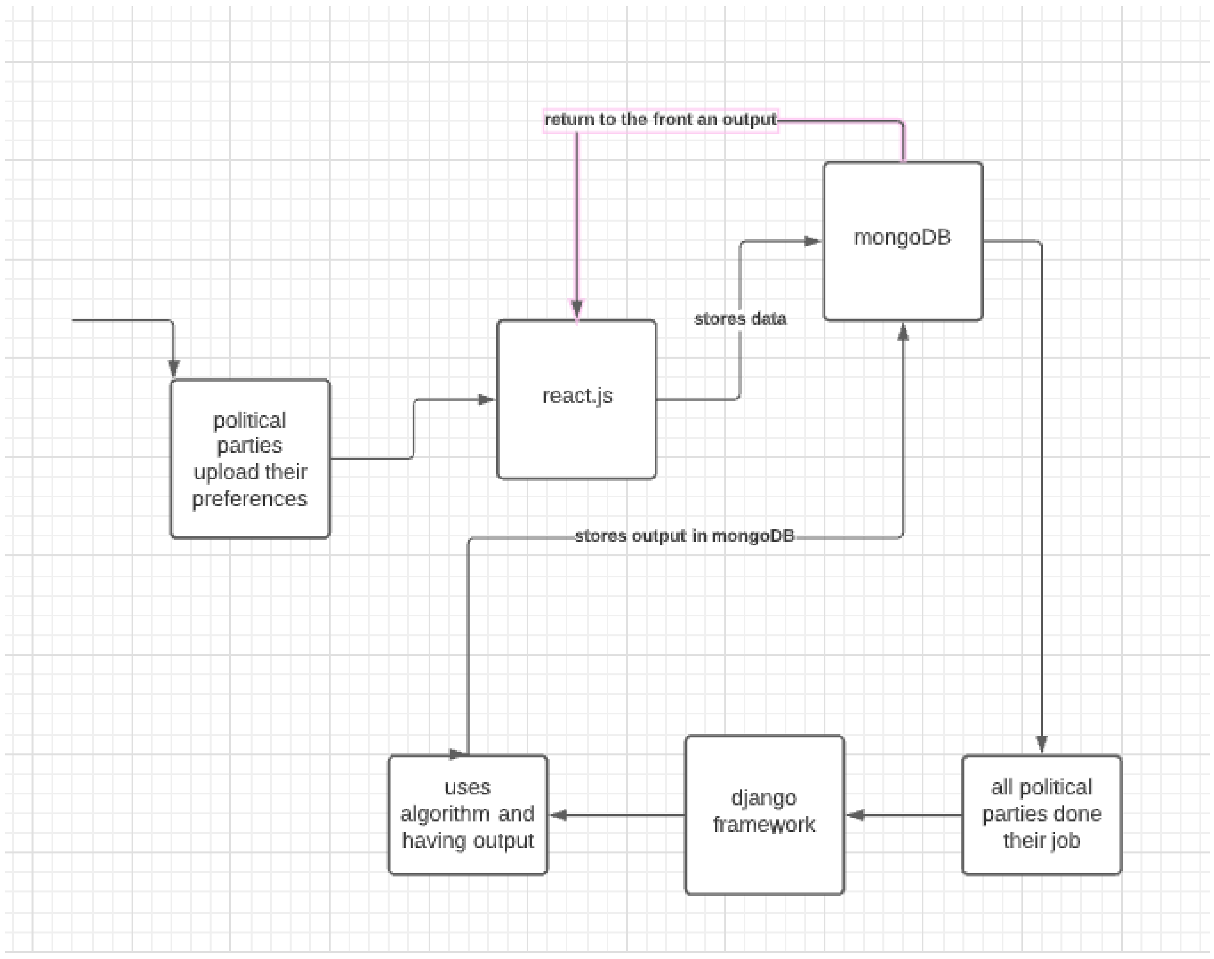
Architectural Design



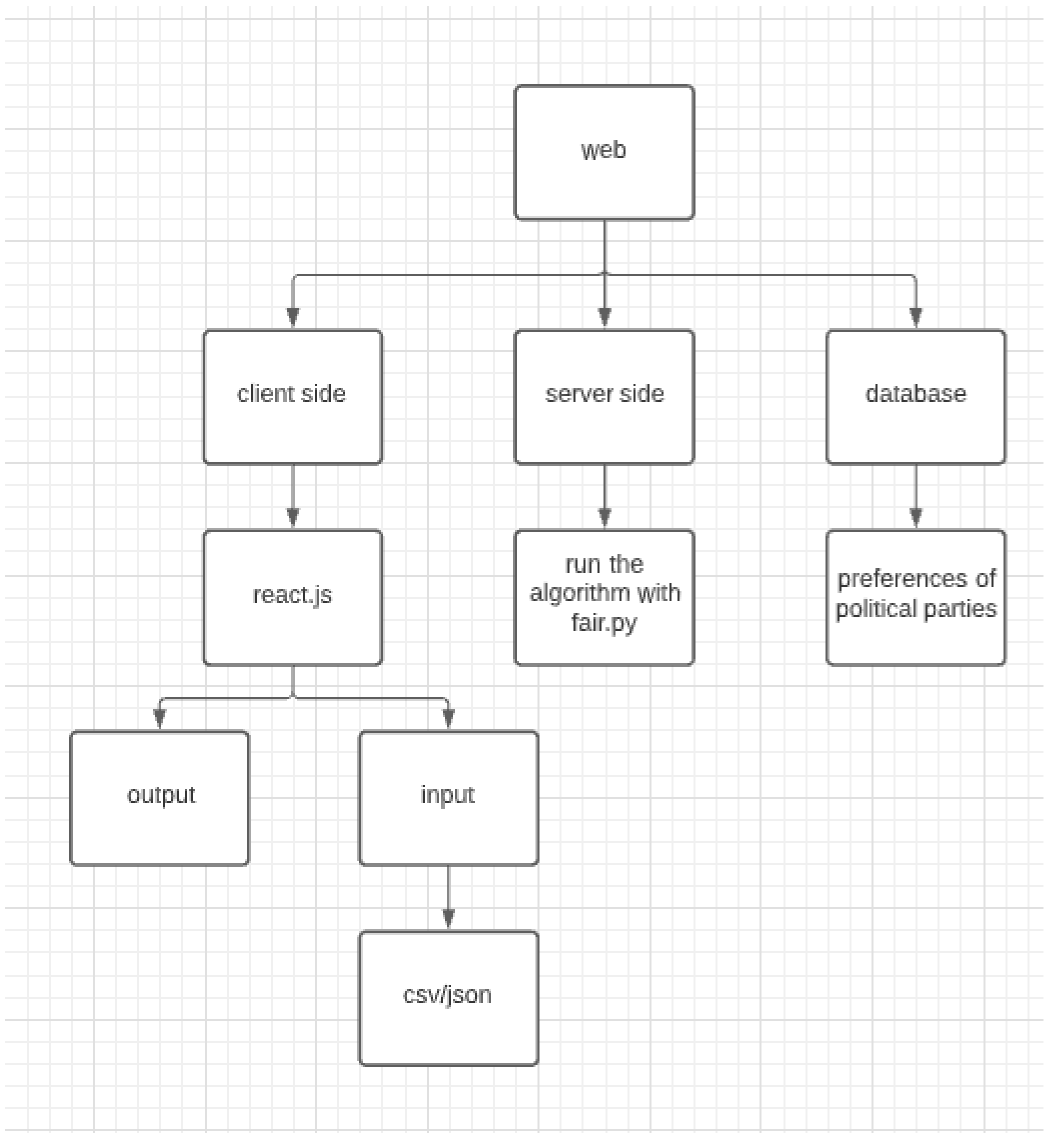
Decomposition Description

Diagrams

Data flow diagram


return to the front an output
mongoDB
stores data
react.js
political parties upload their preferences
stores output in mongoDB
uses algorithm and having output
django framework
all political parties done their job

Structural decomposition diagram:


web
client side
server side
database
react.js
run the algorithm with fair.py
preferences of political parties
output
input
csv/json

DATA DESIGN

Data Description

Each kind of the following data is fed by a csv or json file or through the GUI.

- The potential coalition parties and their numbers of mandates, it is loaded into the system. We transform the data into a class which saves for each party name, number of mandates and a dictionary which will hold the values of the ministries.
- Current ministries and save them into a list.
- From each political party, the values of each ministry. We save it into their dictionary.

In the end, we feed the values into the main algorithm. The values will be stored in mongodb.

The Algorithm in python:

The algorithm is from Fairpy written by Erel Segal-Halevi

```python
def proportional_allocation_with_bounded_sharing(instance:Any, entitlements:List=None) -> Allocation:

v = ValuationMatrix(instance)
if entitlements is None:
    entitlements = np.ones(v.num_of_agents)
else:
    entitlements = np.array(entitlements)
entitlements = entitlements/sum(entitlements)  # normalize
thresholds = v.total_values() * entitlements
return dominating_allocation_with_bounded_sharing(v, thresholds)
```

This function gets the values matrix containing each party values for each item. It sends to dominating allocation with bounded sharing the values and a division that each party gets the number of mandates / sum of all the mandates.
in this way we guarantee that each party will get fair share by their size.

```python
def dominating_allocation_with_bounded_sharing(instance:Any, thresholds:List) -> Allocation:

v = ValuationMatrix(instance)
allocation_vars = cvxpy.Variable((v.num_of_agents, v.num_of_objects))
feasibility_constraints = [
    sum([allocation_vars[i][o] for i in v.agents()])==1
    for o in v.objects()
]
positivity_constraints = [
    allocation_vars[i][o] >= 0 for i in v.agents()
    for o in v.objects()
]
utilities = [sum([allocation_vars[i][o]*v[i][o] for o in v.objects()]) for i in v.agents()]
utility_constraints = [
    utilities[i] >= thresholds[i] for i in range(v.num_of_agents-1)
]
constraints = feasibility_constraints+positivity_constraints+utility_constraints
problem = cvxpy.Problem(cvxpy.Maximize(utilities[v.num_of_agents-1]), constraints)
solvers = [
    (cvxpy.SCIPY, {'method': 'highs-ds'}),        # Always finds a BFS
    (cvxpy.MOSEK, {"bfs":True}),                  # Always finds a BFS
    (cvxpy.OSQP, {}),                             # Default - not sure it returns a BFS
    (cvxpy.SCIPY, {}),                            # Default - not sure it returns a BFS
]
solve(problem, solvers=solvers)
if problem.status=="optimal":
    allocation_matrix = allocation_vars.value
    return allocation_matrix
else:
    raise cvxpy.SolverError(f"No optimal solution found: status is {problem.status}")
```

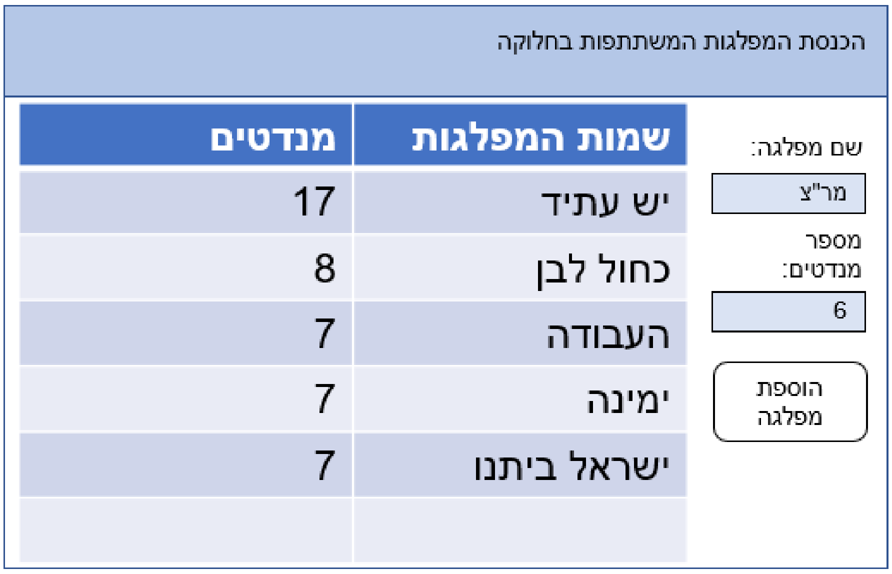
Here we take an existing allocation and use integer programming to divide the items such that each party will get at least the values it got from the first allocation, however this time only m - 1 items will be shared between the parties.

Overview of User Interface Describe the functionality of the system from the user's perspective:

The user is greeted with a window where he can provide all the political parties of the potential coalition by their name and number of mandates. After that the user can move to a screen that he can add all the ministries\items that he wants to divide. Then they are greeted with a screen with a button for each participating party, each of them presses its button and puts the values they want for each ministry\item. All inputs can be done either by the GUI or by uploading a csv\json file. After all parties have put their values and verified them, they are greeted with a screen that asks them to run the algorithm. After that they are shown a table with its rows containing the name of the party and which ministries\ items it got. If they feel unsatisfied with the results they have an option to press a button to allocate values again.

Screen Images Display screenshots showing the interface from the user's perspective

Screen for adding political parties:



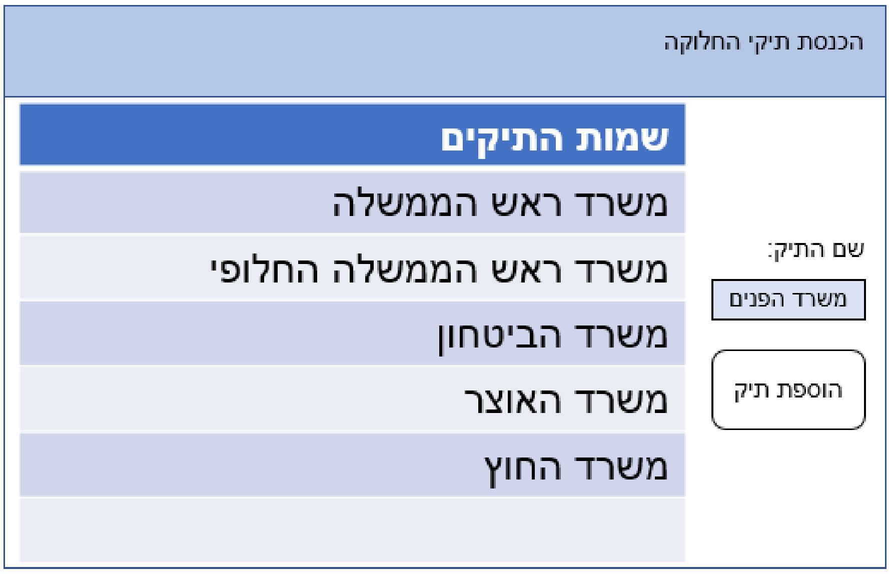
הכנסת המפלגות המשתתפות בחלוקה

שמות המפלגות	מנדטים
יש עתיד	17
כחול לבן	8
העבודה	7
ימינה	7
ישראל ביתנו	7

שם מפלגה: מר"צ

מספר מנדטים: 6

Screen for adding items for division:

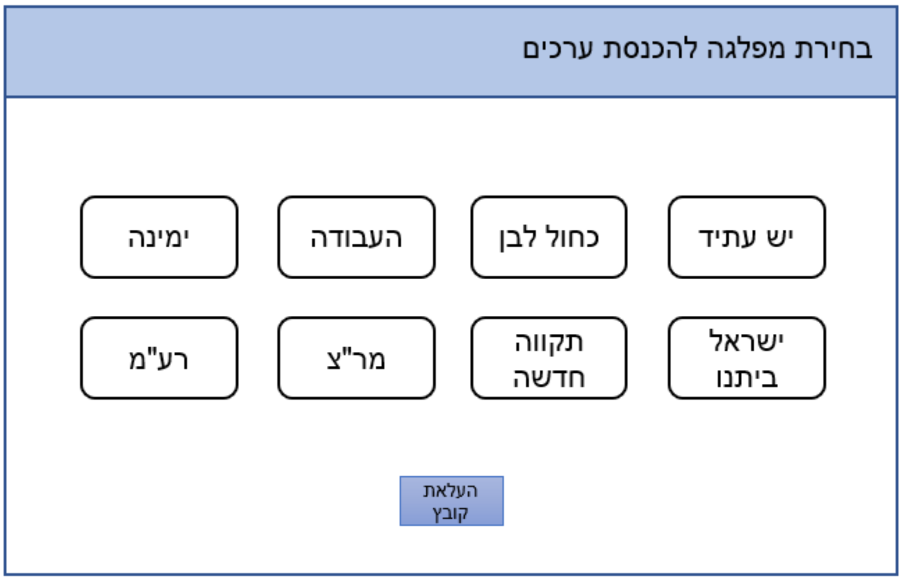

הכנסת תיקי החלוקה

שמות התיקים
משרד ראש הממשלה
משרד ראש הממשלה החלופי
משרד הביטחון
משרד האוצר
משרד החוץ

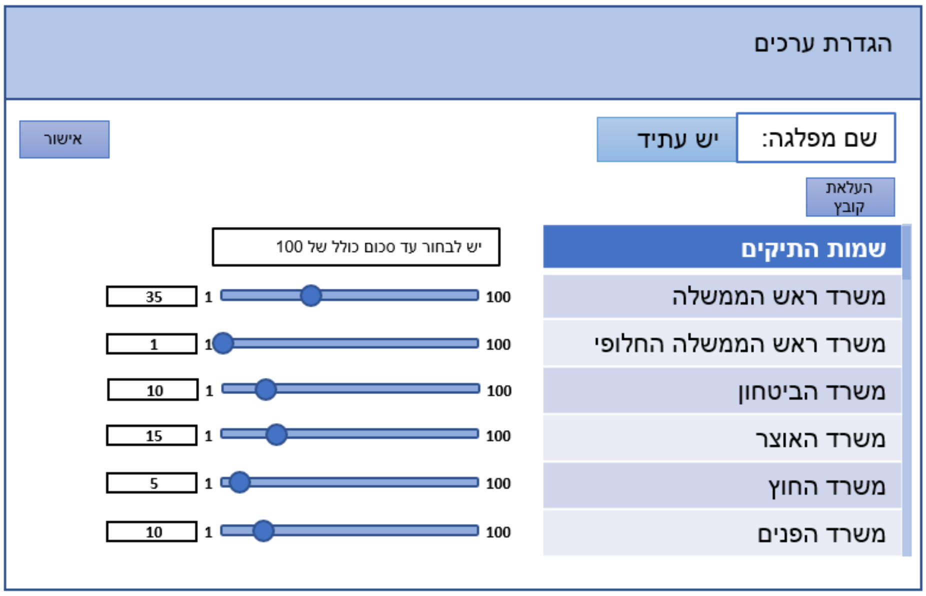
שם התיק:

משרד הפנים

Screen of buttons for each political party:


בחירת מפלגה להכנסת ערכים

Screen for setting values:

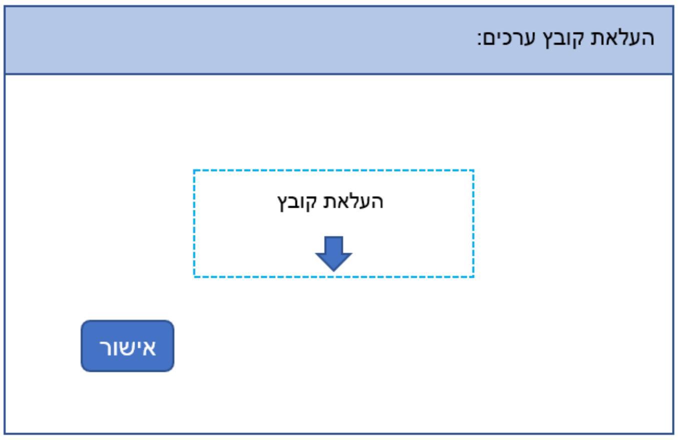


הגדרת ערכים

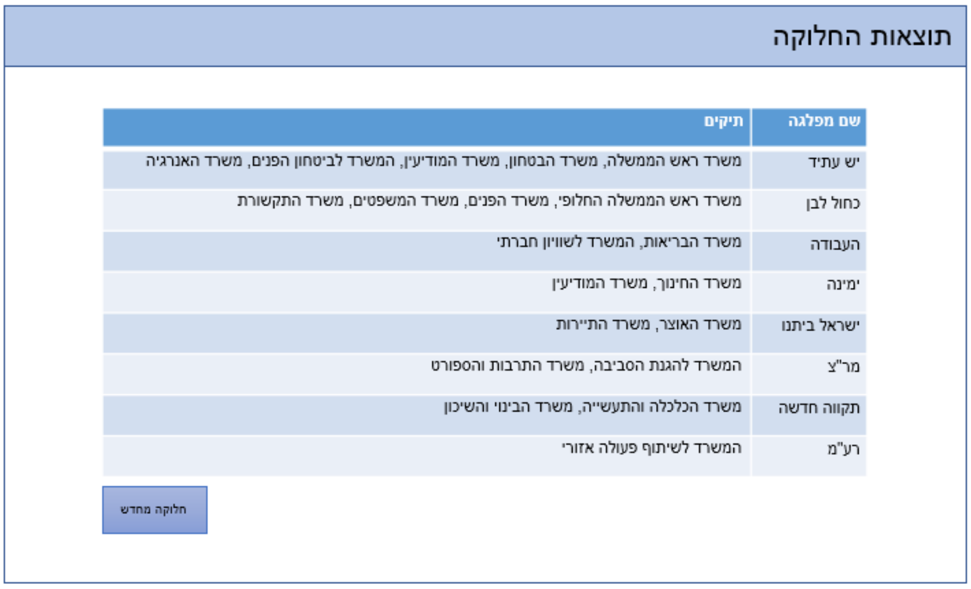
שם מפלגה: יש עתיד

שמות התיקים	ערך	מינימום	מקסימום
משרד ראש הממשלה	35	1	100
משרד ראש הממשלה החלופי	1	1	100
משרד הביטחון	10	1	100
משרד האוצר	15	1	100
משרד החוץ	5	1	100
משרד הפנים	10	1	100

Screen for uploading a file:



Results screen:



תוצאות החלוקה

שם מפלגה	תיקים
יש עתיד	משרד ראש הממשלה, משרד הבטחון, משרד המודיעין, המשרד לביטחון הפנים, משרד האנרגיה
כחול לבן	משרד ראש הממשלה החלופי, משרד הפנים, משרד המשפטים, משרד התקשורת
העבודה	משרד הבריאות, המשרד לשוויון חברתי
ימינה	משרד החינוך, משרד המודיעין
ישראל ביתנו	משרד האוצר, משרד התיירות
מר"צ	המשרד להגנת הסביבה, משרד התרבות והספורט
תקווה חדשה	משרד הכלכלה והתעשייה, משרד הבינוי והשיכון
רע"מ	המשרד לשיתוף פעולה אזורי

<u>Division of the tasks in the project</u>

Andrey Bakhrakh - Connect the algorithm from fairpy to our project and make sure it works as intended.
Aviv Danino - On the server side with django.
Victor Kushnir - On client side with react.js making the GUI.