division algorithm

Software Design

Daniel Abergel – 315660712

Eliahu Satat – 204395644

Netanel Ben-Isahar -204478150

Yarden Gaon – 312469174

Nahashon Satat – 315823880

Elyashiv Deri –

Date: 03/17/2020

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of a distribution algorithm site, this document is intended for the developers and our Project Manager.

1.2 Scope

The system will receive an agent's estimates for all objects and return a distribution (proportional or envy-free result).  
Proportional: An allocation z is proportional (Prop) if each agent prefers her parts to the equal division: 1/n from the sum of all the Objects.   
Envy-free: An allocation z is called envy-free (EF) if every agent prefers her parts to the parts of the others.

1.3 Overview

The SDD document contains details about our products and system design information. See table of contents1.4 Reference Material

1.4 Reference

This site is base on the algorithm in this Article by erel segal - https://arxiv.org/pdf/1908.01669.pdf

1.5 Definitions and Acronyms

Proportional: An allocation z is proportional (Prop) if each agent prefers her parts to the equal division: 1/n from the sum of all the Objects.   
Envy-free: An allocation z is called envy-free (EF) if every agent prefers her parts to the parts of the others.

2. SYSTEM OVERVIEW

The project is divided into two main parts: client-side and server-side.

client-side is a website, The main goal of the client-side is to interact with clients, handling the client's input and output. client-side has been written in JavaScript HTML and CSS.

The server-side goal is to implement a Distribution-Algorithm and handle the requests that came from the website (clients) .

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

A close up of a map

Description automatically generated

3.2 DecompositionDescription

The algorithm objects:  
**valuation**: this class represents all the ratios between the agents.  
**GraphGenerator**: this class functionality is to generate all the possibilities graph (represent as ConsumptionGraph) for some valuation\_matrix.  
**FairProportionalAllocationProblem**: this class solves Fair Proportional Allocation Problem she is inherited from FairAllocationProblem.  
**FairEnvyFreeAllocationProblem**: this class solves Fair Envy Free Allocation Problem she is inherited from FairAllocationProblem.  
**FairAllocationProblem**: this class is an abstract class to solve the Fair Allocation Problem meaning - get agents valuation and a Fair Allocation  
**ConsumptionGraph**: this class represent a graph of consumption of the agents represent by binary matrix - if graph[i][j] = 1 its mean that agent I consumption the j object.  
**Allocation**: this class represent an Allocation of the object to the agents represent by matrix - if graph[i][j] = x its mean that agent I gets x% from object j 0 <= graph[i][j] <= 1.  
**Server:** This class represent the server, supports the website request (RESTful API)

3.3 Design Rationale

We choose this architecture because we want to give people the opportunity to try our algorithm, and the best way to reach a lot of people to test this algorithm is a website on the Internet.  
Our goal is to separate the backend from the frontend this is the best architecture to achieve the goal.  
We have thought about publishing the algorithm as an open-source library but we want to test the algorithm with a lot of scenarios, because of this we have decided to go with the website architecture (website (front) + server (backend))

4. DATA DESIGN  
4.1 Data Description

We don't have any database, our data is singular matrix transforms in a JSON form

4.2 Data Dictionary

-

5. COMPONENT DESIGN

The algorithm objects and function:

ValueRatio:

*Functionality:*

**def** create\_the\_value\_ratio\_for\_2(self, consumption\_graph, x, y)

*this function build the array for value ratio between agent x to agent y  
according to the given graph and the properties of agent x in this graph  
and sort it*

**def** buile\_all\_the\_ratio(valuation\_matrix)

*this function create a list of matrix .  
each matrix is the ratio between agent i and all the  
ether agents  
for example ans[3] = matrix of the ratio between agent 3 and all ether agents  
so ans[3][4] = the ratio array between agent 3 to agent 4*

GraphGenerator :

*Functionality:*

generate\_all\_consumption\_graph(self)

*this is the main function in that part of the algorithm  
she get valuation and generate all the possibilities graph for it  
represent as ConsumptionGraph***:return***: generator of all possibilities graph*

add\_agent(self,genneretor,i)

*this function get generator for all the the graph for i-1 agent  
and generate all the graph for adding agent i***:param** *genneretor: generator for the all the graph for i-1 agents***:param** *i: the index for the new agent***:return***: generator for the all the graph for i agents*

add\_agent\_to\_graph(self, consumption\_graph: ConsumptionGraph)

**:param** *consumption\_graph: : some given ConsumptionGraph that represent agent and there properties***:return***: generator for the all the graphs from adding agent i to the given graph*

code\_to\_consumption\_graph(self, consumption\_graph: ConsumptionGraph, code)

*this function take code that represent new graph and convert  
it to that consumption\_graph  
the calculation of the properties of each agent is p[i]/2 from the end of arr belongs to the new agent  
and len(arr)-p[i]/2 from the start of arr belongs to agent i***:param** *consumption\_graph: the original graph***:param** *code:the code in form (x1,x2...xi) i = the number of agent in graph, xi in range(number of properties of  
 agent i in graph***:return***: consumption\_graph for that code*

FairProportionalAllocationProblem:

*Functionality:*

find\_allocation\_with\_min\_shering(self)

*this function find the proportional allocation for the valuation***:return***: the Proportional allocation*

find\_allocation\_for\_graph(self,consumption\_graph : ConsumptionGraph)

*this function get a consumption graph and use cvxpy to solve  
the convex problem to find a proportional allocation.  
the condition for the convex problem is:  
1) each alloc[i][j] >=0 - an agent cant get minus pesent  
from some item  
2) if consumption\_graph[i][j] == 0 so alloc[i][j]= 0 .  
if in the current consumption graph the agent i doesnt consume the item j  
so in the allocation he is get 0% from this item  
3) the proportional condition (by definition)  
4) the sum of every column in the allocation == 1  
each item divided exactly to 100 percent  
and after solving the problem - check if the result are better from the  
"min\_sharing\_allocation" (meaning if the current allocation as lass shering from "min\_sharing\_allocation")  
and update it***:param** *consumption\_graph: some given consumption graph***:return***: update "min\_sharing\_allocation"*

FairEnvyFreeAllocationProblem:

*Functionality:*

**def** find\_allocation\_with\_min\_shering(self)

*this function find the envy free allocation for the valuation***:return***: the envy free allocation*

find\_allocation\_for\_graph(self,consumption\_graph : ConsumptionGraph)

*this function get a consumption graph and use cvxpy to solve  
the convex problem to find a envy free allocation.  
the condition for the convex problem is:  
1) each alloc[i][j] >=0 - an agent cant get minus pesent  
from some item  
2) if consumption\_graph[i][j] == 0 so alloc[i][j]= 0 .  
if in the current consumption graph the agent i doesnt consume the item j  
so in the allocation he is get 0% from this item  
3) the envy free condition (by definition)  
4) the sum of every column in the allocation == 1  
each item divided exactly to 100 percent  
and after solving the problem - check if the result are better from the  
"min\_sharing\_allocation" (meaning if the current allocation as lass shering from "min\_sharing\_allocation")  
and update it***:param** *consumption\_graph: some given consumption graph***:return***: update "min\_sharing\_allocation"*

FairAllocationProblem :

*Functionality: abstract*

ConsumptionGraph:

*Functionality:*

**def** num\_of\_sharing(self)

*this function return the number of  
sharing in the ConsumptionGraph*

generate\_all\_code(self)

*this function generate all the codes for that graph  
(the code represent the new graph that can built from this graph and adding new agent)***:return***: generator for all the codes*

sum\_of\_agent\_properties(self)

*this function return array that each arr[i] = the number  
of properties of agent i in graph multiple by 2 plus 1***:return***: the number of properties of each agent in array*

Allocation:

*Functionality:*

num\_of\_shering(self)

*this function calculate the number of sharing in  
the allocation***:return***: the number of sharing*

**def** round(self)

*this function round the alloction\_matrix for 3 digit after the point*

APP:

*Functionality:*

**:return***: the home page Home.html*

**def** home()

*this function returns the home page Home.html*

**:return***: the home page Home.html*

**def** send\_dom(dom)

*this function returns any DOM objects*

**:return***: the dom file the function receives in the function parameter*

**def** send\_css(css)

*this function returns any CSS objects*

**:return***: the css file the function receives in the function parameter*

**def** send\_images(image)

*this function returns any IMAGE objects*

**:return***: the image file the function receives in the function parameter*

**class** Algorithm(Resource):

(this functions is RESTful API using flask-RESTful)

**def** get(self)

*return {‘Algorithm’ : ‘available’}*

**def** post(self)

*receive: JSON includes matrix for the algorithm*

*return: TIME OUT or The algorithm result as a JSON file .*

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

Our website is a web application that is used as the user interface. Our website has a main page that consists the important details that the user should know. From that page, the user can navigate to other pages; e.g. "who we are", "contact us" pages and so on.   
Another navigation option from the main page is to navigate to "first step" page by clicking on the home button.  
The "first step" page is the most important page in our website and it is responsible for the data insertion by the user.

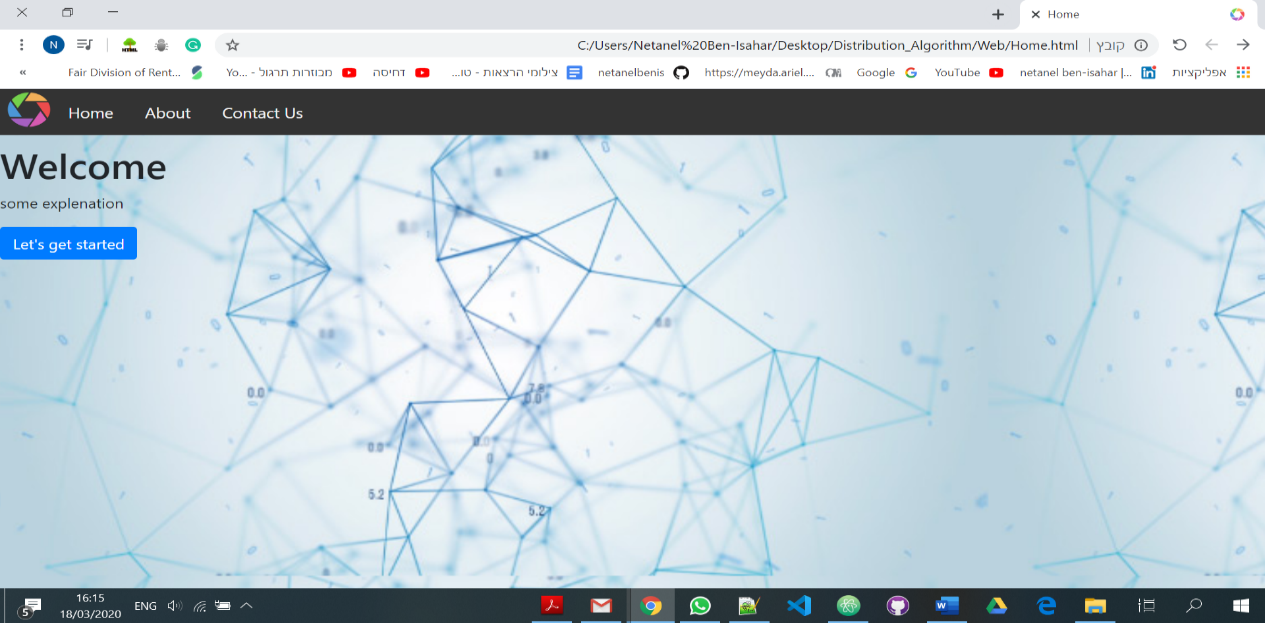
First, the user inserts the names of the participants (separated by commas) and the objects (separated by commas).  
After clicking on the continue button, the user gets few more text boxes(number of participants\*number of objects) every box represents the rate of every object by every participant.  
Next, the data insertion is completed and clicking the "send" button will insert the data into a matrix and send it to the server.  
Lastly, the server performs calculations, it then sends a matrix as a result and the user will have the results showed as graphs (pie charts) for every participant.

6.2 Screen Images

Front End:

The next images were taken from the half-built web application.

**Home page**



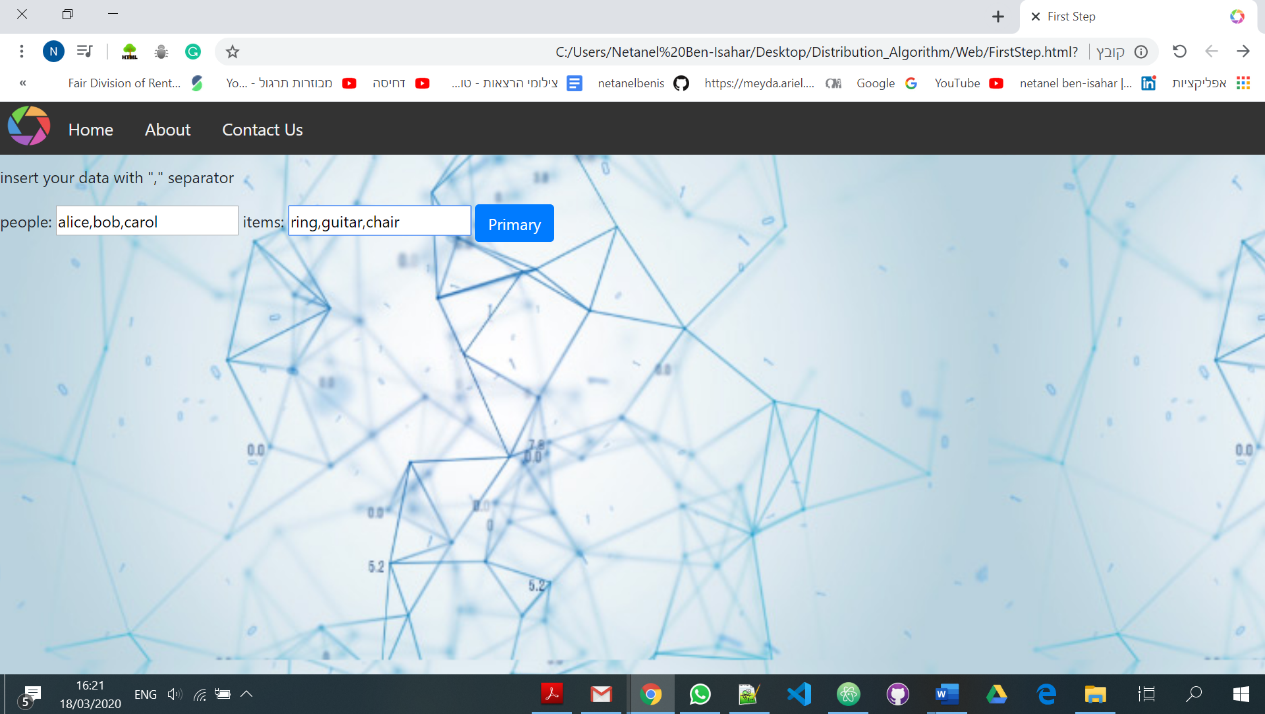
Buttons for returning to the home page

Navigation to the "about us" page

Navigation to the "Contact Us" page

Navigation bar

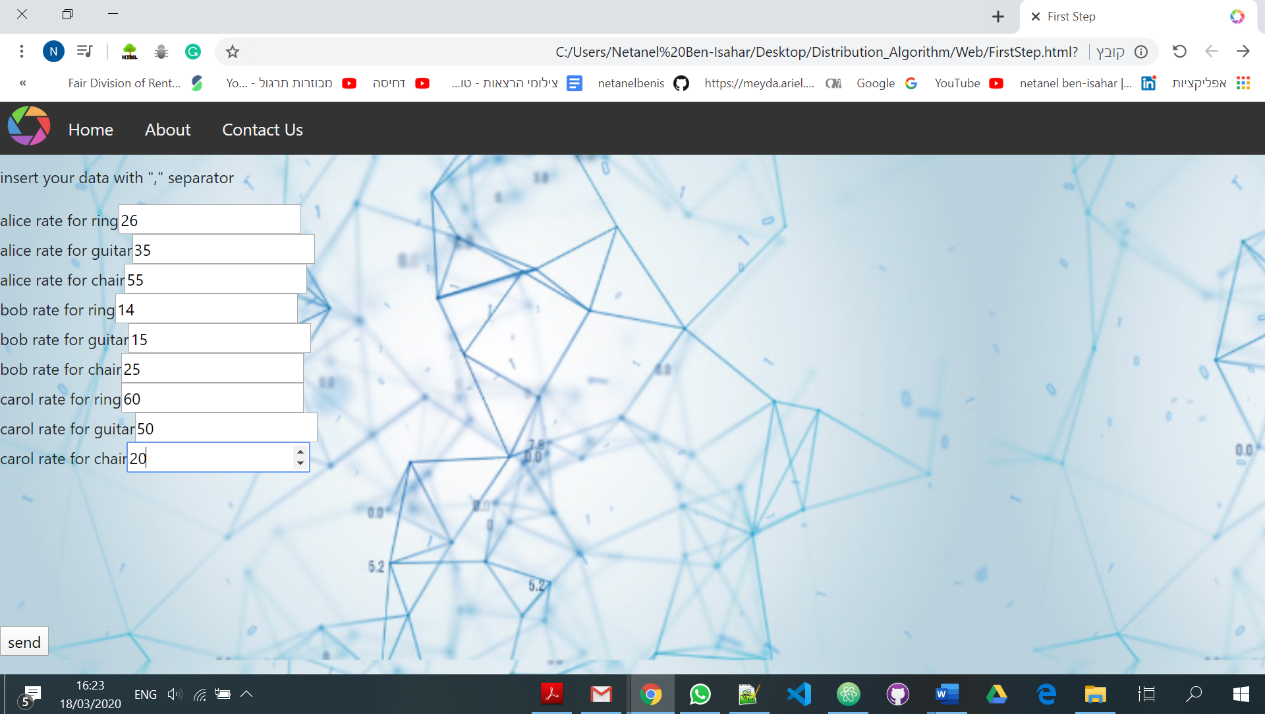
Navigate to the "algorithm section"



Inserting names of participants

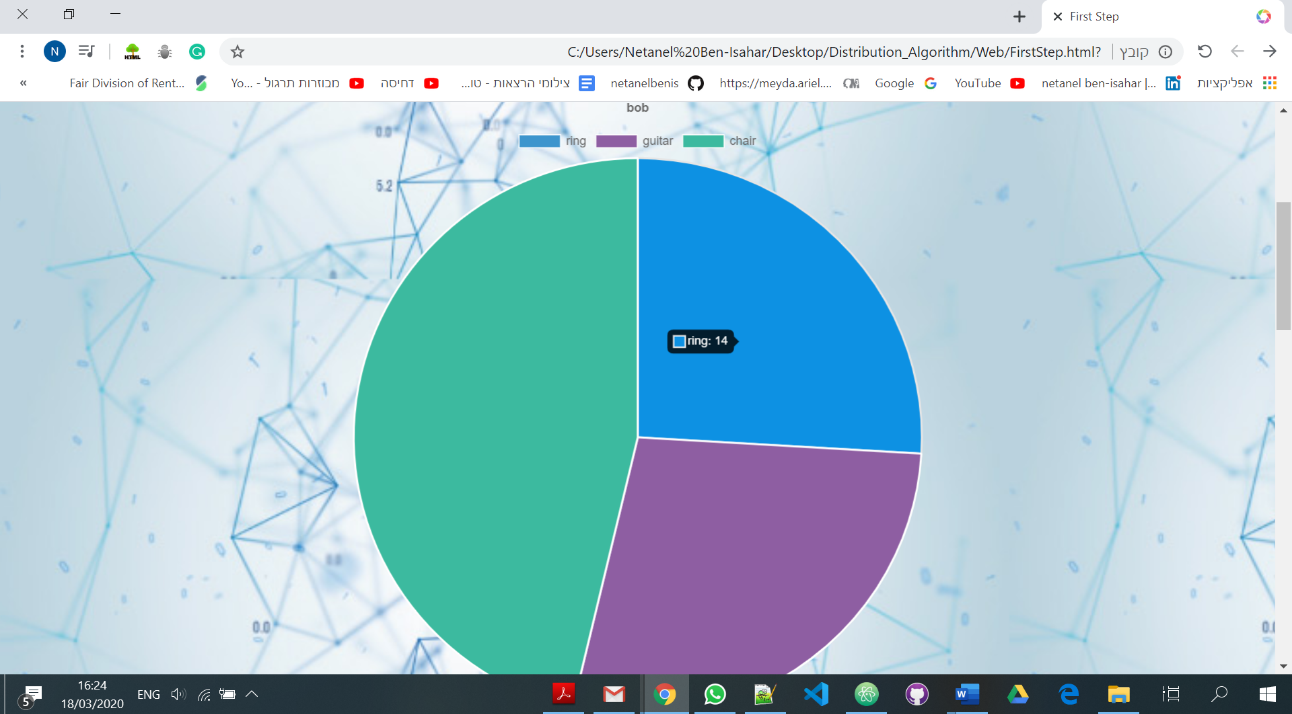
Inserting names of objects

Activation button



Every participant rates every object

Send button



Every participant gets his own chart that represent the percent of every object that he gets

7. REQUIREMENTS MATRIX

This table contains the requirements from the SRS document and all the components that respond to the requirements

|  |  |
| --- | --- |
| The server will receive a matrix(int) as input that represents the participant's estimates for items. | The server receives a JSON format that includes the matrix using RESTful API – app.py |
| 2. The system will know to round the answer up to three digits after the decimal point | Allocation |
| 3. The system will know how to produce all possible allocations for each graph | GraphGenerator |
| 4. The system will know how to find the allocation with the minimum number of shares | FairAllocationProblem |
| 5. The system will know how to produce all possible allocations for each agent | GraphGenerator |
| 6. The system will know how to solve a proportionality problem | FairProportionalAllocationProblem |
| 7. The system will know how to solve the envy-free problem | FairEnvyFreeAllocationProblem |
| 8. The system will know how to produce all the ratio between the agents' assessments | ValueRatio |
| 9. The system will know how to produce all the ratio between the agents' assessments | ValueRatio |
| 10. The system will know how many shares of agents among the agents there are in each allocation | Allocation |
| 11. The system will know how to solve the value optimization problem for each assignment | FairAllocationProblem |
| 12. The system will know how to produce all possible allocations for each evaluation of agent objects. | GraphGenerator |
| 13. The system will know to represent each graph as a sequence of single-value numbers | ConsumptionGraph |
| 14. The system will know to pass any code from section 13 to the matrix. | GraphGenerator |
| 15. The website should handle matrix inputs up to size 5 \* 5 and 4 \* 20 | The web will restrict the input data to the valid inputs. |
| 16. The system may also handle inputs beyond the size specified in the previous section and up to 5 \* 20 | If the system can handle a larger input so the input limits will updated as well. |
| 17. The system may only produce graphs with a small number of washes from the number of agents | NONE relevant |
|  |  |
| 18. The site will be able to send emails if the answer is delayed for more than a minute | We will use with an open source API to send those emails if the results will delayed. |
| 19. The site will contain all the information the user needs to know about the algorithm (answer, settings, etc.) | The user will have, in the main page, all of the relevant information that he need. |
| 20. The site will contain a page that will allow the user to submit the information to the server to calculate the algorithm. | Known as the "first step" page. |
| 21. The server can handle several requests at the same time. | uWSGI + nginx + apache . |
| 22. The server knew how to handle several types of API that come from the site. | Flask API |
| 23 The server knew how to deal with a timeout problem when the request passed the minute. | Still on progress server + website + algorithm design |