Seat Allocation Portal

A CS251 Report by Group 31

Pradyot Prakash
130050008
pp2105@gmail.com

Chandra Maloo
130050009
maloochandra@gmail.com

Shantanu Thakoor
130050008
shanu.thakoor@gmail.com

IIT Bombay

Team 31 : !false

Introduction

This is a software used for Allocation of seats of various engineering programmes to students based on their performance in a common entrance examination - JEE. The software takes the Merit list containing various ranks of all the students and their choices of preference of the programs. On the contrary to what it seems this not an easy task. We have used two algorithms for the same. First one is quite simple one assuming this to be an easy job. But as it turns out this is not a fair algorithm. So we have used the well known Gale Shapeley algorithm for a fair allocation of seats. We have also created a web portal using Django framework for the same. On this portal each user has an account and he/she can login to fill in the preferences of his choice. Later on, when the allocation has been done then the student can see which college has been allotted to him. Based on previous year ranks we are also suggesting the possible programmes that a student might get admitted to based on previous year opening and closing ranks.

Introduction

This is a software used for Allocation of seats of various engineering programmes to students based on their performance in a common entrance examination - JEE. The software takes the Merit list containing various ranks of all the students and their choices of preference of the programs. On the contrary to what it seems this not an easy task. We have used two algorithms for the same. First one is quite simple one assuming this to be an easy job. But as it turns out this is not a fair algorithm. So we have used the well known Gale Shapeley algorithm for a fair allocation of seats. We have also created a web portal using Django framework for the same. On this portal each user has an account and he/she can login to fill in the preferences of his choice. Later on, when the allocation has been done then the student can see which college has been allotted to him. Based on previous year ranks we are also suggesting the possible programmes that a student might get admitted to based on previous year opening and closing ranks.

Introduction

This is a software used for Allocation of seats of various engineering programmes to students based on their performance in a common entrance examination - JEE. The software takes the Merit list containing various ranks of all the students and their choices of preference of the programs. On the contrary to what it seems this not an easy task. We have used two algorithms for the same. First one is quite simple one assuming this to be an easy job. But as it turns out this is not a fair algorithm. So we have used the well known Gale Shapeley algorithm for a fair allocation of seats. We have also created a web portal using Django framework for the same. On this portal each user has an account and he/she can login to fill in the preferences of his choice. Later on, when the allocation has been done then the student can see which college has been allotted to him. Based on previous year ranks we are also suggesting the possible programmes that a student might get admitted to based on previous year opening and closing ranks.

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

Gale Shapeley Algorithm GaleShapely algorithm is pathbreaking but simple in its conception and implementation. Essentially, it goes as follows:

- Every candidate applies to the first program on its preference list
- Every program rejects certain candidates and waitlists the others based on capacity and the meritlist it uses
- Rejected candidates move one index further along their preference list, waitlisted candidates stay on the same index
- Algorithm terminates when either all candidates reach the end of their preference list, or no candidate is rejected by any program when it filters its applications
- When the algorithm terminates, a candidate is alloted the program that (s)he is currently waitlisted for; it (s)he is not waitlisted for anything, (s)he does not get alloted a seat
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Seat Allocation Algorithms

MeritOrder algorithm is also simple; perhaps a tad too simple. It follows a greedy algorithm approach, allotting candidates the best seat it possibly can the first time it sees them, and never looks at them again. This algorithm is not fair to all the candidates

- Arrange the candidates in order of their ranks
- Arrange the respective ranklists based on their category
- Start from the first Candidate
- For every candidate, go through his/her list of preferences in descending order until we find one that (s)he is eligible for; assign him/her that course
- If the candidate is not eligible for any course, (s)he is not alloted any seat
- Go to the next candidate
- Algorithm terminates when all the candidates are thus processed
- Only minimal changes are needed to serve needs of DS and Foreign national student allocations

Package Structure - code

code/common/

This has the following common classes used by both the algorithms:

Constants.class

Contains all the constant variables names used anywhere in the code. For e.g. male="MALE", gen=0, etc.

Candidates.class

All the properties of a particular student are stored in an object of this class. It has all the attributes like rank, category, roll number, etc.

VirtualProgram.class

All the programs in various colleges are stored in an object of this class. It has attributes which store the number of seats available in different categories for the program, course code, etc.

MeritList.class

Contains the ranks of all the students of different categories

Package Structure - code

code/common/
This has the following common classes used by both the algorithms:
Constants.class
Contains all the constant variables names used anywhere in the code. For e.g. male="MALE", gen=0, etc.
Candidates.class
All the properties of a particular student are stored in an object of this class. It has all the attributes like rank, category, roll number, etc.
VirtualProgram.class
All the programs in various colleges are stored in an object of this class. It has attributes which store the number of seats available in different categories for the program, course code, etc.
MeritList.class
Contains the ranks of all the students of different categories

Package Structure - code

code/common/
This has the following common classes used by both the algorithms:
Constants.class
Contains all the constant variables names used anywhere in the code.
For e.g. male="MALE", gen=0, etc.
Candidates.class
All the properties of a particular student are stored in an object of this class. It has all the attributes like rank, category, roll number, etc.
VirtualProgram.class
All the programs in various colleges are stored in an object of this class. It has attributes which store the number of seats available in different categories for the program, course code, etc.
MeritList.class
Contains the ranks of all the students of different categories

Package Structure - code

code/common/

This has the following common classes used by both the algorithms:

Constants.class

Contains all the constant variables names used anywhere in the code. For e.g. male="MALE", gen=0, etc.

Candidates.class

All the properties of a particular student are stored in an object of this class. It has all the attributes like rank, category, roll number, etc.

VirtualProgram.class

All the programs in various colleges are stored in an object of this class. It has attributes which store the number of seats available in different categories for the program, course code, etc.

MeritList.class

Contains the ranks of all the students of different categories

Package Structure - code

code/common/

This has the following common classes used by both the algorithms:

Constants.class

Contains all the constant variables names used anywhere in the code. For e.g. male="MALE", gen=0, etc.

Candidates.class

All the properties of a particular student are stored in an object of this class. It has all the attributes like rank, category, roll number, etc.

VirtualProgram.class

All the programs in various colleges are stored in an object of this class. It has attributes which store the number of seats available in different categories for the program, course code, etc.

MeritList.class

Contains the ranks of all the students of different categories

Package Structure - code

code/common/

This has the following common classes used by both the algorithms:

Constants.class

Contains all the constant variables names used anywhere in the code. For e.g. male="MALE", gen=0, etc.

Candidates.class

All the properties of a particular student are stored in an object of this class. It has all the attributes like rank, category, roll number, etc.

VirtualProgram.class

All the programs in various colleges are stored in an object of this class. It has attributes which store the number of seats available in different categories for the program, course code, etc.

MeritList.class

Contains the ranks of all the students of different categories

Package Structure - code

code/gale/

GaleShapeleyAdmission.class

This class runs the entire GS algorithm and generates the corresponding output files

code/merit/

MeritOrderAdmission.class

This class runs the entire Merit Order algorithm and generates the corresponding output files

code/main/

Main.class

This class makes call to respective classes as and when needed and gets the entire allocation procedure executed

Package Structure - code

code/gale/
GaleShapeleyAdmission.class
This class runs the entire GS algorithm and generates the corresponding output files
code/merit/
MeritOrderAdmission.class
This class runs the entire Merit Order algorithm and generates the corresponding output files
code/main/
Main.class
This class makes call to respective classes as and when needed and gets the entire allocation procedure executed

Package Structure - code

code/gale/
GaleShapeleyAdmission.class
This class runs the entire GS algorithm and generates the corresponding output files
code/merit/
MeritOrderAdmission.class
This class runs the entire Merit Order algorithm and generates the corresponding output files
code/main/
Main.class
This class makes call to respective classes as and when needed and gets the entire allocation procedure executed

Package Structure - code

code/gale/
GaleShapeleyAdmission.class
This class runs the entire GS algorithm and generates the corresponding output files
code/merit/
MeritOrderAdmission.class
This class runs the entire Merit Order algorithm and generates the corresponding output files
code/main/
Main.class
This class makes call to respective classes as and when needed and gets the entire allocation procedure executed

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1,2,3,4,...,100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv, MeritOrderAdmission.csv, GaleShapeleyAdmissionPretty.csv, MeritOrderAdmissionPretty.csv are the outputs generated

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 .. 100, then you can also run as : $ java code.Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 ... 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 ... 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 .. 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 ... 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 .. 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated.

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 .. 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated.

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 .. 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated.

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 .. 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated.

How to use

Allocation Part - Java

Base folder(base/) contains the following: code, doc, test_cases, Makefile

Compiling

- Go to the base folder : $ cd base/
- Type into the command line : $ make all

Running the program

- Go to the base folder : $ cd base/
- Type into the command line, folder_name is the name of the folder in test_cases : $ java code.Main.Main folder_name
- Here if you have various testcases with successive integers as folder names, e.g. 1 2 3 4 .. 100, then you can also run as : $ java Main.Main 1 100
- Go to the folder : $ cd test_cases/folder_name
- GaleShapeleyAdmission.csv MeritOrderAdmission.csv GaleShapeleyAdmissionPretty.csv MeritOrderAdmissionPretty.csv are the outputs generated.

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command 'make all'

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command "make all"

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command 'make all'

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation which were created when you executed the command "make all"

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command 'make all'

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command "make all"

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command "make all"

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command 'make all'

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command 'make all'

How to use

Adding your own testcases

- Go to the test_cases folder : $ cd base/test_cases/
- Create a directory : $ mkdir folder_name
- Go the created directory : $ cd folder_name
- Create the following three files : $ touch ranklist.csv choices.csv programs.csv
- Populate the above three files in the prescribed format (refer given testcases)

Cleaning up...

- Go to the base folder : $ base/
- Type into the command line : $ make clean
- This removes all the .class files and also the java documentation, which were created when you executed the command 'make all'

Parsing the pdf

update.py

- This file is present in the base directory of lab11. This preogram reads 'programmeCode.pdf' and generates the data_u-2012.csv which contains the branch to their code mapping.
- This program uses the shell script 'pdftotext' and generates the text output 'temp' from which the program further reads and populates the csv file.
- This csv file is very essential to the working of the further project.

Parsing the pdf

update.py

- This file is present in the base directory of lab11. This preogram reads 'programmeCode.pdf' and generates the data_u-2012.csv which contains the branch to their code mapping.
- This program uses the shell script 'pdftotext' and generates the text output 'temp' from which the program further reads and populates the csv file.
- This csv file is very essential to the working of the further project.

Parsing the pdf

update.py

- This file is present in the base directory of lab11. This preogram reads 'programmeCode.pdf' and generates the data_u-2012.csv which contains the branch to their code mapping.
- This program uses the shell script 'pdftotext' and generates the text output 'temp' from which the program further reads and populates the csv file.
- This csv file is very essential to the working of the further project.

How to use

Web Portal - Python

Pre-Allocation

- Student has to login into the portal using his roll number and the password (first time password is sent to his/her email id). In the submitted project, the password is same as the username. The user has the choice of changing it later.
- There is also an option of forgot password where the user can reset his password by answering a security question
- Once the student logs in he/she can change his/her personal details by clicking on 'Update profile', near 'Logout' button on top right. However, when the student logs in for the first time, (s)he is redirected to the 'Update profile' page so that (s)he can change whatever (s)he feels necessary.
- He/she can fill in his/her choices of preference anytime by going to 'Update profile'

How to use

Web Portal - Python
Pre-Allocation

- Student has to login into the portal using his roll number and the password (first time password is sent to his/her email id). In the submitted project, the password is same as the username. The user has the choice of changing it later.
- There is also an option of forgot password where the user can reset his password by answering a security question
- Once the student logs in he/she can change his/her personal details by clicking on 'Update profile', near 'Logout' button on top right. However, when the student logs in for the first time, (s)he is redirected to the 'Update profile' page so that (s)he can change whatever (s)he feels necessary.
- He/she can fill in his/her choices of preference anytime by going to 'Update profile'

How to use

Web Portal - Python
Pre-Allocation

- Student has to login into the portal using his roll number and the password (first time password is sent to his/her email id). In the submitted project, the password is same as the username. The user has the choice of changing it later.
- There is also an option of forgot password where the user can reset his password by answering a security question
- Once the student logs in he/she can change his/her personal details by clicking on 'Update profile', near 'Logout' button on top right. However, when the student logs in for the first time, (s)he is redirected to the 'Update profile' page so that (s)he can change whatever (s)he feels necessary.
- He/she can fill in his/her choices of preference anytime by going to 'Update profile'

How to use

Web Portal - Python
Pre-Allocation

- Student has to login into the portal using his roll number and the password (first time password is sent to his/her email id). In the submitted project, the password is same as the username. The user has the choice of changing it later.
- There is also an option of forgot password where the user can reset his password by answering a security question
- Once the student logs in he/she can change his/her personal details by clicking on 'Update profile', near 'Logout' button on top right. However, when the student logs in for the first time, (s)he is redirected to the 'Update profile' page so that (s)he can change whatever (s)he feels necessary.
- He/she can fill in his/her choices of preference anytime by going to 'Update profile'

How to use

Web Portal - Python
Pre-Allocation

- Student has to login into the portal using his roll number and the password (first time password is sent to his/her email id). In the submitted project, the password is same as the username. The user has the choice of changing it later.
- There is also an option of forgot password where the user can reset his password by answering a security question
- Once the student logs in he/she can change his/her personal details by clicking on 'Update profile', near 'Logout' button on top right. However, when the student logs in for the first time, (s)he is redirected to the 'Update profile' page so that (s)he can change whatever (s)he feels necessary.
- He/she can fill in his/her choices of preference anytime by going to 'Update profile'

How to use

Web Portal - Python

Pre-Allocation contd..

- A student can also see the list of possible programs the students may get based on previous year opening and closing ranks using the 'Seat Predictor' which is the user's homepage.
- The number of active users is displayed whenever a query is made in the homepage. The rank statistics are shown only when the number of users currently online are more than 50.

How to use

Web Portal - Python
Pre-Allocation contd..

- A student can also see the list of possible programs the students may get based on previous year opening and closing ranks using the 'Seat Predictor' which is the user's homepage.
- The number of active users is displayed whenever a query is made in the homepage. The rank statistics are shown only when the number of users currently online are more than 50.

How to use

Web Portal - Python
Pre-Allocation contd..

- A student can also see the list of possible programs the students may get based on previous year opening and closing ranks using the 'Seat Predictor' which is the user's homepage.
- The number of active users is displayed whenever a query is made in the homepage. The rank statistics are shown only when the number of users currently online are more than 50.

Django : Unchained

Structure

The entire Lab11 has been built around the Python's Django framework. The main directory contains several files which are the different modules of the project. The Admission folder contains the basic files which render the entire project onto a browser.

Admission folder

- settings.py : This file contains the configuration data for the entire website. It keeps track of the functions and middlewares to cal when a page is requested.
- urls.py : Whenever any link is accessed, the control passes to this file. It contains the list of mappings of urls, i.e. which url represents which function in views.py
- views.py : This is the main file which does all the background calculations and processes the data given by the user. The different functions in the file receive request from specific urls and return an appropriate page.

Django : Unchained

Structure

The entire Lab11 has been built around the Python's Django framework. The main directory contains several files which are the different modules of the project. The Admission folder contains the basic files which render the entire project onto a browser.

Admission folder

- settings.py : This file contains the configuration data for the entire website. It keeps track of the functions and middlewares to cal when a page is requested.
- urls.py : Whenever any link is accessed, the control passes to this file. It contains the list of mappings of urls, i.e. which url represents which function in views.py
- views.py : This is the main file which does all the background calculations and processes the data given by the user. The different functions in the file receive request from specific urls and return an appropriate page.

Django : Unchained

Structure

The entire Lab11 has been built around the Python's Django framework. The main directory contains several files which are the different modules of the project. The Admission folder contains the basic files which render the entire project onto a browser.

Admission folder

- settings.py : This file contains the configuration data for the entire website. It keeps track of the functions and middlewares to cal when a page is requested.
- urls.py : Whenever any link is accessed, the control passes to this file. It contains the list of mappings of urls, i.e. which url represents which function in views.py
- views.py : This is the main file which does all the background calculations and processes the data given by the user. The different functions in the file receive request from specific urls and return an appropriate page.

Django : Unchained

Structure
The entire Lab11 has been built around the Python's Django framework. The main directory contains several files which are the different modules of the project. The Admission folder contains the basic files which render the entire project onto a browser.
Admission folder

- settings.py : This file contains the configuration data for the entire website. It keeps track of the functions and middlewares to cal when a page is requested.
- urls.py : Whenever any link is accessed, the control passes to this file. It contains the list of mappings of urls, i.e. which url represents which function in views.py
- views.py : This is the main file which does all the background calculations and processes the data given by the user. The different functions in the file receive request from specific urls and return an appropriate page.

Django : Unchained

Structure

The entire Lab11 has been built around the Python's Django framework. The main directory contains several files which are the different modules of the project. The Admission folder contains the basic files which render the entire project onto a browser.

Admission folder

- settings.py : This file contains the configuration data for the entire website. It keeps track of the functions and middlewares to cal when a page is requested.
- urls.py : Whenever any link is accessed, the control passes to this file. It contains the list of mappings of urls, i.e. which url represents which function in views.py
- views.py : This is the main file which does all the background calculations and processes the data given by the user. The different functions in the file receive request from specific urls and return an appropriate page.

Django : Unchained

Structure contd..

- Data.py : It is a supporting file that helps render and keep track of the Dyanamic data being requested and accessed.
- SessionMiddleware.py : This is a supporting class that clears the session data when there has been no activity for some specified time. This basically helps in increasing the security and reduce traffic on the site and thus help it run smoothly.

Database folder

- models.py : This contains the Student class which is the template for the database. This stores all related information for the student.
- migrations : This contains the history of all changes to database.

Django : Unchained

Structure contd..

- Data.py : It is a supporting file that helps render and keep track of the Dyanamic data being requested and accessed.
- SessionMiddleware.py : This is a supporting class that clears the session data when there has been no activity for some specified time. This basically helps in increasing the security and reduce traffic on the site and thus help it run smoothly.

Database folder

- models.py : This contains the Student class which is the template for the database. This stores all related information for the student.
- migrations : This contains the history of all changes to database.

Django : Unchained

Structure contd..

- Data.py : It is a supporting file that helps render and keep track of the Dyanamic data being requested and accessed.
- SessionMiddleware.py : This is a supporting class that clears the session data when there has been no activity for some specified time. This basically helps in increasing the security and reduce traffic on the site and thus help it run smoothly.

Database folder

- models.py : This contains the Student class which is the template for the database. This stores all related information for the student.
- migrations : This contains the history of all changes to database.

Django : Unchained

Structure contd..

- Data.py : It is a supporting file that helps render and keep track of the Dyanamic data being requested and accessed.
- SessionMiddleware.py : This is a supporting class that clears the session data when there has been no activity for some specified time. This basically helps in increasing the security and reduce traffic on the site and thus help it run smoothly.

Database folder

- models.py : This contains the Student class which is the template for the database. This stores all related information for the student.
- migrations : This contains the history of all changes to database.

Django : Unchained

Structure contd..
HTML folder

- This folder contains the HTML pages which show the final webpages to the user. These contain the Python template tags and they are used to fill large data fields within some of the web pages.

manage.py

- This is the most powerful file and is responsible for managing **Everything**. This function is used to create the Database, the entire project and then this controls the database as well. Any changes to the database can be made easily by this program.

Any other files present in the directory are the supplementary files for maintaining active users.

Django : Unchained

Structure contd..

HTML folder

- This folder contains the HTML pages which show the final webpages to the user. These contain the Python template tags and they are used to fill large data fields within some of the web pages.

manage.py

- This is the most powerful file and is responsible for managing **Everything**. This function is used to create the Database, the entire project and then this controls the database as well. Any changes to the database can be made easily by this program.

Any other files present in the directory are the supplementary files for maintaining active users.

Django : Unchained

Structure contd..
Adding students to database

- We have populated the database with some default entries. You can add more users to the database though. Only these users will have access to the portal.
- One example user is : G111100093 and password is the same.
- The module reads from two files - 'ranklist.csv' and 'choices.csv'. To add new students, clear the two files and then add new users to the files. If they contain some users already then they will also get added to the database and that will not allow the already existing users to login.
- After above task has been achieved, execute the populateDatabase() method in the Data.py file in Admissions directory.

Django : Unchained

Structure contd..
Adding students to database

- We have populated the database with some default entries. You can add more users to the database though. Only these users will have access to the portal.
- One example user is : G111100093 and password is the same.
- The module reads from two files - 'ranklist.csv' and 'choices.csv'. To add new students, clear the two files and then add new users to the files. If they contain some users already then they will also get added to the database and that will not allow the already existing users to login.
- After above task has been achieved, execute the populateDatabase() method in the Data.py file in Admissions directory.

Django : Unchained

Structure contd..

Adding students to database

- We have populated the database with some default entries. You can add more users to the database though. Only these users will have access to the portal.
- One example user is : G111100093 and password is the same.
- The module reads from two files - 'ranklist.csv' and 'choices.csv'. To add new students, clear the two files and then add new users to the files. If they contain some users already then they will also get added to the database and that will not allow the already existing users to login.
- After above task has been achieved, execute the populateDatabase() method in the Data.py file in Admissions directory.

Django : Unchained

Structure contd..
Adding students to database

- We have populated the database with some default entries. You can add more users to the database though. Only these users will have access to the portal.
- One example user is : G111100093 and password is the same.
- The module reads from two files - 'ranklist.csv' and 'choices.csv'. To add new students, clear the two files and then add new users to the files. If they contain some users already then they will also get added to the database and that will not allow the already existing users to login.
- After above task has been achieved, execute the populateDatabase() method in the Data.py file in Admissions directory.

How to use

Post-Allocation

- Once the last date for filling the preferences is over, the admin can get the choices.csv file from the database
- Then the output can be put in test_cases folder and the Allocation program can be run
- The corresponding output can be put up then on the portal so that if the student logins then he can see his allocation

Come on, go and get your seat! We are awaiting you here :D

How to use

Post-Allocation

- Once the last date for filling the preferences is over, the admin can get the choices.csv file from the database
- Then the output can be put in test_cases folder and the Allocation program can be run
- The corresponding output can be put up then on the portal so that if the student logins then he can see his allocation

Come on, go and get your seat! We are awaiting you here :D

How to use

Post-Allocation

- Once the last date for filling the preferences is over, the admin can get the choices.csv file from the database
- Then the output can be put in test_cases folder and the Allocation program can be run
- The corresponding output can be put up then on the portal so that if the student logins then he can see his allocation

Come on, go and get your seat! We are awaiting you here :D

How to use

Post-Allocation

- Once the last date for filling the preferences is over, the admin can get the choices.csv file from the database
- Then the output can be put in test_cases folder and the Allocation program can be run
- The corresponding output can be put up then on the portal so that if the student logins then he can see his allocation

Come on, go and get your seat! We are awaiting you here :D

How to use

Post-Allocation

- Once the last date for filling the preferences is over, the admin can get the choices.csv file from the database
- Then the output can be put in test_cases folder and the Allocation program can be run
- The corresponding output can be put up then on the portal so that if the student logins then he can see his allocation

Come on, go and get your seat! We are awaiting you here :D

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Team

Pradyot Prakash

- Django part (major)
- Java part (minor)
- Beamer
- Video

Chandra Maloo

- Django part (major)
- Java part (minor)
- Testcases
- LaTeX Report

Shantanu Thakoor

- Java part (major)
- Django part (minor)
- Makefile
- Javadoc

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com

Bibliography

References

- www.djangobook.com
- www.tangowithdjango.com
- www.stackoverflow.com
- www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
- www.w3schools.com