Computer Science Project
2020-'21
Sudoku Web App

x

INDIAN SCHOOL SOHAR

CERTIFICATE

This is to certify that

x *of class* ***XII***

has carried out the project entitled

"Sudoku Web App"

as per the syllabus prescribed by

the Central Board of Secondary Education, New Delhi

for the subject

Computer Science(083)

during the academic year ***2020-21.***

Deepa Dinesh
Teacher, Dept. of Computer Science

Sanchita Verma
Principal

External Examiner

Internal Examiner

Acknowledgements

This project would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am greatly indebted to the teacher-in-charge, Ms. Deepa Dinesh for her guidance, constant supervision and for providing necessary information regarding the project.

I would like to express my gratitude towards my parents for their kind cooperation and encouragement which helped me in the completion of this project.

I would also like to express my special gratitude and thanks to my classmates in developing the project and to the people who have willingly helped me out with their abilities.

Contents

Introduction

Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Features of Python

Easy to code:

Python is a high-level programming language. Python is very easy to learn as compared to other languages like C, JavaScript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

Free and Open Source:

Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

Object-Oriented Language:

One of the key features of Python is Object-Oriented Programming. Python supports object-oriented language and concepts of classes, objects, encapsulation, etc.

High-Level Language:

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture or manage memory.

Feasibility Study

The feasibility study is the important step in any software development process. This is because it makes analysis of different aspects like - cost required for developing and executing the system, the time required for each phase of the system and so on. If these important factors are not analyzed then definitely it would have impact on the organization the development and the system would be a total failure.

The purpose of feasibility study is not to solve the problem, but to determine whether the problem is worth solving. By making analysis this way it would be possible to make a report of identified area of problem. By making a detailed analysis in this area a detailed document or report is prepared in this phase which has details like project plan or schedule of the project, the cost estimated for developing and executing the system, target dates for each phase of delivery of system developed and so on. This phase is the base of software development process since further steps taken in software development life cycle would be based on the analysis made on this phase and so careful analysis has to be made in this phase.

TELOS

The feasibility study concentrates on the following area (TELOS):

- Technology and System Feasibility
- Economic Feasibility
- Legal Feasibility
- Operational Feasibility
- Schedule Feasibility

Technology and System Feasibility

The assessment is based on an outline design of system requirements, to determine whether the company has the technical expertise to handle completion of the project.

Economic Feasibility

The economic feasibility study evaluates the cost of the software development against the ultimate income or benefits expected from the developed system.

It includes identifying cost and benefit factors like - Development costs and Operating costs. There must be scopes for profit after the successful completion of the project.

Legal Feasibility

It determines whether the proposed system conflicts with legal requirements, e.g. a data processing system must comply with the local Data Protection Acts.

Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

Schedule Feasibility

A project will fail if it takes too long to be completed before it is useful. Typically this means estimating how long the system will take to develop, and if it can be completed in a given time period using some methods like payback period. Schedule feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the project deadlines reasonable?

Advantages of Feasibility Study

- As the initial step of software development life cycle, feasibility study has all the analysis part in it, which helps in analyzing the system requirements completely.
- Helps in identifying the risk factors involved in developing and deploying the system.
- It helps in making cost/benefit analysis which helps the organization and system to run efficiently.
- It is a report which could be used by the senior or top persons in the organization. This is because, based on the report the organization decides about cost estimation, funding and other important decisions which is very essential for an organization to run profitably and for the system to run stable.

Software Development Life Cycle

The Systems Development Life Cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project from an initial feasibility study through maintenance of the completed application.

The following are the activities of the SDLC:

- Software requirement analysis
- Systems analysis and design
- Design/Code generation
- Testing
- Development and Maintenance

A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation. A number of system development life cycle (SDLC) models have been created such as waterfall, fountain, spiral etc.

Requirement Analysis/Investigation

The 1st stage of SDLC is the investigation phase. During this stage, business opportunities and problems are identified, and information technology solutions are discussed. Multiple alternative projects may be suggested and their feasibility analyzed. The results of the feasibility study can then be compiled into a report, along with preliminary specifications. When the investigation stage ends, a decision whether or not to move forward with the project should be made.

System Analysis

The goal of system analysis is to determine where the problem is, in an attempt to fix the system. It analyzes the requirement for the proposed system. To understand the nature of the program to build, the system engineer must understand the information domain for the software, as well as required functions, performance and the interfacing. This step involves breaking down the system in different pieces to analyze the situation, analyzing project goals, breaking down what needs to be created. From the available information the system engineer develops a list of system level requirement for the project.

Design

Systems design describes screen layouts, business rules, process diagrams, a complete entity- relationship diagram with a full data dictionary and other documentation. It defines specifically how the software is to be written including an object model, the client/server technology, a detailed database design etc. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input design. Analysis and design are very important in the whole development cycle. Any glitch in the design could be very expensive to solve in the later stage of the software development. The design must be translated into a machine readable form.

Testing

In this stage, all the pieces of software are brought together into a special testing environment and then are checked for errors, bugs and interoperability. Unit, system and user acceptance testing is often performed.

Deployment and Maintenance

Deployment is the final stage of initial development. It involves installation, initial training and may involve hardware and network upgrades. Software will definitely undergo change once it is delivered to the customer. There may be many reasons for the change. Change could be due to some unexpected input values into the system. The software should be developed to accommodate changes that could take place during the post implementation period. Maintaining the system is also an important aspect of SDLC.

Hardware and Software

Hardware

Laptop Specifications

Dell G7
Intel Core i7, 16 GB RAM

Software

1. **Python**
2. **MySQL**
3. **Django**
4. **Visual Studio Code**
5. **PyCharm Community Edition**
6. **HTML**
7. **JavaScript**
8. **TailwindCSS**
9. **Git SCM**
10. **GitHub**
11. **Google Cloud Platform**

12. **Gunicorn**

13. **LaTeX**

14. **Adobe Photoshop**

About the Project

The aim of the project, Sudoku Webapp, is to provide an aesthetically rich interface for sudoku enthusiasts to enjoy sudoku puzzles.

A website with 3 webpages - home page, sudoku page and leaderboard page - is the frontend of the project with which the user interacts. The entire backend is handled using python and the django framework. The sudoku game boards are generated using a combination of custom classes and functions as well as functions from the random module in python. These game boards are passed onto the website using queues. A MySQL database is the primary data store which is used to store the leaderboard data, and the sudoku game boards that are served onto the site. The webpages are made using HTML, JavaScript and a CSS framework called TailwindCSS.

A personal aim with this project was to see how much we, as a team, could push ourselves to make a good, functional service. It challenged our creativity, programming skills and our ability to work and co-operate as a team. This project was a great learning experience.

Source Code

File Structure and Dependencies

File Structure

A rough structure of the project is as follows:

```
SudokuWebapp
├── latex-report
├── sudoku-django-project
│   ├── mainapp
│   │   ├── migrations
│   │   ├── static
│   │   └── templates
│   ├── static
│   ├── sudoku
│   ├── utils
│   ├── manage.py
│   └── requirements.txt
├── .gitignore
└── README.md
```

The `SudokuWebapp` directory is the root of the repository, and contains the entire project.
Inside this, the `sudoku-django-project` is the root of the *Django project*, and is where all the code is held, while the `latex-report` directory contains all the images and LaTeXcode used to make this report.
The `.gitignore` is a special file used by the Git SCM. All files that are meant to be ignored by the version-control system (and not committed to history) are included in this.
The `sudoku-django-project/requirements.txt` is where all project dependencies are listed.
The entire source code of the project is hosted on our Github repository, and can be viewed at `https://www.github.com/cs-gang/sudokuwebapp`.

Dependencies

```
asgiref==3.2.10
cffi==1.14.3
cryptography==3.2.1
Django==3.1.2
```

```
django-mysql==3.9.0
flake8==3.8.4
gunicorn==20.0.4
mccabe==0.6.1
mysqlclient==2.0.1
protobuf==3.13.0
pycodestyle==2.6.0
pycparser==2.20
pyflakes==2.2.0
PyMySQL==0.10.1
pytz==2020.1
six==1.15.0
sqlparse==0.3.1
```

All dependencies were installed in an isolated *Virtual Environment*.
This `requirements.txt` file was auto-generated by using the `pip freeze > requirements.txt` command.

Backend

Backend, also referred to as the *"server-side"*, is responsible for facilitating communication between the presentation layer and the data layer.
The backend of this project is handled by the *Django* web framework.

Django Project Architecture

Django uses two important terms in development: *project* and *app*.
A **project** refers to the entire web application.
An **app** refers to a sub-module, catering to one specific part of the project.
In our case, the *project* is `sudoku-django-project`. This project contains only one *app*, called `mainapp`.
Django, being a web framework, auto-generates a lot of files for us when we create a new project. Almost all the code in the `sudoku-django-project/sudoku` directory was made like this. They contain instructions and settings for the entire project itself.
Some of these files are listed below:

- *manage.py*
  Used to manage the project and run adminsitrative commands like running the built-in server.

- *sudoku/settings.py*
  Contains most of the configuration for the Django project, like what databases to use, where to look for static files etc.

- *sudoku/urls.py*
  Contains information about locations where URLs of the project has been declared.

The application folder `sudoku-django-project/mainapp` also contains many pre-made files.

Django projects separates data (*models*), the logic (*view*) and the final presentation (*template*) that users will see. This kind of architecture is hence called **MTV (Model Template View)** architecture.

1. A **model** is responsible for data management, and deals with access and relationships between data. Django uses an **ORM (Object Relational Mapping)** to make this process more intuitive.

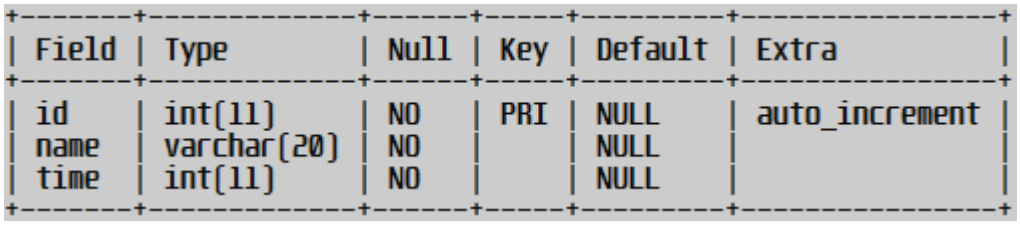
2. A **template** is responsible for presentation of data to the user. These are the HTML pages.

3. A **view** acts like a bridge between models and templates. A view accesses model data and redirects it to a template for presentation.

Models

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
check_board	longtext	NO		NULL	
game_board	longtext	NO		NULL	

Figure 1: Game boards table



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(20)	NO		NULL	
time	int(11)	NO		NULL	

Figure 2: Leaderboards Table

```
    from django.db import models
    from typing import Union

    class GameBoards(models.Model):
        # An autoincrementing ID column which will be used as
    primary key is automatically added.
        game_board = models.TextField()
        check_board = models.TextField()
        def __str__(self) -> int:
            return self.id

    class Leaderboard(models.Model):
        name = models.CharField(max_length=20, default="
   Player", blank=False)
        time = models.IntegerField()

        def __str__(self) -> str:
            return ", ".join([self.name, str(self.time)])

```

Listing 1: mainapp/models.py

Views and Forms

```
from django.shortcuts import render, redirect
from django import http
from django.views.decorators.http import require_POST,
   require_GET
from django.db.models import Max
import json

from utils.classes import BoardsQueue
from utils.exceptions import QueueUnderflowError

from .models import GameBoards, Leaderboard
from .forms import AddToLeaderboardForm

queue = BoardsQueue()

lower, upper = 0, 11    # ID range to load to queue
```

```
# filling queue with tables saved on the database
for result in GameBoards.objects.filter(id__in=list(range(lower, upper))):
    queue.enqueue([result.id, json.loads(result.game_board), json.loads(result.check_board)])
else:
    lower, upper = 11, 21

@require_GET
def index(request):
    return render(request, "mainapp/index.html", {})

@require_GET
def game(request):
    global upper, lower
    try:
        board_id ,game_board, check_board = queue.dequeue()
    except QueueUnderflowError:
        if upper == 101:        #there are currently only 100 items on the database, so it loops back to the starting.
            lower, upper = 0, 11
        else:
            lower += 10
            upper += 10

        for result in GameBoards.objects.filter(id__in=list(range(lower, upper))):  #loading new tables from database
            queue.enqueue([result.id, json.loads(result.game_board), json.loads(result.check_board)])
        else:
            lower, upper = upper, upper + 10

        board_id, game_board, check_board = queue.dequeue()
    context = {'board_id': board_id, 'game_board': game_board, 'check_board': check_board, 'form': AddToLeaderboardForm()}

    return render(request, "mainapp/game.html", context)

def result(request):
    if request.method == 'POST':
        form = AddToLeaderboardForm(request.POST)
        if form.is_valid():
            time = form.cleaned_data['time']
            username = form.cleaned_data['username']

        current_worst = Leaderboard.objects.all().aggregate(Max('time'))
        if time < current_worst['time__max']:
```

```
57              new = Leaderboard(name=username, time=time)
58              new.save()
59
60          return redirect('index')
61
62  def leaderboard(request, home=""):
63      if home == "lb":
64          data = Leaderboard.objects.all().order_by('time')
65          formatted_data = [[entry.name, entry.time] for entry
    in data][:10]
66          context = {"data": formatted_data}.
67          return render(request, "mainapp/leaderboard.html",
    context)
68      else:
69          return redirect("index")
70
```

Listing 2: mainapp/views.py

Variables declared in the *context* dictionary are passed onto the templates.

A *form* was used to send the player's data over HTTP POST request, back to the server, if they have to be added to the leaderboard.

```
1  from django import forms
2
3  class AddToLeaderboardForm(forms.Form):
4      username = forms.CharField()
5      time = forms.IntegerField()
6
```

Listing 3: mainapp/forms.py

Sudoku game boards were prepared before hand, and stored in the database. Small chunks were retrieved on demand and placed in a queue, to serve to live users. Functions and classes related to preparing the boards were made in the `utils` directory in the project.

```
1  from random import shuffle, randint
2  import typing
3  import utils.exceptions
4
5  BoardType = typing.List[typing.List[typing.Union[None, int]]]
         # type hint alias for board - a list of lists
6        # with int or None as values
7  class Sudoku:
8      """ A class that acts like a sudoku puzzle. """
9
10     def __init__(self):  # noqa: ANN204
```

```
        self.counter = 1
        self.top_boxes = [_Box() for _ in range(3)]
        self.mid_boxes = [_Box() for _ in range(3)]
        self.bottom_boxes = [_Box() for _ in range(3)]

        self.original = [
            [1, 2, 3, 4, 5, 6, 7, 8, 9],
            [4, 5, 6, 7, 8, 9, 1, 2, 3],
            [7, 8, 9, 1, 2, 3, 4, 5, 6],
            [2, 3, 1, 5, 6, 4, 8, 9, 7],
            [5, 6, 4, 8, 9, 7, 2, 3, 1],
            [8, 9, 7, 2, 3, 1, 5, 6, 4],
            [3, 1, 2, 6, 4, 5, 9, 7, 8],
            [6, 4, 5, 9, 7, 8, 3, 1, 2],
            [9, 7, 8, 3, 1, 2, 6, 4, 5]
        ]

        self.generator()
        self.full_board = self.get_all_row_values()
        self.puzzle_maker()

    # Getter methods
    def get_column_values(self, index: int) -> list:  # returns column values in the form of
        box_index = index // 3  # a list. Indexing from 0-8 from
        element_index = index % 3  # left to right.
        column = []
        for i in range(9):
            if i % 3 == 0 and i > 1:
                box_index += 3
                element_index -= 9
            element = self[box_index][element_index]
            column.append(element.get_value())  # appends value of the element.
            element_index += 3
        return column

    def get_row_values(self, index: int) -> list:  # returns row values in the form of a list.
        box_index = (index // 3) * 3  # Indexing from 0-8 from top to bottom
        element_index = index % 3 * 3
        row = []
        for i in range(9):
            if i % 3 == 0 and i > 1:
                box_index += 1
                element_index -= 3
            element = self[box_index][element_index]
```

```
55              row.append(element.get_value())
56              element_index += 1
57          return row
58
59      def get_all_row_values(self) -> list:  # Return a list of
        all the rows.
60          rows = []
61          for i in range(9):
62              row = self.get_row_values(i)
63              rows.append(row)
64          return rows
65
66      # Puzzle-Generation methods
67      def possible_cell_values(self, row: int, col: int) ->
    list:
68
69          element_possibility = [1, 2, 3, 4, 5, 6, 7, 8, 9]
70
71          for col_value in self.get_column_values(col):
72              if col_value in element_possibility:  #
    Reoccurring in the same column
73                  element_possibility.remove(col_value)
74
75          for row_value in self.get_row_values(row):
76              if row_value in element_possibility:  #
    Reoccurring in the same row
77                  element_possibility.remove(row_value)
78
79          for m in element_possibility:
80              if m in [self[(row // 3) * 3 + col // 3][n].
    get_value() for n in range(9)]:
81                  element_possibility.remove(m)
82
83          return element_possibility
84
85      @staticmethod
86      def check_complete(grid: list) -> bool:
87          for i in grid:
88              for x in i:
89                  if x == 0:
90                      return False
91          return True
92
93      def generator(self) -> None:
94
95          shuffled = []
96
97          for i in range(3):
98              rows = [self.original[i * 3], self.original[(i *
```

```
3) + 1], self.original[(i * 3) + 2]]
99
100             shuffle(rows)
101             shuffled.extend(rows)
102
103         self.set_value_of_grid(shuffled, "row")
104         shuffled = []
105
106         for i in range(3):
107             cols = [self.get_row_values(i * 3), self.
    get_row_values((i * 3) + 1), self.get_row_values((i * 3) +
     2)]
108
109             shuffle(cols)
110             shuffled.extend(cols)
111
112         self.set_value_of_grid(shuffled, "col")
113
114     def set_value_of_grid(self, list_val: list, index_type:
    str) -> None:
115
116         if index_type == "row":
117             for box in range(9):
118                 for element in range(9):
119                     self[box][element].set_value(list_val[(
    box // 3) * 3 + element // 3][(box % 3) * 3 + element %
    3])
120         elif index_type == "col":
121             for box in range(9):
122                 for element in range(9):
123                     self[box][element].set_value(list_val[(
    box % 3) * 3 + element % 3][(box // 3) * 3 + element //
    3])
124
125     def unique_sol_check(self, grid: list) -> bool:
126
127         # Recursive method to check whether there is a unique
     solution for a number removed from grid
128         for i in range(81):
129
130             row = i // 9
131             col = i % 9
132
133             if grid[row][col] == 0:
134                 for value in range(10):
135                     if value in self.possible_cell_values(row
    , col):
136                         grid[row][col] = value
137                         self.set_value_of_grid(grid, "row")
```

```
                        if self.check_complete(grid):
                            self.counter += 1
                            break
                        else:
                            if self.unique_sol_check(grid):
                                return True
                break
        grid[row][col] = 0
        self.set_value_of_grid(grid, "row")

    def puzzle_maker(self) -> None:

        # adds spaces to the finished board
        attempts = 5
        while attempts > 0:
            row = randint(0, 8)
            col = randint(0, 8)
            while self[row][col].get_value() == 0:
                row = randint(0, 8)
                col = randint(0, 8)
            backup = self[row][col].get_value()
            self[row][col].set_value(0)

            copy = self.get_all_row_values()

            self.counter = 0
            self.unique_sol_check(copy)

            if self.counter != 1:
                self[row][col].set_value(backup)
                attempts -= 1

    @staticmethod
    def check(user_input: list, full_board: list) -> typing.
Union[bool, list]:

        if user_input == full_board:
            return True
        else:
            return [[True if full_board[row][col] ==
user_input[row][col] else False for col in range(9)] for
row in range(9)]

    # operator overloading methods.
    def __iter__(self):  # noqa: ANN204
        for i in self.top_boxes:  # x here is a box
            yield i  # iterates through the boxes in the same
    way as
        for i in self.mid_boxes:  # a matrix; i.e. left to
```

```
    right.
183             yield i
184         for i in self.bottom_boxes:
185             yield i
186 
187     def __getitem__(self, index: int):  # noqa: ANN204
188         count = 0  # returns box object at index 3
189         for i in self:
190             if count == index:
191                 return i
192             count += 1
193 
194     def __eq__(self, b: "Sudoku") -> bool:  # functionality
    -> sudoku1 == sudoku2
195         for box in range(9):  # compares all values of both
    sudokus.
196             for element in range(9):
197                 if self[box][element].get_value() != b[box][
    element].get_value(): return False
198         return True
199 
200 
201 class _Box:
202     """ A class that acts like one of the nine 3x3 boxes in
    sudoku. """
203 
204     def __init__(self):  # noqa: ANN204
205         self.top_row = [_Element() for _ in range(3)]
206         self.mid_row = [_Element() for _ in range(3)]
207         self.bottom_row = [_Element() for _ in range(3)]
208 
209     # Operator overloading methods
210     def __iter__(self):  # noqa: ANN204
211         for element in self.top_row:  # Iterates in the same
    way as a matrix
212             yield element  # i.e. from left to right.
213         for element in self.mid_row:
214             yield element
215         for element in self.bottom_row:
216             yield element
217 
218     def __getitem__(self, index: int):  # noqa: ANN204
219         count = 0  # returns value of specified index.
220         for element in self:  # indexing from 0-9; indexing
    is in the same way
221             if count == index:  # as a matrix.
222                 return element
223             count += 1
224 
```

```
225     def __setitem__(self, index: int, val: typing.Optional[
    int]=None):  # noqa: ANN204
226         count = 0  # assignment at a specific index.
227         for element_place in range(9):
228             if count == element_place:
229                 self[element_place].set_value(val)
230             count += 1
231 
232     def __contains__(self, val: typing.Optional[int]=None) ->
    bool:  # functionality -> val in box
233         for element in self:  # returns True or False.
234             if element.get_value() == val:
235                 return True
236         return False
237 
238 
239 class _Element:
240     """ A class that acts as the element(number) in one of
    the boxes of sudoku. """
241 
242     def __init__(self, value: typing.Optional[int]=None):   #
    noqa: ANN204
243         self._value = value  # The value the element has.
244 
245     # getter methods.
246     def get_value(self) -> typing.Any:  # To access the value
    this element has.
247         return self._value
248 
249     # setter methods.
250     def set_value(self, value: typing.Optional[int]=None) ->
    None:  # To access the value this element will have.
251         self._value = value
252 
253 
254 class BoardsQueue:
255     """FIFO queue containing upto {max_length} Sudoku boards
    at a time.
256     Front of the queue is the index 0, and the back is -1.
257 
258     Raises: QueueOverflowError -> on trying to insert into
    the queue beyond it's specified max_size
259             QueueUnderflowError -> on trying to get an item
    from the empty queue."""
260     def __init__(self, max_size: int=10):   # noqa: ANN204
261         self._queue = []
262         self.max_size = max_size
263 
264     def enqueue(self, board: list) -> None:
```

```
265         if len(self._queue) > self.max_size:
266             raise utils.exceptions.QueueOverflowError
267 
268         self._queue.append(board)
269 
270     def dequeue(self) -> typing.Union[list, None]:
271         if len(self._queue) == 0:
272             raise utils.exceptions.QueueUnderflowError
273 
274         return self._queue.pop(0)
275 
```

Listing 4: utils/classes.py

Some exceptions that are raised by the queue are defined in `utils/exceptions.py` file. These exceptions inherit from the base `Exception` class.

URLconf

URL configuration, or *URLconf*, is the process of defining URLs for your app. This is done by writing some Python code, which acts as a mapping between URL path expressions, and their corresponding view functions. The URLconf for this project has two layers:

1. *Project-level URLs*

```
1 from django.contrib import admin
2 from django.urls import path, include
3 
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('', include('mainapp.urls')),
7 ]
8 
```

Listing 5: sudoku/urls.py

   This simply redirects every URL except `/admin` to the URL configuration for `mainapp`.

2. *App-level URLs*

```
1 from django.urls import path
2 from . import views
3 
4 urlpatterns = [
5     path("", views.index, name="index"),
6     path("game", views.game, name="game"),
```

```
    path("result", views.result, name="result"),
    path("leaderboard/<home>", views.leaderboard, name="
    leaderboard")
]

```

Listing 6: mainapp/urls.py

This maps a URL to a specific view function defined in `views.py`; when any of these URLs are reached, their corresponding view function is automatically called.

Frontend

Frontend, also referred to as the *"client-side"*, is the part of the website that you can see and interact with directly.

Web pages are designed using three languages:

- Hypertext Markup Language (HTML)
  It is used for laying out the structure of the webpage and for making forms that share data with the backend.
- Cascading Style Sheets (CSS)
  It is used to improve the page visually.
- JavaScript
  It is a programming language that can be used in both frontend and backend. It adds behaviour to web pages and makes them interactive.

As CSS alone is time consuming to write, frameworks are used to speed up the development. In our project TailwindCSS framework has been used.

Output

Figure 3: Index Page

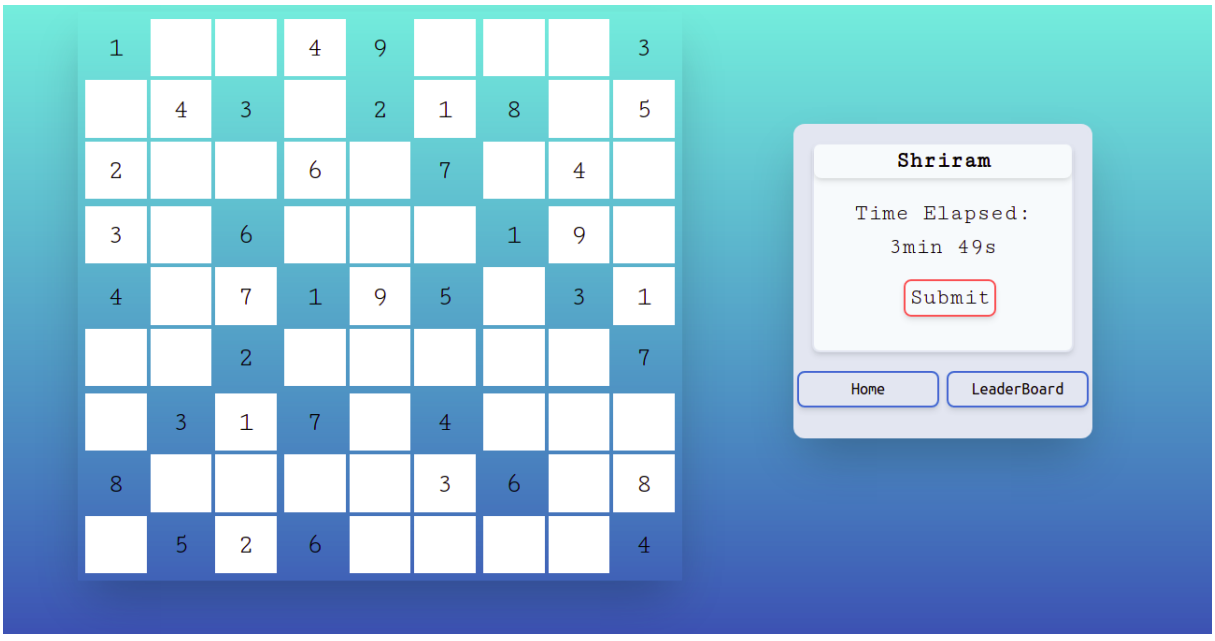


Figure 4: Game Page

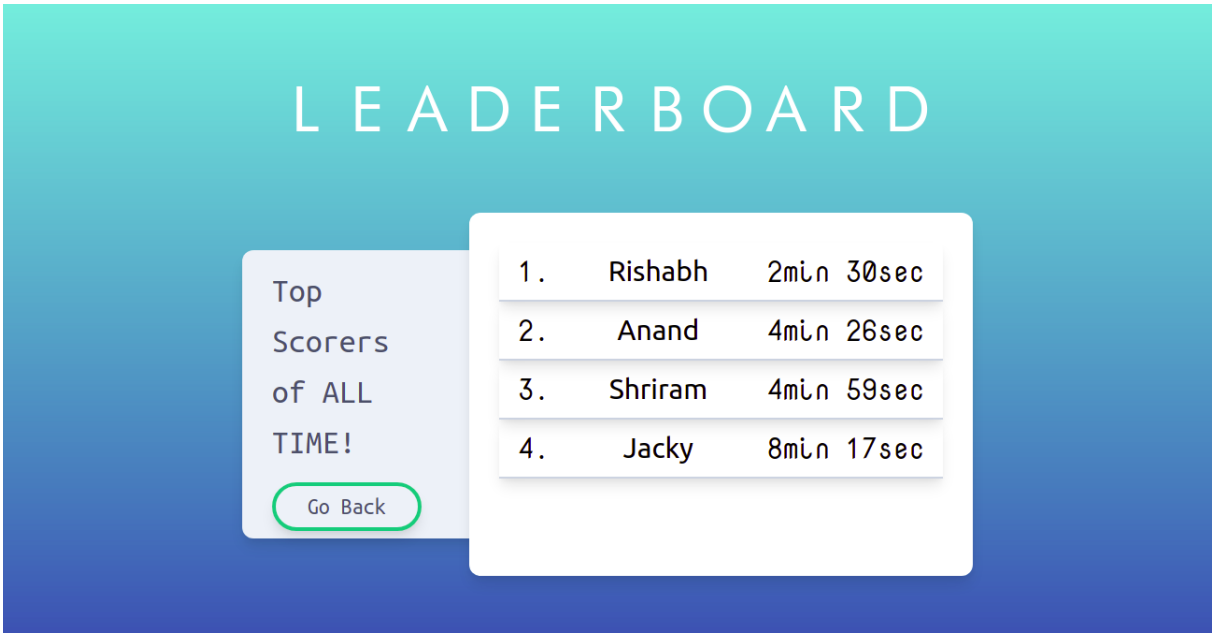


Figure 5: Leaderboard Page

Bibliography

[1] Python 3. *Interpreted, high-level, general-purpose programming language.* `https://docs.python.org/3/`

[2] Django. *The Web Framework for perfectionists with a deadline* `https://docs.djangoproject.com/en/3.1/`

[3] TailwindCSS. *A utility-first CSS framework packed with classes.* `https://tailwindcss.com/docs`

[4] MySQL Database. *Fast relational database.* `https://dev.mysql.com/doc/`

[5] Wikipedia. *The free online encyclopedia.* `https://www.wikipedia.com`

[6] Stackoverflow. *Where developers learn, share and build.* `https://stackoverflow.com/`

[7] LaTeX. *High quality type-setting system.* `https://www.latex-project.org/help/documentation/`

[8] Git SCM *Distributed Version-Control System.* `https://git-scm.com/`