МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Искусственного Интеллекта
Базовая кафедра БК252

ПРАКТИЧЕСКАЯ РАБОТА

По дисциплине «Системы управления базами данных»

Студент группы ККСО-03-19	Николенко В.О.
Студент группы ККСО-03-19	Воеводин К.А.
Студент группы ККСО-03-19	Носов Г.А.
Студент группы ККСО-03-19	Базарова Е.О.
Преподаватель	Колесников С.В.

Москва – 2023

СОДЕРЖАНИЕ

1. Введение 3
2. Описание проекта 4
   2.1. Цель 4
   2.2. Процедура логического проектирования 4
   2.3. Математическая модель 5
   2.4. Аналитика 6
3. Практическая реализация проекта 8
   3.1. Архитертура проекта 8
   3.2. Программный код 10

1. ВВЕДЕНИЕ

Целью этого проекта для курса "Системы управления базами данных" является разработка программы, включающей в себя процессы сбора и обрадотки данных. Подробности разработки приведены в этом отчете. Отчёт включает в себя распределение задач и процессов внутри группы студентов.

Мы начнём с поставленной задачи, разбиения её на этапы, поднимемся на уровень реализации логики локальной БД и реализации программного кода, после чего произведем проверку.

2. ОПИСАНИЕ ПРОЕКТА

2.1. Цель

Целью является разработка приложения, которое получающет данные с помощью API, обрабатывает их и сохраняет в локальную БД. В роли БД будут 2 таблицы, создаваемые и наполняемые данными в результате работы программы.

2.2. Процедура логического проектирования

В процессе разработки испольъзовались такие инструменты как github и trello. Каждый из студентов отвечал за конкретные задачи:

Николенко В. О. - разработка API, её подключение и написание основной части функций вызова.

Воеводин К. А. - рассчет математической модели и общих параметров БД.

Носов Г. А. - доработка мат модели и реализация обработки и хранения данных.

Базарова Е. О. - доработка, тест API и его настройка, составление и написание отчета.

Программа реализована на языке TypeScript на npm-подобном пакетном менеджере Yarn. Данные загружаются с помощью API с сайта hltv.org.

Hltv.org - это сайт с подробной информацией о киберспортивных событиях, с подробной статистикой о игроках, командах, турнирах и т.д. Вокруг киберспорта в последние годы сформировалось большое количество внимания, фанатской базы и, соответсвенно, различных турниров с крупными призовыми наградами и ценными трофеями, попасть на которые могут либо профессиональные команды, имеющие большие успехи в последних событиях, либо малоизвестные команды, которые наравне с сотнями других соревнуются для получения приглашения на основные турниры, в которых соревнуются 15-20 сильнейших команд. Таким образом, формируются популярные команды, сделавшие себе репутацию и на протяжении нескольких лет могут находиться на слуху, но при этом при встрече с менее известным соперником потерпеть поражение.

Для того, чтобы здраво оценивать соревновательную форму той или иной команды, можно путём сбора и обработки информации, в реальном времени иметь представление о текущей форме конкретной команды.

2.3. Математическая модель

Для рассчета формы команд используются несколько критериев:

1. Рейтинг команды со значениями от 0 до 1000 (в таблице обозначен как ranking), рассчитываемый на основе числа выиграных турниров, занятого места и важности самих турниров.

2. Текущий уровень формы каждого отдельно взятого игрока (обозначен как rating 2.0) - основывается на результатах игрока за каждый сыгранный матч по параметрам (kd, avg damage, число раундов взятых командой благодаря этому игроку), в таблице обозначен как avg2.0 - средее значение данного параметра по команде.

3. Число матчей, сыгранных за последний месяц / полтора / два с половиной (в таблице обозначены как ma1, ma1,5, ma2,5) - выясняется сколько матчей было сыграно за промежуток времени и сколько из них выиграно, делится число побед на число игр в общем.

4. Статистика сыгранных матчей команды, сыгранных ей на конкретной карте. Таким образом, составляется список карт каждой команды, на которых она имеет лучшие результаты. Информация по статистике наждой команды на каждой карте находится во второй таблице.

```
(async function main() {
  //========================================
  await setBaseInfo();
  // await setMapsInfo();
  // await db.read("stat.xlsx");
  // await db.read("mapsStats.xlsx");
  //========================================
  // console.log(
  //   await getTeam(((await getTopTeams(false))[0] as TeamRanking).team.name)
  // );
  // // Преобразуем объект в строку JSON
  // const jsonData = JSON.stringify(await HLTV.getEvents());
  // // Сохраняем данные в новый файл
  // await writeFile("balance.json", jsonData);
})();
```

Таким образом выглядят вызываемые функции в файле main, за один запуск программы мы можем вызвать только одну из них, т. к. сайт с исходной информацией содержит ограничение по числу запросов за одну сессию. Воизбежание проблем с качеством и скоростью работы программы было принято решение разделить функции и отдельно запускать их с каждым запуском программы.

2.4. Аналитика

Для анализа статистики игр команд используются данные о прошедших матчах на различных картах в течение последних 2,5 месяцев. Мы использовали рейтинг, который учитывает результаты игр, их важность и результаты игр на определенных картах.

Затем мы построили линейную диаграмму, на которой отображены пять самых сильных и пять самых слабых команд. Для этого мы отсортировали команды по их рейтингу и выбрали пять сильнейших команд и пять команд со слабыми результатами.

Для определения влияния карты на результаты игры мы использовали коэффициент Пирсона, который позволяет оценить силу связи между двумя переменными. В данном случае мы сравнивали рейтинг команды и результаты ее игр на конкретной карте. Коэффициент Пирсона вычисляется по формуле:

$$r_{xy} = \frac{\sum (x_i - M_x)(y_i - M_y)}{\sqrt{\sum (x_i - M_x)^2 \cdot (y_i - M_y)^2}}$$

где x_i - рейтинг команды на карте i, y_i - результаты игр команды на карте i, n - количество карт. Коэффициент Пирсона принимает значения от -1 до 1, где -1 означает полную отрицательную связь, 1 - положительную связь, а 0 - отсутствие связи.

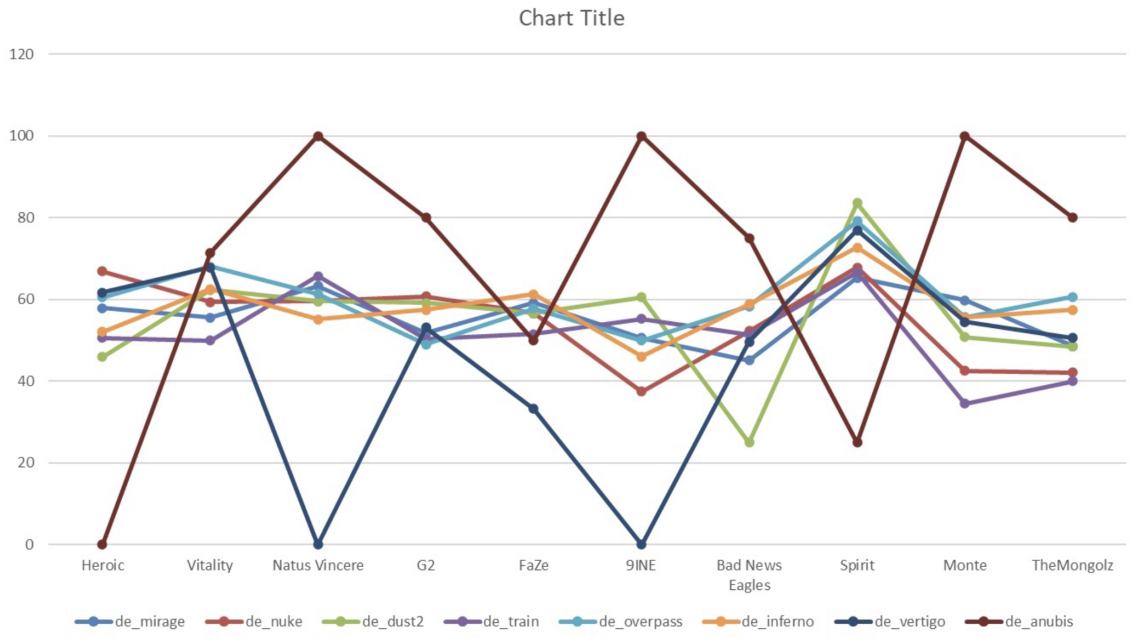
Карты	Коэффициент Пирсона
de_mirage	0,209439708361612
de_nuke	0,201912136190255
de_dust2	0,020100280309735
de_train	0,255836968824764
de_overpass	0,167396361189406
de_inferno	0,11687376885097
de_vertigo	-0,09629569239211
de_anubis	0,045355025934818

Анализ статистики игр показал, что обучаться лучше всего на карте de train. Коэффициент Пирсона между рейтингом команды и результатами игр на этой карте был выше, чем на других картах, что свидетельствует о более сильной связи между этими факторами. Данная карта обеспечивает наилучшие условия для развития навыков и стратегий команды, что в свою очередь влияет на результаты игр.

На картах de anubis и de dust2 будет сложнее спрогнозировать результаты игр, что подтверждается коэффициентом Пирсона, который был ниже, чем на других картах. Это свидетельствует о том, что на этих картах результаты игр могут быть менее предсказуемыми и менее связанными с рейтингом команды. Также было обнаружено, что на карте de vertigo слабые команды

могут показывать хорошие результаты, что может быть связано с особенностями карты и стратегиями, которые используются на ней.

Линейная диаграмма пять самых сильных и пять самых слабых команд показала, что разница в рейтинге между самыми сильными и самыми слабыми командами в целом не является значительной. Например, на некоторых картах, рейтинг самой сильной команды может быть в два или более раза выше, чем рейтинг самой слабой команды. Однако, как уже было отмечено, на определенных картах даже слабые команды могут показывать хорошие результаты.



3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОЕКТА

3.1. Архитертура проекта

Проект представляет из себя несколько файлов с программным кодом: DB.ts - представлен классом DB который имеет 2 поля: конструктор и функции, 2 публичные переменные класса - имя файла и область данных, которая будет записана в виде new Excel.Workbook(), для этого использует библиотека exceljs. По итогу получаем 2 следующих таблицы:

	A	B	C	D	E	F	G
1	teamId	name	ranking	avg2_0	ma1	ma1_5	ma2_5
2	7175	Heroic	904	5,54	NaN	NaN	NaN
3	9565	Vitality	895	5,56	NaN	NaN	NaN
4	4608	Natus Vin	772	5,43	NaN	NaN	NaN
5	5995	G2	697	5,45	NaN	NaN	NaN
6	6667	FaZe	680	6,45	NaN	NaN	NaN
7	5752	Cloud9	606	5,53	1	1	1
8	5973	Liquid	418	5,29	NaN	NaN	NaN
9	8297	FURIA	345	5,3	NaN	NaN	NaN
10	4869	ENCE	290	6,4	1	1	1
11	5378	Virtus.pro	268	4,99	NaN	NaN	NaN
12	4494	MOUZ	239	5,3	NaN	NaN	NaN
13	10503	OG	239	6,36	1	1	1
14	7532	BIG	237	6,34	1	1	1
15	4411	Ninjas in P	216	5,12	NaN	NaN	NaN
16	4991	fnatic	214	5,13	NaN	NaN	NaN
17	8135	FORZE	195	5,27	1	1	1
18	4773	paiN	178	6,65	1	1	1
19	5005	Complexit	161	5,13	NaN	NaN	NaN
20	10278	9INE	149	5,35	1	1	1
21	11518	Bad News	145	5,12	NaN	NaN	NaN
22	7020	Spirit	135	5,03	NaN	NaN	NaN
23	11811	Monte	113	5,49	1	1	1
24	6248	TheMongc	111	4,89	NaN	NaN	NaN
25	6665	Astralis	107	5,64	1	1	1
26	7718	Movistar I	104	4,14	1	1	1
27	8008	Grayhoun	101	5,43	NaN	1	1
28	9928	GamerLeg	78	5,34	1	1	1
29	11164	Into the Bi	77	5,29	1	1	1
30	9806	Apeks	75	5,38	1	1	1
31	11309	00NATION	74	6,15	1	1	1

Рис.1. Таблица со статистикой команд.

	A	B	C	D	E	F	G	H	I	J
1	teamId	name	de_mirage	de_nuke	de_dust2	de_train	de_overpass	de_inferno	de_vertigo	de_anubis
2	7175	Heroic	57,9	66,9	45,9	50,6	60,6	52,1	61,7	0
3	9565	Vitality	55,6	59,4	62,3	50	68,1	62,6	67,9	71,4
4	4608	Natus Vin	63,3	59,6	59,6	65,7	61,4	55,2	0	100
5	5995	G2	51,8	60,8	59,2	50,3	49	57,5	53,1	80
6	6667	FaZe	59,2	56,6	56,6	51,6	57,8	61,3	33,3	50
7	5752	Cloud9	60,1	42,5	61,5	62,8	59	50,2	56,2	100
8	5973	Liquid	60,5	51	54	56,5	62,4	61,1	59,2	55,6
9	8297	FURIA	65,2	57,3	0	63,3	60,8	61,3	64,6	40
10	4869	ENCE	56,8	62,4	59,7	67,3	46,7	41,5	55,6	11,1
11	5378	Virtus.pro	56,1	60,2	53,2	50,7	51	52,9	47,9	0
12	4494	MOUZ	53,7	47,6	55,4	42,1	50,8	54,6	40	42,9
13	10503	OG	56,1	50,3	64,1	52,3	50	50	56,8	50
14	7532	BIG	59,6	58,4	51,3	58,5	56,3	64,4	49,2	0
15	4411	Ninjas in P	64	58,7	60,1	53,1	53	62,1	46,2	56,2
16	4991	fnatic	57,6	75,6	76,8	46,3	60,2	57,3	60,2	70,6
17	8135	FORZE	56,8	50,8	47,6	44,4	48,1	50,4	48,5	20
18	4773	paiN	50,9	66,7	47,5	33,3	60	18,8	69,5	78,6
19	5005	Complexit	55,4	55,8	64,8	57	55,8	53,3	50	71,4
20	10278	9INE	50,6	37,5	60,5	55,3	50	46,1	0	100
21	11518	Bad News	45,1	52,3	25	51,3	58,3	58,9	49,6	75
22	7020	Spirit	65,3	67,8	83,6	66,7	79,2	72,7	76,9	25
23	11811	Monte	59,8	42,6	50,8	34,5	55,7	55,6	54,5	100
24	6248	TheMongc	48,6	42,1	48,5	40	60,7	57,5	50,6	80

Рис.2. Таблица со статистикой побед команд по картам.

HLTV-HANDLER - Представлен набором функций/типов/переменных, которые видны извне и к которым можно обращаться из других программ/-модулей. Они в свою очередь обращаются к hltv api и подтягивают базовые данные с сайта, которые впоследствии обрабатываются внутри нашей программы и заносятся в таблицу в нужной нам форме.

В результате до записи в таблицу имее возможность вывода в терминал следующие параметры объектов:

```
$ npx ts-node ./main.ts
[
  'teamId',  'name',
  'ranking', 'avg2_0',
  'ma1',     'ma1_5',
  'ma2_5'
]
[ 7175, 'Heroic', 904, 5.54, 'NaN', 'NaN', 'NaN' ]
[ 9565, 'Vitality', 895, 5.5600000000000005, 'NaN', 'NaN', 'NaN' ]
[ 4608, 'Natus Vincere', 772, 5.43, 'NaN', 'NaN', 'NaN' ]
[ 5995, 'G2', 697, 5.450000000000001, 'NaN', 'NaN', 'NaN' ]
[ 6667, 'FaZe', 680, 6.450000000000001, 'NaN', 'NaN', 'NaN' ]
[ 5752, 'Cloud9', 606, 5.53, 1, 1, 1 ]
[ 5973, 'Liquid', 418, 5.29, 'NaN', 'NaN', 'NaN' ]
```

Рис.3. Полученные данные для записи в таблицу 1.

```
[
  'teamId',      'name',
  'de_mirage',   'de_nuke',
  'de_dust2',    'de_train',
  'de_overpass', 'de_inferno',
  'de_vertigo',  'de_anubis'
]
[
  7175, 'Heroic',
  57.9, 66.9,
  45.9, 50.6,
  60.6, 52.1,
  61.7, 0
]
[
  9565, 'Vitality',
  55.6, 59.4,
  62.3, 50,
  68.1, 62.6,
  67.9, 71.4
]
[ 4608, 'Natus Vincere', 63.3, 59.6, 59.6, 65.7, 61.4, 55.2, 0, 100 ]
[
  5995, 'G2', 51.8,
  60.8, 59.2, 50.3,
  49,   57.5, 53.1,
  80
]
[
  6667, 'FaZe', 59.2,
  56.6, 56.6,   51.6,
  57.8, 61.3,   33.3,
  50
]
```

Рис.3. Полученные данные для записи в таблицу 2.

3.2. Программный код

```
            public addRaw<T>(sheet: string, entity: T) {
    let worksheet = this.workbook.getWorksheet(sheet);

    if (typeof worksheet === "undefined") {
      worksheet = this.workbook.addWorksheet(sheet);

      worksheet.columns = [
        ...Object.keys(entity as Object).map((x: any) => {
          if (x == "id1_id2") {
            return { header: x, key: x, width: x.length + 25};
          }
          return { header: x, key: x, width: x.length + 5};
        }),
      ];
    } else {
      // comparison of type of entity and existing columns
      // spoiler: must be equal
      let k: keyof typeof entity;
      for (k in entity)
        if ((worksheet._keys[k] as String) == (entity[k]
         as String))
          throw "Incompatible types";
    }

    worksheet.addRow(entity);
  }
```

Функция addRaw принимает некую сущность произвольного типа, в нашем случае это объект:

```
interface teamInfo {
teamId: number;
name: string;
ranking: number;
avg2_0: number;
ma1: number;
ma1_5: number;
ma2_5: number;
}
```

И преобразует её в строку, для каждого элемента выделен свой столбец.

Впоследствии эта строка записывается в область данных, которая была описана выше.

```
    public showWorkbook(workbook?: Workbook) {
    let currentWB = typeof workbook === "undefined" ?
     this.workbook : workbook;

    for (let i = 0; i < currentWB.worksheets.length; i++) {
      for (let j = 1; j <= currentWB.worksheets[i].rowCount;
       j++) {
        console.log(currentWB.worksheets[i].getRow(j)
        .values.slice(1));
      }
    }
  }
```

Функция showWorkbook выводит нынешнее состояние области данных, которую мы заполниоли при помощи addRaw.

Функции read/write:

```
    public async write(fileName: string) {
    await this.workbook.xlsx.writeFile(fileName);
    console.log(`File {fileName} is written`);
  }

  public async read(fileName: string) {
    const workbook = await this.workbook.xlsx
    .readFile(fileName);
    this.showWorkbook(workbook);
  }
```

Write - записывает наш workbook в файл с именем, в случае, если файла нет, то создаёт его. read - читает из файла с заданным именем, если файла нет - выдаёт ошибку.