ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Курсов проект по
"Системи за управление на бази данни"
Вариант 3

Изготвил:
Иван Димитров №12, 12Б

Глава 1

Използвани технологии

1.1 SQL

Език за структурирани заявки (SQL) е език за програмиране, който за създава, видоизменя, извлича и обработва данни от релационни системи за управление на бази данни. Стандартизирането му се поема от Международната организация по стандартизация (ISO) и Американският национален институт за стандарти (ANSI).

1.2 MySQL

MySQL е многопоточна, многопотребителска, SQL система за управление на бази данни. Оригиналният `и автор е MySQL AB, но след това бива закупена от Oracle Corporation. Некомерсиалният лиценз е съставен спрямо Общ публичен лиценз на GNU, а ако се използва от компании с комерсиална цел, то трябва да се закупи специален лиценз за ползване. Текущата стабилна версия на продукта е 8.0.

1.3 ORM

Обектно-релационно картографиране (Object–relational mapping) е концепцията, която представлява писане на заявки (прости или сложни), като се използва обектно-ориентирано програмиране с цел разработването на даден проект, който използва база данни, заместващи SQL.

1.4 TypeORM

TypeORM е ORM, който може да работи на платформи, които използват TypeScript и/или JavaScript (например NodeJS, Cordova, Ionic, React Native, NativeScript, Expo, Electron и др.). Целта му е лесното изграждане на бази данни за всякакъв вид приложения - от малки до големи с множество бази данни и сложни връзки.

1.5 TypeScript

TypeScript е супермножество на JavaScript, създадено от Microsoft. То надгражда JavaScript с предоставянето на типове, скоупове, интерфейси и много други. Езикът се компилира до JavaScript, което позволява поддръжката му от всички браузъри.

1.6 NestJS

Когато трябва да се създават мащабируеми и големи приложения на Node.js, често, за сървърната част, се използва библиотеката NestJS. Тя поддържа и препоръчва използването на TypeScript, но писането на JavaScript също е опция. Библиотеката залага най-вече на парадигмите на ООП (Обектно- -Ориентирано Програмиране), но комбинира и елементи от ФП (Функционално Програмиране) и ФРП (Функционално Реактивно Програмиране.). В NestJS за отделните функционалности се създават модули. Всеки модул е изграден от три неща: контролер или резолвър, провайдър и модел.

1.6.1 Контролер

Контролерът е основната част в осъществяването на комуникация между клиента и сървъра. Той приема заявките на клиента и му връща отговор. Всеки контролер отговаря за няколко маршрута,

като в повечето случай те са няколко. За да се разбере кой контролер отговаря за кой маршрут, NestJS има маршрутизиращ механизъм. Създаването на контролер става лесно, чрез декораторът @Controller(). Различните декоратори свързват класовете с необходимата метаинформация и по този начин Nest създава маршрутизираща карта. Чрез нея, заявките подадени от клиента се свързват със съответните контролери. В декоратора може да се специфицира маршрут зададен като префикс и по този начин се групират подобни маршрути. Ако се реализира REST приложно-програмен интерфейс, то за улеснение на програмиста съществуват 4 основни декоратора за HTTP заявките - @Get(), @Post(), @Put(), @Delete(). В тях също може да се постави префикс като маршрут, на който да отговаря дадена заявка чрез конкатенация на главния такъв.

1.6.2 Провайдър

Едно от основните свойства в NestJS е създаването на връзки между различните обекти. Това става с т.нар. зависимости, които се инжектират, за да създадат въпросните връзки. Тази функционалност принадлежи на провайдъра. Той е клас, анотиран с @Injectable() декоратора. Провайдъри са много от класовете на NestJS - хранилища, помощни класове, сървиси.

1.6.3 Модул

Цялата структура на едно NestJS приложение се организира от модули. Те се анотират с декораторът @Module(). Винаги има един основен модул, от който започва изграждането на връзките на приложението - вътрешната информационна структура, чрез която се обединяват модули, провайдъри, контролери, 25 резолвъри и други. По този начин организирането на структурата на приложението става ефективна и подредена. Така, в

архитектурата на приложението ще има множество модули, всеки съдържащ тясно свързан комплект от възможности.

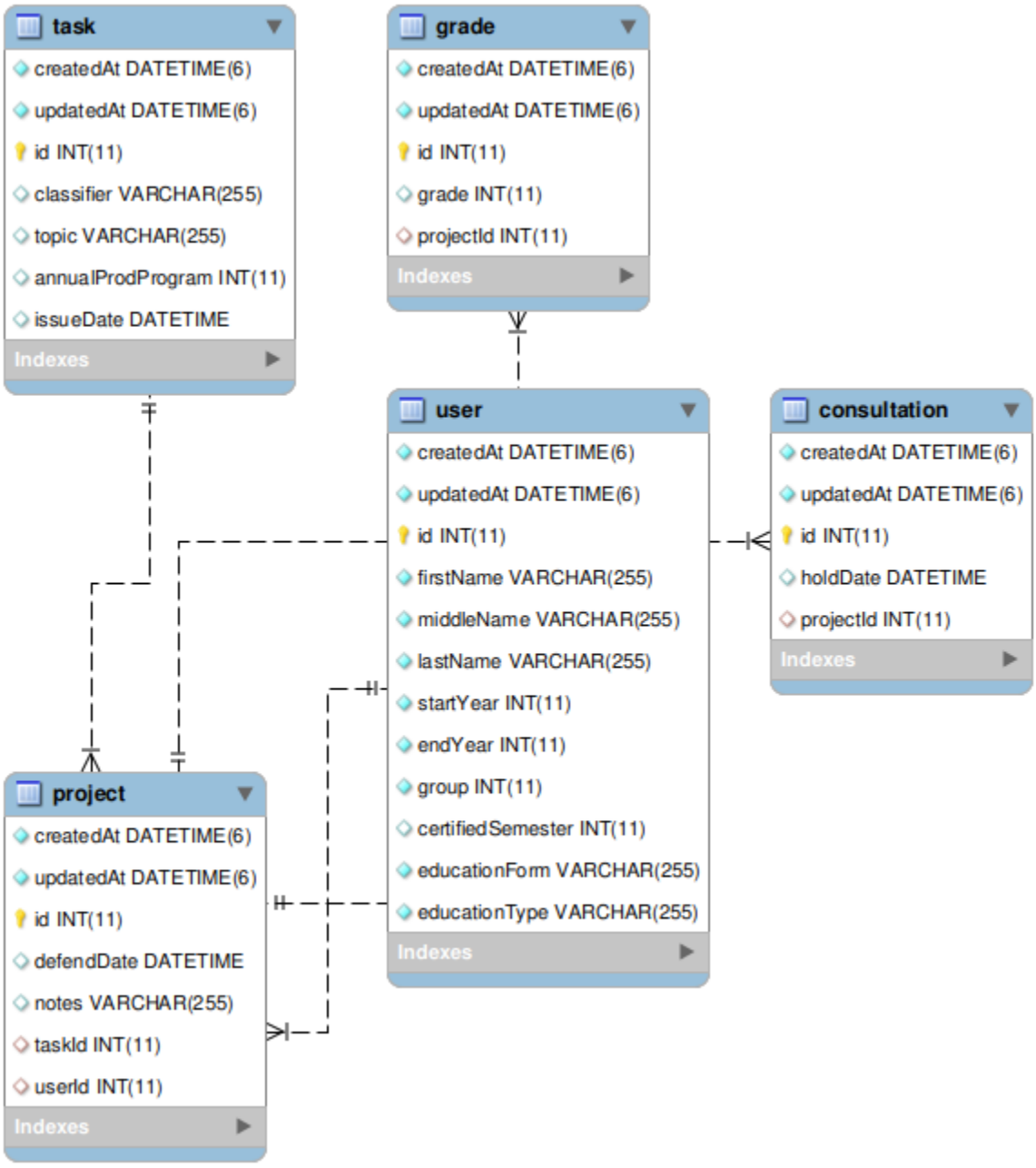
Глава 2

Функционални изисквания. Проектиране на базата данни

2.1 Функционални изисквания

- Базата данни да позволява въвеждане и да съдържа:
    - Данни за студента: фак. No, имена, учебна година;
    - Данни за вида обучение: образователна степен, форма, група;
    - Данни за заданието на проекта: Класификатор, Тема, годишна производствена програма;
    - Данни за N на брой консултации – дата;
    - Данни за защита на проекта: дата, оценка 1, оценка 2, оценка 3, оценка 4, оценка 5, обща оценка, окончателна оценка, бележки;
- Базата от данни да позволява търсене по критериите имена на студента, факултетен номер, учебна година, група, образователна степен, група, класификатор.
- Да е предвидена възможност за статистическа обработка на данните.

2.2 Проектиране на базата данни



2.2.1 Потребител

Всеки потребител (студент/ученик) съдържа следните колони:

- **createdAt** (DATETIME) - дата на създаване на записа
- **updatedAt** (DATETIME) - дата на последно актуализиране на записа
- **id** (INT) - уникален идентификатор, който се инкрементира автоматично с всеки запис
- **firstName** (VARCHAR) - първо име на потребителя
- **middleName** (VARCHAR) - бащино име на потребителя
- **lastName** (VARCHAR) - фамилия на потребителя
- **startYear** (INT) - начало на учебната година
- **endYear** (INT) - край на учебната година
- **group** (INT) - номер на група
- **certifiedSemester** (INT) - номер на заверен семестър на студента
- **educationForm** (VARCHAR) - определя формата на обучение - задочно или редовно
- **educationType** (VARCHAR) - ниво на образователна степен - бакалавър, магистър или доктор

2.2.2 Проект

Всеки проект съдържа следните колони:

- **createdAt** (DATETIME) - дата на създаване на записа
- **updatedAt** (DATETIME) - дата на последно актуализиране на записа
- **id** (INT) - уникален идентификатор, който се инкрементира автоматично с всеки запис
- **defendDate** (DATETIME) - дата на защита на проекта
- **notes** (VARCHAR) - бележки към проекта под формата на стринг
- **taskId** (INT) - външен ключ, който сочи към първичния ключ на заданието на дадения проект

- **userId** (INT) - външен ключ, който сочи към първичния ключ на потребителя, притежаващ дадения проект
- ***avgGrade** (DOUBLE) - средноаритметичната оценка на даден проект. Изчислява се динамично спрямо наличните оценки.
- ***finalGrade** (INT) - финалната оценка на даден проект. Закръглява се спрямо средноаритметичната оценка.

  *полетата **са** налични, но не са част от базата данни, а се изчисляват динамично спрямо ORM и специални анотации.

2.2.3 Задание

Всяко задание съдържа следните колони:

- **createdAt** (DATETIME) - дата на създаване на записа
- **updatedAt** (DATETIME) - дата на последно актуализиране на записа
- **id** (INT) - уникален идентификатор, който се инкрементира автоматично с всеки запис
- **classifier** (VARCHAR) - класификатор
- **topic** (VARCHAR) - тема на заданието
- **annualProdProgram** (INT) - годишна производствена програма
- **issueDate** (DATETIME) - дата на издаване на техническото задание

2.2.4 Оценка

Всяка оценка съдържа следните колони:

- **createdAt** (DATETIME) - дата на създаване на записа
- **updatedAt** (DATETIME) - дата на последно актуализиране на записа
- **id** (INT) - уникален идентификатор, който се инкрементира автоматично с всеки запис
- **grade** (INT) - оценка между 2 и 6

- **projectId** (INT) - външен ключ, който сочи към първичния ключ на даден проект

2.2.5 Консултация

Всяка консултация съдържа следните колони:

- **createdAt** (DATETIME) - дата на създаване на записа
- **updatedAt** (DATETIME) - дата на последно актуализиране на записа
- **id** (INT) - уникален идентификатор, който се инкрементира автоматично с всеки запис
- **holdDate** (DATETIME) - дата на провеждане на консултация
- **projectId** (INT) - външен ключ, който сочи към първичния ключ на даден проект

2.2.6 OneToOne релации между таблиците

- Между "**Проект**" и "**Задание**" таблиците (**taskId** ---> **id**) - един проект може да има едно задание и едно задание може да се съдържа само в един проект.

2.2.7 OneToMany (ManyToOne) релации между таблиците

- Между "**Проект**" и "**Консултация**" таблиците (**id** <--- **projectId**) - един проект може да има множество консултации, но един запис на консултация може да реферира само един проект.
- Между "**Проект**" и "**Оценка**" таблиците (**id** <--- **projectId**) - един проект може да има множество оценки, но един запис на оценка може да реферира само един проект.
- Между "**Потребител**" и "**Проект**" таблиците (**id** <--- **userId**) - един потребител може да има множество проекти, но един проект може да принадлежи само на един потребител.

Глава 3

Начин на използване.
Ръководство за инсталиране

3.1 Начин на използване

3.1.1 API на Потребител

- **GET** *http://localhost:3000/users*

  Извличат вече създадени потребители. Използва query параметри. **Всички те са опционални**. При нито един зададен параметър се взимат всички записи на потребители от базата. Параметрите са:

  - **id** (int) - идентификатор на потребителя
  - **firstName** (string) - първо име на потребителя
  - **middleName** (string) - бащино име на потребителя
  - **lastName** (string) - фамилно име на потребителя
  - **schoolYear** (string) - начало и край на учебната година
  - **group** (int) - номер на група
  - **educationType** (string) - ниво на образователна степен
  - **classifier** (string) - класификатор
  - **handed** (boolean) - дали ученикът е предал проекта или не
  - **grades** (int) - брой оценки

  Примерна заявка:

  *http://localhost:3000/users?id=1&firstName=Иван&middleName=Огнянов&lastName=Димитров&schoolYear=2016/2017&group=228&educationType=бакалавър&classifier=Елементи&handed=**true**&grades=5*

- **POST** *http://localhost:3000/users*

  Създава се потребител. Използва request body. То трябва да съдържа:

  - **firstName** (string) - първо име на потребителя
  - **middleName** (string) - бащино име на потребителя
  - **lastName** (string) - фамилно име на потребителя
  - **startYear** (int) - начало на учебната година
  - **endYear** (int) - край на учебната година
  - **group** (int) - номер на група
  - ***certifiedSemester** (int) - номер на заверен семестър на студента
  - **educationForm** (string) - форма на обучение
  - **educationType** (string) - ниво на образователна степен
  - ***projectsIds** (int[]) - масив с идентификатори на проекти, които ще принадлежат на ученика

    *опционални незадължителни

  Примерно тяло на заявка:

```
{
        “firstName”: “Иван”,
        “middleName”: “Огнянов”,
        “lastName”: “Димитров”,
        “startYear”: 2016,
        “endYear”: 2017,
        “group”: 228,
        “certifiedSemester”: 2,
        “educationForm”: “редовно”,
        “educationType”: “бакалавър”,
        “projectsIds”: [1, 2, 3, 6]
}
```

- **PUT** *http://localhost:3000/users/:id*

  Аналогично на **POST** заявката, но използва идентификатор като параметър и **всички параметри на тялото са опционални**.

3.1.2 API на Проект

- **GET** *http://localhost:3000/projects*

  Извличат се всички проекти.

- **POST** *http://localhost:3000/projects*

  Създава се проект. Използва request body. То трябва да съдържа:

  - ***taskId** (int) - идентификатор на задание, което ще принадлежи на проекта
  - ***consultationsIds** (int[]) - масив с идентификатори на консултации, които ще принадлежат на проекта
  - ***defendDate** (Date) - дата на защита на проекта
  - ***gradesIds** (int[]) - масив с идентификатори на оценки, които ще принадлежат на проекта
  - ***notes** (string) - бележки към проекта
  - **userId** (int) - идентификатор на потребител, на когото принадлежи проекта

    *опционални незадължителни

  Примерно тяло на заявка:

```
{
        “taskId”: 2,
        “consultationsIds”: [7, 8, 3, 6],
        “defendDate”: “Apr 12 2021 22:54:12”,
        “notes”: “Липсва задание”,
```

```
    “userId”: 1
}
```


- **PUT** *http://localhost:3000/projects/:id*

  Аналогично на **POST** заявката, но използва идентификатор като параметър и **userId** е опционален член на тялото на заявката.

3.1.3 API на Задание

- **GET** *http://localhost:3000/tasks*

  Извличат се всички задания.

- **POST** *http://localhost:3000/tasks*

  Създава се задание. Използва request body. **Всички параметри са опционални.** То трябва да съдържа:

  - **classifier** (string) - класификатор на заданието
  - **topic** (string) - тема на заданието
  - **annualProdProgram** (int) - годишна производствена програма
  - **issueDate** (Date) - дата на издаване на техническото задание

  Примерно тяло на заявка:


```
{
    “classifier”: “Елементи”,
    “topic”: “Корпусни елементи за калкулатор”,
    “annualProdProgram”: 45000,
    “issueDate”: “Apr 12 2021 22:54:12”
}
```

- **PUT** *http://localhost:3000/tasks/:id*

  Аналогично на **POST** заявката, но използва идентификатор като параметър.

3.1.4 API на Оценка

- **GET** *http://localhost:3000/grades*

  Извличат се всички оценки.

- **POST** *http://localhost:3000/grades*

  Създава се оценка. Използва request body. То трябва да съдържа:

  - ***grade** (int) - оценка от 2 до 6
  - **projectId** (int) - идентификатор на проект, към който ще принадлежи оценката

    *опционални незадължителни

  Примерно тяло на заявка:

```
{
        “grade”: 6,
        “projectId”: 1
}
```

- **PUT** *http://localhost:3000/grades/:id*

  Аналогично на **POST** заявката, но използва идентификатор като параметър и **projectId** е опционален член на тялото на заявката.

3.1.5 API на Консултация

- **GET** *http://localhost:3000/consultations*

  Извличат се всички консултации.

- **POST** *http://localhost:3000/consultations*

  Създава се консултация. Използва request body. То трябва да съдържа:

  - ***holdDate** (Date) - дата, на която ще се проведе консултацията
  - **projectId** (int) - идентификатор на проект, към който ще принадлежи консултацията

    ***опционални незадължителни**

  Примерно тяло на заявка:

```
{
        “holdDate”: “Apr 12 2021 22:54:12”,
        “projectId”: 1
}
```

- **PUT** *http://localhost:3000/consultations/:id*

  Аналогично на **POST** заявката, но използва идентификатор като параметър и **projectId** е опционален член на тялото на заявката.

3.2 Ръководство за инсталиране

3.2.1 Необходими инсталирани програми

- MariaDB 10.0+ или MySQL 8.0+

- Node.js 12+ и NPM

3.2.2 Необходима конфигурация

В главната директория на проекта е наличен файл **.env.TEMPLATE**. Той трябва да се копира и прекръсти на **.env**. В него има полета, които трябва да се попълнят за конфигурация на базата данни (полето **DB_SOCKET** не е задължително). Примерна конфигурация:

```
DB_NAME=subdproject
DB_HOST=localhost
DB_PORT=3306
DB_USERNAME=root
DB_PASSWORD=changeme
DB_DRIVER=mariadb (or mysql)
DB_SOCKET=
```

След това, отново в главната директория на проекта трябва да се отвори терминал и да се изпълнят следните команди (в дадения ред):

- за инсталиране на модулите:

  $ npm install

- за компилиране на проекта до production версия:

  $ npm run build

- за пускане на проекта:

  $ npm run start:prod

- **(допълнително)** за пускане на проекта в dev среда (не е нужен build):

  $ npm run start:dev