Съдържание

[1. Увод 4](#_Toc196688883)

[1.1. Описание на целите и задачите на проекта. 4](#_Toc196688884)

[1.2. Значимост на избраната тема. 4](#_Toc196688885)

[2. Концепция на работа с бази данни в Django 5](#_Toc196688886)

[2.1. Въведение 5](#_Toc196688887)

[2.2. Основи на работа с бази данни в Django 5](#_Toc196688888)

[2.2.1 Модели 5](#_Toc196688889)

[2.2.2 Връзки между модели 6](#_Toc196688890)

[2.2.3 Видове полета 6](#_Toc196688891)

[2.2.4 Работа с бази данни 6](#_Toc196688892)

[3. Реализация на обектно-релационни модели в Django 7](#_Toc196688893)

[3.1. Въведение 7](#_Toc196688894)

[3.2. Основи на обектно-релационните модели 8](#_Toc196688895)

[3.2.1 Дефиниране на модели 8](#_Toc196688896)

[3.2.2 Връзки между модели 8](#_Toc196688897)

[3.2.3 Работа с модели в Django 9](#_Toc196688898)

[4. Видове полета и заявките в Django 10](#_Toc196688899)

[4.1. Въведение 10](#_Toc196688900)

[4.2. Видове полета в Django 10](#_Toc196688901)

[4.2.1 Основни типове полета 10](#_Toc196688902)

[**2. Допълнителни типове полета** 12](#_Toc196688903)

[**4.3.** **Заявки в Django** 12](#_Toc196688904)

[4.3.1 Основни операции 12](#_Toc196688905)

[4.3.2 2. Сложни заявки 13](#_Toc196688906)

[ **Сортиране на записи** 14](#_Toc196688907)

[5. Описание на съществуващите решения и литература 14](#_Toc196688908)

[5.1. Въведение 14](#_Toc196688909)

[5.2. Съществуващи решения за работа с бази данни 14](#_Toc196688910)

[5.2.1 Релационни бази данни 14](#_Toc196688911)

[5.2.2 Нерелационни бази данни (NoSQL) 15](#_Toc196688912)

[5.2.3 ORM решение в Django 15](#_Toc196688913)

[5.3. Литература 16](#_Toc196688914)

[6. Избор на технологии и алгоритми 16](#_Toc196688915)

[6.1. Въведение 16](#_Toc196688916)

[6.2. Обоснованост на избора на Django 17](#_Toc196688917)

[6.2.1 Простота на използване 17](#_Toc196688918)

[6.2.2 Вградени функции 17](#_Toc196688919)

[6.2.3 Обширна общност и документация 17](#_Toc196688920)

[6.3. Алтернативи и защо не са избрани? 18](#_Toc196688921)

[6.3.1 Flask 18](#_Toc196688922)

[6.3.2 Ruby оn Rails 18](#_Toc196688923)

[6.3.3 ASP.NET 18](#_Toc196688924)

[7. Основни характеристики и функционалности 18](#_Toc196688925)

[7.1. Архитектура на Django 19](#_Toc196688926)

[7.1.1 Модел 19](#_Toc196688927)

[7.1.2 Визуализация 19](#_Toc196688928)

[7.1.3 Шаблони 19](#_Toc196688929)

[7.2. Основни функции на Django 19](#_Toc196688930)

[7.2.1 Административен панел 19](#_Toc196688931)

[7.2.2 Поддръжка на различни бази данни 20](#_Toc196688932)

[7.2.3 Разширяемост 20](#_Toc196688933)

[8. Основи на работа с бази данни 20](#_Toc196688934)

[8.1. Въведение 20](#_Toc196688935)

[8.2. Определение за база данни 20](#_Toc196688936)

[8.3. Видове бази данни 21](#_Toc196688937)

[8.3.1 Релационни бази данни 21](#_Toc196688938)

[8.3.2 Нерелационни бази данни. 21](#_Toc196688939)

[8.4. Основни операции с бази данни 21](#_Toc196688940)

[8.5. Свързване с бази данни 22](#_Toc196688941)

[9. Практическа част 23](#_Toc196688942)

[9.1. Обяснение на моделите в Django за анкети 23](#_Toc196688943)

[9.1.1 Въведение 23](#_Toc196688944)

[9.1.2 Взаимоотношения между моделите 26](#_Toc196688945)

[ **Статистики** 26](#_Toc196688946)

[9.1.3 Защо ползваме Django модели? 26](#_Toc196688947)

[9.2. Обобщение 27](#_Toc196688948)

[10. Заключение 27](#_Toc196688949)

[**9.** **Литература:** 29](#_Toc196688950)

# Увод

## Описание на целите и задачите на проекта.

Днешният свят е изправен пред нови предизвикателства, свързани с управлението на данни. С напредъка на технологията, необходимостта от ефективно и безопасно съхранение, манипулиране и извличане на информация става все по-голяма. В сърцето на тези процеси стоят базите данни, които се явяват ключов компонент в разработката на съвременни уеб приложения. В контекста на днешната информационна ера, интеграцията на релационни и нерелационни бази данни в уеб платформите не само подобрява производителността на приложенията, но и предоставя на потребителите достъп до информация в реално време.

Този дипломен проект има за цел да изследва концепцията за работа с бази данни в популярния уеб фреймуорк Django. Основната задача е да се проучат основите на релационните бази данни, начина по който Django управлява данните чрез своя обектно-релационен машинг (ORM), както и да се представят различните методи за работа с данни — от дефиниране на модели до извършване на сложни заявки. В допълнение, проектът ще разгледа реализирането на обектно-релационни модели и тяхната интеграция в Django, с цел да предостави на разработчиците мощни и интуитивни инструменти за работа с бази данни.

## Значимост на избраната тема.

Значимостта на избраната тема е многопластова. Възможността за ефективно управление на данни е с решаващо значение за бизнеса, образованието и почти всички сфери на живота. Прилагането на концепции за работа с бази данни, особено в контекста на уеб разработката, е не само актуално, но и необходимо за всяка организация, която желае да оптимизира своите процеси и да увеличи своето конкурентно предимство. На фона на бързо променящите се технологии, успешното овладяване на Django и неговите функционалности за работа с бази данни ще бъде ключов фактор за кариерното развитие на всеки програмист.

# Концепция на работа с бази данни в Django

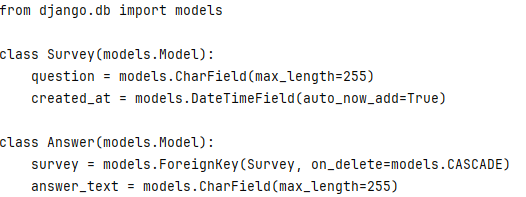
## Въведение

Работата с бази данни е основен аспект на повечето уеб приложения. При разработката на приложения с Django, един от най-силните аспекти на фреймуорка е неговата интеграция с бази данни. Django предоставя мощни инструменти за работа с релационни бази данни, което позволява на разработчиците лесно да извършват операции по създаване, четене, актуализиране и изтриване на данни (CRUD операции).

## Основи на работа с бази данни в Django

### Модели

В Django, базите данни се представят чрез модели. Моделите определят структурата на данните и как те ще бъдат организирани в базата данни. Django използва проект, наречен Object-Relational Mapping (ORM), който позволява на разработчиците да работят с базата данни чрез Python класове вместо чрез SQL команди. Всеки модел е свързан с таблица в базата данни и атрибутите на модела съответстват на колоните в таблицата. Пример за дефиниране на модел в Django:



В примера по-горе, дефинираме два модела: Survey и Answer. Тези класове ще позволят на приложението да работи с таблици, свързани с анкети и техните отговори.

### Връзки между модели

Django дава възможност за създаване на различни типове връзки между моделите:

* **Едно-към-едно (One-to-One):** Позволява свързване на два модела по такъв начин, че за всяка единица от единия модел, може да съществува само една единица от другия.
* **Едно-към-много (One-to-Many):** Най-често срещаният тип връзка, където един запис от един модел може да се свърже с множество записи от друг модел.
* **Много-към-много (Many-to-Many)**: Позволява на множество записи от един модел да съществуват в отношения с множество записи от друг модел.

Чрез използването на Foreign Key и ManyToManyField полета във модела, Django автоматично управлява сложността на релациите.

### Видове полета

Django предлага разнообразие от типове полета, които могат да се използват в моделите:

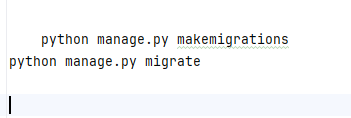
* Char Field: За текстови низове с ограничена дължина.
* Text Field: За по-дълги текстови стойности.
* Integer Field: За съхранение на цели числа.
* DateTimeField: За съхранение на дати и часове.
* Boolean Field: За булеви стойности (истина/лъжа).

Изборът на правилния тип поле е важен, тъй като той диктува как ще бъдат съхранявани и манипулирани данните в базата.

### Работа с бази данни

* **Инициализация и миграции**

След дефинирането на моделите, следва следващата стъпка — инициализация на базата данни. Django използва система за миграции, която проследява промените в моделите и синхронизира базата данни с новите структури. Миграциите позволяват на разработчиците да добавят, променят или изтриват таблици и полета, без да губят данни.



Командите по-горе генерират скриптове за миграции и прилагат промените в базата данни.

* **Извършване на заявки с ORM**

Django ORM позволява извършване на заявки чрез използване на Python код. Разработчиците могат да извършват CRUD операции по различни начини, без да пишат директно SQL. Например, за да получим всички анкети, можем да напишем:



За да добавим нов отговор:



Тази простота е основна причина Django да бъде предпочитан при разработка на приложения, свързани с бази данни.

# Реализация на обектно-релационни модели в Django

## Въведение

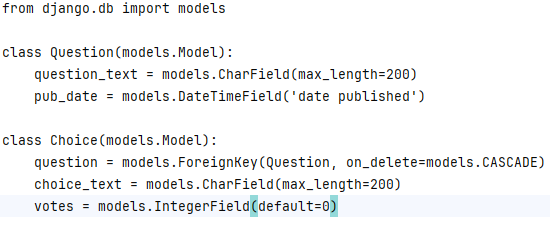
Обектно-релационното моделиране (ORM) е концепция, която свързва обектно-ориентираното програмиране с релационните бази данни. В Django, ORM позволява на разработчиците да взаимодействат с базата данни, използвайки Python класове вместо SQL заявки. Това подобрява производителността и намалява сложността при работа с данни. В този раздел ще разгледаме как се реализират обектно-релационни модели в Django, как се създават модели, как взаимодействат различните модели помежду си, и как ORM опростява манипулацията на данни.

## Основи на обектно-релационните модели

### Дефиниране на модели

В Django, всеки модел представлява таблица в базата данни. Моделите се дефинират като Python класове, наследяващи на models. Model, и всеки атрибут на класа представлява колона в таблицата. При дефиниране на моделите, можем да зададем различни типове полета, които описват вида на данните, които ще се съхраняват.

Пример:



В примера, Question и Choice дефинират две таблици в базата данни, свързани помежду си чрез Foreign Key, което показва множество отговори (избори) за един въпрос.

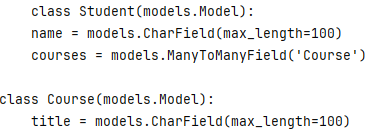
### Връзки между модели

Django предоставя различни механизми за създаване на връзки между модели, включително:

**Едно-към-едно**: Дефинира се с OneToOneField. Ползва се, когато искаме един запис от единия модел да съответства на точно един запис от другия модел.

**Едно-към-много**: Дефинира се с Foreign Key. Например, в горния код, Choice има Foreign Key към Question, което означава, че един въпрос може да има множество избори.

**Много-към-много**: Дефинира се с ManyToManyField. Този тип връзка е полезен, когато искаме множество записи от един модел да са свързани с множество записи от друг модел.



Тук, един студент може да бъде записан в множество курсове и всеки курс може да има много студенти.

### Работа с модели в Django

* **Създаване и миграции**

След като сме дефинирали моделите, следва следващата стъпка — миграции. Миграциите са инструмент за синхронизиране на моделите с базата данни. Django автоматично генерира миграции въз основа на промените в моделите.

****

Командата make migrations създава нова миграция, а migrate прилага промените в базата данни, създавайки новите таблици или актуализирайки съществуващите.

* **Манипулиране на данни**

След като моделите са създадени и мигрирани, можем да взаимодействаме с тях чрез Django ORM. Основните операции, които можем да извършваме, включват:

* **Създаване на нови записи:**

****

* **Четене на записи:**

****

* **Актуализиране на записи:**

****

Тези операции ни позволяват лесно да управляваме данни, без да пишем SQL заявки.

# Видове полета и заявките в Django

## Въведение

Django е мощен уеб фреймуорк, който предоставя интуитивни механизми за работа с релационни бази данни. Важно е да разберем различните типове полета, които можем да използваме в моделите, и как да извършваме заявки за манипулация на данни чрез Django ORM. Тази част от проекта ще разгледа широко наличните полета, техните характеристики и типичните заявки, които можем да извършваме

## Видове полета в Django

### Основни типове полета

* Django предоставя разнообразие от полета, с които можем да моделираме данни в базата. Основните типове полета включват:

**Char Field**: Използва се за кратки текстови стойности.

Например:



**Text Field**: За по-дълги текстове, когато е необходимо:



**Integer Field**: За цели числа:



**Float Field**:

* Използва се за дробни числа.
* Пример:



**Boolean Field**:

* Използва се за логически стойности (True/False).
* Пример:



**Date Field и DateTimeField**:

* Използват се за дати и дати с времева информация.
* Пример:



**Email Field**:

* Използва се за съхраняване на имейл адреси.
* Пример:



**2. Допълнителни типове полета**

**File Field**:

* Използва се за качване на файлове
* Пример:



**Image Field**:

* Използва се за качване на изображения.
* Пример:



**URLField**:

* Използва се за съхраняване на URL адреси.
* Пример:



* 1. **Заявки в Django**

Заявките в Django се извършват чрез ORM, което позволява на разработчиците да работят с бази данни чрез Python обекти, вместо да пишат SQL заявки. Това не само опростява работата с базата данни, но също така я прави по-сигурна.

### Основни операции

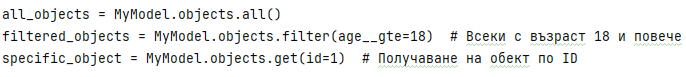
**Създаване на записи**:

* За създаване на нов запис в базата данни, просто инстанцирате модела и извиквате save().
* Пример:

****

**Четене на записи**:

* Можете да извличате записи с методите all(), filter(), get().
* Пример:



**Обновяване на записи**:

* Можете да актуализирате съществуващи записи.
* Пример:



**Изтриване на записи**:

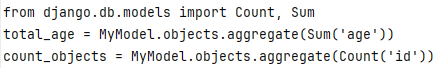
* Изтриването на записи също е просто.
* Пример:



### 2. Сложни заявки

**Агрегати**:

* Можете да сумирате, изчислявате средната стойност и т.н.
* Пример:



**Наследяване на филтри**:

* Можете да комбинирате филтри.
* Пример:



* **Сортиране на записи**

Също така можете да сортирате резултатите от запитванията:



# Описание на съществуващите решения и литература

## Въведение

В съвременния свят използването на бази данни и тяхната интеграция в уеб приложения е от решаващо значение за успеха на софтуерните решения. Особено в контекста на фреймуорка Django, който предлага интуитивен и мощен ORM, разработчиците могат да проектират и реализират решения, които са както функционални, така и ефективни. В този раздел ще разгледаме съществуващите решения, свързани с бази данни и Django, както и литературата, която обхваща тези теми.

## Съществуващи решения за работа с бази данни

### Релационни бази данни

Релационните бази данни (RDBMS) са най-често използваните системи за управление на бази данни, като предлагат структурирано съхранение на данни. Най-популярните RDBMS технологии включват:

**MySQL**: Широко използвана база данни, известна със своята производителност и зададените функционалности. MySQL е предпочитана за много уеб приложения и е основен компонент за Django.

**PostgreSQL**: Още една високо оценена релационна база данни, която се отличава с разширени функции, като поддръжка на JSON, комплексни запитвания и разширяемост. Често е избор на разработчици, търсещи допълнителна гъвкавост.

**SQLite**: Лека и самостоятелна база данни, идеална за разработки и тестови среди. Django използва SQLite по подразбиране, което го прави достъпен за начинаещи.

### Нерелационни бази данни (NoSQL)

В последните години нерелационните бази данни спечелиха популярност, особено за приложения с големи обеми данни и динамично съдържание. Някои известни решения включват:

**MongoDB**: Документно-ориентирана база данни, която предлага гъвкаво и просто съхранение на данни. MongoDB е предпочитана за приложения с неструктурирани данни.

**Cassandra**: Система с висока производителност за съхранение на данни, целяща да предостави висока наличност и устойчивост. Често се използва в приложения с голям обем на данни.

### **ORM решение в Django**

Django вгражда собствен ORM, което позволява на разработчиците да взаимодействат с базите данни чрез Python код, вместо да пишат SQL заявки. Това улеснява манипулирането на данни и методите за каширане, което подобрява производителността на приложенията.

**Model система**: В Django моделите представляват класове, в които се дефинират полетата и поведението на данните. Този подход опростява работата с бази данни и в същото време осигурява мощен и интуитивен интерфейс.

**Query Set**: Обектите, които представляват колекции от обекти, и предлагат методи за филтриране, агрегати и манипулация на данните.

## Литература

Съществуваща литература относно работата с бази данни и фреймуърка Django обхваща множество книги, статии и онлайн ресурси. По-долу са посочени основни примери:

**"Django Ford Beginners" на William S. Vincent**: Тази книга предлага изчерпателно ръководство за начинаещи, които искат да се запознаят с Django. Съдържа практически примери и обяснява основите на работа с бази данни.

**"Two Scoops ок Django" на Audrey Sony Greenfield и Danone Sony Greenfield**: Тази книга предоставя множество полезни съвети за разработчици на Django, включително технологии за работа с бази данни и оптимизация на производителността.

**Документация на Django**: Официалната документация на Django е незаменим ресурс, който предлага информация за инсталиране, конфигуриране и работа с библиотеката. Тя съдържа примери и обяснения за основните компоненти, включително моделите и ORM.

**"Mastering Django" на Intel George**: Тази книга предоставя възможности за усъвършенстване на уменията в Django и разглежда в дълбочина структурата на базите данни и модела на данни в контекста на Django.

# Избор на технологии и алгоритми

## Въведение

При разработването на приложения с работа с бази данни, изборът на технологии и алгоритми играе решаваща роля за успеха на проекта. В настоящия раздел ще обосновем избора на фреймуърка Django, ще разгледаме алтернативите му и ще анализираме причините, поради които не са избрани.

## Обоснованост на избора на Django

### Простота на използване

Django е известен със своите интуитивни интерфейси и вградени функционалности, които позволяват на разработчиците бързо да стартират нови проекти. Структурата на проектите в Django е ясна и организирана, което позволява лесно управление на кода и компонентите. Тази простота е изключително важна за нови разработчици, които искат бързо да постигнат работещо приложение.

### **Вградени функции**

Django предлага множество вградени функции, които съкращават времето за разработка. Някои от тях включват:

**Административен панел**: Django автоматично генерира административен интерфейс за управление на данни, което спестява време на разработчиците.

**Сигурност**: Django включва вградени механизми за защита от често срещани уязвимости, като Cross-site Scripting (XSS), SQL инжекции и Cross-Site Request Forgery (CSRF). Високото ниво на сигурност е особено важно при работа с чувствителни данни в приложения.

**ORM (Object-Relational Mapping)**: Django ORM позволява разширено взаимодействие с релационни бази данни, което осигурява проста и ефективна работа с данни.

### Обширна общност и документация

Друга важна причина за избора на Django е неговата огромна общност и наличието на обширна документация. Разработчиците лесно могат да намерят решения на проблеми, примери и ръководства в онлайн ресурсите. Активната поддръжка от общността допринася за непрекъснатото развитие на фреймуърка.

## Алтернативи и защо не са избрани?

Въпреки, че Django предлага много предимства, съществуват и алтернативни технологии, които не бяха избрани за настоящата разработка. Някои от тях включват:

### Flask

Flask е минималистичен фреймуърк, който предлага по-голяма гъвкавост в създаването на приложения. Въпреки това, за разлика от Django, Flask не предлага толкова много вградени инструменти. Изборът на Flask би изисквал допълнителна работа за интегриране на функции, които в Django са предварително предоставени.

### Ruby оn Rails

Ruby оn Rails е много мощен фреймуърк за уеб приложения, базиран на език Ruby. Въпреки че предлага богат набор от функции, той не е толкова популярен сред разработчиците на Python и поради това може да затрудни търсенето на помощ и ресурси. Изборът на Django, който е проектиран да работи с Python, позволява по-добра интеграция в съществуващи Python проекти.

### **ASP.NET**

ASP.NET е платформа, разработена от Microsoft, особено подходяща за проекти, свързани с Windows. Въпреки, че предлага силни функции и отлична интеграция с Windows услуги, ASP.NET изисква специфични знания, свързани с Windows, което може да ограничени възможностите за разширяване и сложност на разработката.

# Основни характеристики и функционалности

Проектиран за бързо разработване на уеб приложения, Django предлага много вградени функции и инструменти, които опростяват процеса на разработка и позволяват на разработчиците да се фокусират върху важните аспекти на проекта. Създаден от експерти за разработка на новини и информационни сайтове, Django е изключително приложение с отворен код и е широко използван за изграждане на широк спектър от уеб приложения.

## Архитектура на Django

Django следва архитектурния модел Model-View-Template (MVT), който е подобен на Model-View-Controller (MVC), прилаган в много други уеб фреймуъркове. MVT разделя логиката на приложението, представянето на данните и визуализацията, което улеснява поддръжката и разширяемостта на приложенията.

### Модел

Моделите в Django определят структурата на данните, които ще се съхраняват в релационната база данни. Всеки модел е представен от Python клас, който описва таблица в базата данни и се използва за управление на данни. Django ORM позволява лесно извършване на CRUD операции (създаване, четене, обновяване, изтриване).

### Визуализация

Визуализацията (или представяне на данни) в Django се осъществява чрез шаблоните, които се използват за генериране на HTML. Тези шаблони позволяват динамично представяне на информацията, базирана на данни от моделите. Django предоставя вградена система за шаблони, която позволява на разработчиците лесно да включват логика и условия в представянето на данни.

### Шаблони

Шаблоните в Django се пишат на специфичен език за шаблони, който позволява на разработчиците да включват условия, цикли и извиквания на функции в HTML. Те също така предоставят удобни механизми за социализиране на данни в JSON и XML формати, което е полезно за API приложения.

## Основни функции на Django

### Административен панел

Една от най-значимите предимства на Django е автоматично генерираният административен панел, който позволява администраторите да управляват базите данни и данни по интуитивен начин. Този панел е изключително полезен за администриране и мониторинг на приложенията, спестявайки време за разработчиците.

### Поддръжка на различни бази данни

Django поддържа множество релационни бази данни, включително PostgreSQL, MySQL, SQLite и Oracle. С това разнообразие, разработчиците могат лесно да изберат система за управление на бази данни, която най-добре отговаря на техните нужди.

### Разширяемост

Django дава възможност за разширение и интеграция с множество библиотеки и пакети, което позволява на разработчиците да добавят нови функционалности към своите приложения.

# Основи на работа с бази данни

## Въведение

Базите данни играят ключова роля в съвременните информационни технологии. Те служат за съхранение, управление и манипулиране на данни, което е от съществено значение за почти всяка приложна система. Основната цел на работата с бази данни е да осигури ефективно обработване и достъп до информация, която да спомага за взимането на решения и подобряване на бизнес процесите.

## Определение за база данни

Базата данни е организирана колекция от свързани данни, които могат да бъдат достъпни и управлявани за различни приложения. Основната архитектура на една база данни включва структурирани таблици, които съдържат записи и полета. Всеки запис представя един елемент на информация, а полетата съдържат атрибути на този елемент.

## Видове бази данни

### Релационни бази данни

Релационните бази данни (RDBMS) са най-разпространените типове бази данни. Те организират данните в таблици, свързани помежду си чрез ключове. Основните характеристики на релационните бази данни включват:

**Структурирани таблици**: Данните са организирани в редове и колони.

**Първоначално и уникално идентифициране**: Всеки ред (или запис) може да бъде уникално идентифициран чрез първичен ключ.

**SQL (Structured Query Language)**: Стандартният език за работа с релационни бази данни, който позволява извършване на запитвания, добавяне, обновяване и изтриване на данни.

### Нерелационни бази данни.

Нерелационните (или NoSQL) бази данни са проектирани да прилагат различни модели на съхранение на данни, включително:

**Документно ориентирани**: Данните са съхранявани в формати като JSON или XML.

**Ключ-стойност бази**: Данните се съхраняват под формата на ключове и стойности, което позволява бърз достъп до информация.

**Графови бази**: Предназначени за управление на данни, свързани чрез графови структури.

## Основни операции с бази данни

Работата с бази данни включва няколко основни операции, които могат да бъдат разделени в четири основни категории, известни като CRUD операции:

* **C (Create)**: Добавяне на нови записи в базата данни.
* **R (EAC)**: Извличане и четене на данни от базата данни.
* **U (Update)**: Обновяване на съществуващи записи.
* **D (Delete)**: Изтриване на записи от базата данни.

## Свързване с бази данни

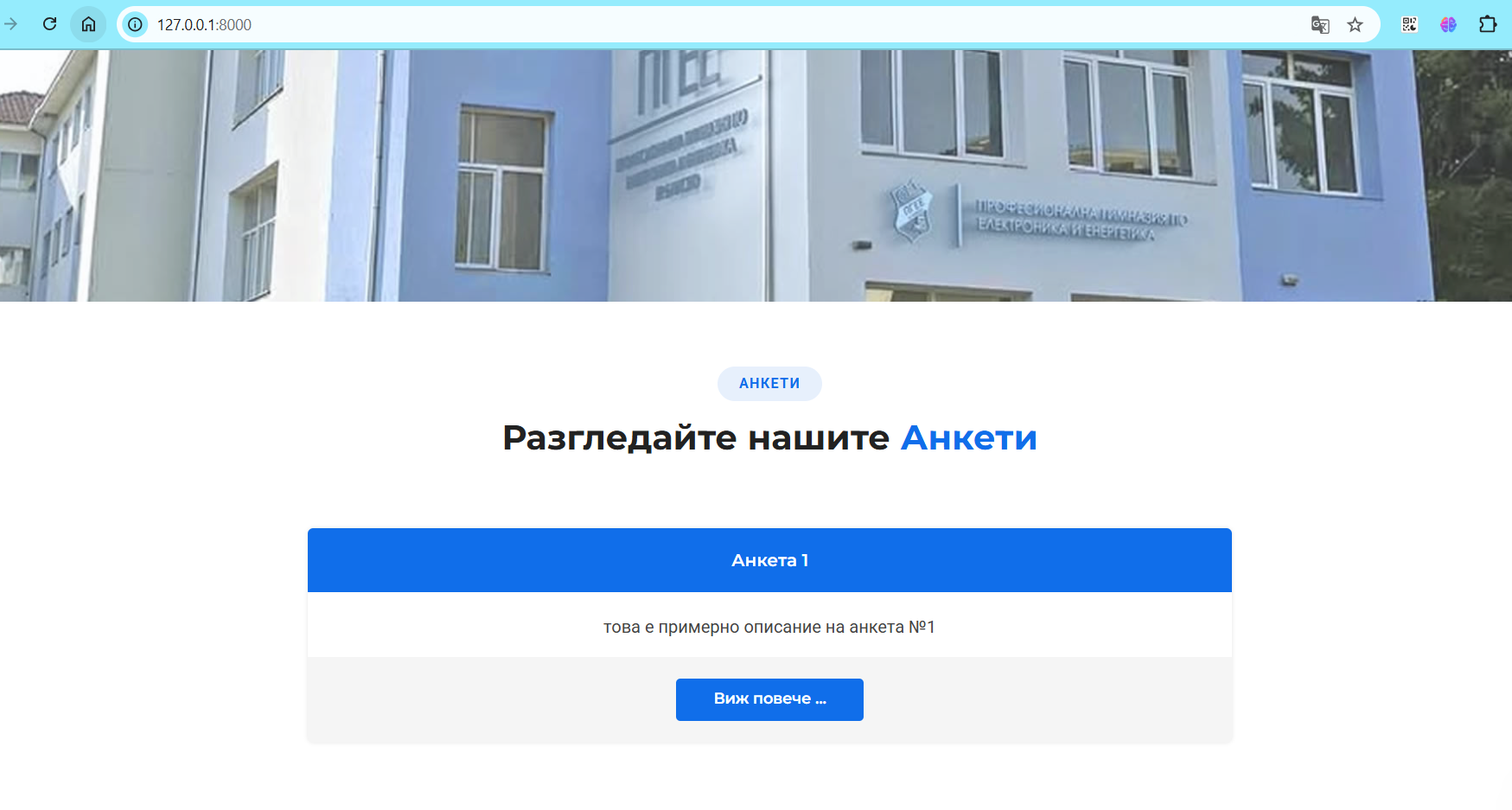
Свързването с база данни обикновено включва използване на драйвери или библиотеки, които позволяват на приложението да комуникира с базата данни. В контекста на Django, например, се използва ORM (Object-Relational Mapping), което позволява манипулиране на база данни чрез Python класове и обекти.





# Практическа част

## Обяснение на моделите в Django за анкети



### Въведение

Кодът, дефинира три модела в база данни, използвайки Django – един от най-популярните уеб фреймворци на Python. Тези модели представляват логическото описание на данните за една анкета: самата анкета (Poll), въпросите в нея (Question) и отговорите (Option).

Моделите са класове, които се свързват с таблици в база данни и определят какви полета съдържат и какви взаимоотношения има между тях.

* **Модел Poll** (Анкета)

class Poll(models.Model):

theme = models.CharField('Тема', max\_length=100, default='', help\_text='Тема на анкетата')

description = models.TextField('Описание', default='', blank=True, help\_text='Описание на анкетата')

picture = models.ImageField('Картинка', upload\_to='poll\_pics', blank=True)

active = models.BooleanField('Активна', default=True, help\_text='Дали анкетата е видима за потребителите')

**Poll** е модел, който съхранява информация за една анкета. Всеки обект от класа Poll представлява отделна анкета в базата данни.

Полета:

* **Theme (тема):** Текстово поле, за името или заглавието на анкетата (до 100 символа).
* **Description (описание):** По-дълъг текст, описващ анкетата. Може да е празен.
* **Picture (картинка):** Снимка, свързана с анкетата. Ако няма, полето може да бъде празно (blank=True).
* **Active (активна):** Булево поле, което указва дали анкетата е активна и видима за потребителите.

Специални методи и настройки:

* \_\_stir\_\_(self): връща текстово представяне на обекта — темата на анкетата.
* Вътрешният клас Meta дава имена за модела на български (за администрацията и интерфейсите).
* **Модел Question (Въпрос)**

Class Question (models. Model):

text = models.TextField('Текст', default='', blank=True, help text='Текст на въпроса')

type = models.PositiveSmallIntegerField(choices=QUESTION\_TYPE, default=TYPE1, help\_text='тип на въпроса')

pool\_id = models.ManyToManyField(Poll, verbose\_name='id на анкета, в която се ползва', blank=True)

stat\_attempts = models.IntegerField('Брой отговори', default=0, help\_text='генерира се автоматично').

Всеки обект от Question представлява един въпрос в една или повече анкети. Въпросите могат да имат различни типове – с един отговор или с множество отговори.

Полета:

* text (текст): Текстът на въпроса.
* type (тип): Цяло число, което определя вида на въпроса. Възможни стойности:
  + 1 – въпрос с един отговор (radio button).
  + 2 – въпрос с множествен избор (checkbox).
* pool\_id: Връзка "много към много" (ManyToManyField) с анкетите (Poll), което означава, че един въпрос може да бъде включен в повече от една анкета и една анкета може да съдържа повече от един въпрос.
* stat\_attempts: Число, което показва колко пъти въпросът е отговарян (записва се статистика).

Методи**:**

update\_statistics(self): Метод, който увеличава броя на отговорите и записва промените в базата.

\_\_str\_\_(self): Връща текстовото съдържание на въпроса за удобство.

Meta: отново имена на български.

* **Модел Option** (Опции за отговори)

class Option(models.Model):

question\_id = models.ForeignKey(Question, on\_delete=models.CASCADE, null=True, related\_name='options')

text = models.CharField('Текст', max\_length=200, default='', blank=True, help\_text='Формулировка (текст) на опцията(отговора)')

checked = models.BooleanField('Отговор', null=True, help\_text='генерира се автоматично')

stat\_attempts = models.IntegerField('Брой отговори', default=0, help\_text='генерира се автоматично')

Option е модел, който съдържа възможните отговори (опции) за даден въпрос. Връзката с въпроса е тип „едно към много“ (ForeignKey) – един въпрос може да има много опции.

Полета:

* **question\_id:** Връзка към даден въпрос (Question). При изтриване на въпроса, опцията също се изтрива (on\_delete=models.CASCADE).
* **text:** Текстът на конкретната опция.
* **checked:** Булево поле, което указва дали конкретният отговор е избран – в кода е с null=True и се създава автоматично после (например след гласуване).
* **stat\_attempts:** Статистика за това колко пъти този отговор е избран.

Методи:

* update\_statistics(self): Увеличава стойността на статистиката при избор на отговора.
* \_\_str\_\_(self): Връща текстово описание („#ID: Текст“) на опцията за лесно идентифициране.

Meta: имена на български.

### Взаимоотношения между моделите

Една анкета (Poll) съдържа много въпроси - моделът Question има ManyToManyField към Poll.

Един въпрос (Question) има много опции - чрез ForeignKey от Option.

* **Статистики**

За всеки въпрос и всяка опция се пази статистика за броя опити (изпълнения, гласове). Това позволява в последствие да се анализира кои въпроси са били отговаряни по-често и кои опции са предпочитани.

Методите update\_statistics са създадени за обновяване на тези полета, като увеличават брояча с 1 всеки път, когато се регистрира нов отговор.

### Защо ползваме Django модели?

* Django моделите абстрахират работата с базата данни. Вместо директни SQL заявки, описваш данните като Python класове и полета.
* Моделите позволяват лесно създаване, четене, обновяване и изтриване на данни.
* Връзките между данните се описват ясно и лесно за ползване в код.

## Обобщение

Тези модели задават основната структура на една система за анкети, която ще позволи на потребителите да:

Създават анкети с име, описание и снимка.

* Формират въпроси към анкетите с избор между типове отговори (единствен или множествен избор).
* Добавят възможни отговори към всеки въпрос.
* Следят статистиката колко пъти всяка анкета, въпрос и опция са били използвани или избирани.

# Заключение

В рамките на настоящия проект беше разгледана концепцията за работа с бази данни в уеб фреймуорка Django, акцентирайки на важността на правилното управление на данни в съвременната информационна среда. Чрез систематичен анализ на релационните и нерелационните бази данни, както и на различните инструменти и механизми, предоставяни от Django, проектът демонстрира, че съвременните приложения изискват надеждни и ефективни решения за осъществяване на взаимодействие с данните.

Django, с вградената си обектно-релационна система, предоставя мощни средства, позволяващи на разработчиците бързо и безопасно да съхраняват и обработват информация. Основният акцент върху ORM (Object-Relational Mapping) показва как фреймуоркът оптимизира начина, по който потребителите взаимодействат с базата данни, намалявайки необходимостта от директно писане на SQL код и премахвайки риска от грешки.

Темата на проекта е особено значима в контекста на бързо променящите се технологии и нуждата от адаптиране към новите изисквания на пазара. Правилното управление на данни не само че подпомага оперативната ефективност, но е и ключово за стратегическото развитие на бизнеса. Във времена, когато информационната сигурност и оптимизацията на ресурсите са от най-голямо значение, използването на стабилни фреймуорк като Django представлява разумен избор за всяка съвременна разработка.

Заключителните изводи от проекта показват, че усвояването на принципите и инструментите на Django не само че е полезно за начинаещи и напреднали разработчици, но и изключително важно за бъдещето на уеб приложенията. На база на направените анализи и представени примери, можем да потвърдим, че Django предлага на разработчиците необходимата стабилност и гъвкавост, за да реализират иновации, отговарящи на съвременните бизнес изисквания.

В заключение, проектът подчертава значимостта на добрата основа в работата с бази данни и преимуществата, които един мощен инструмент като Django може да предостави на разработчиците, стремящи се към създаването на устойчиви и ефективни уеб решения.

# Литература:

1. Колисниченко, Денис, Адаптивен уеб дизайн с Bootstrap, Асеневци, С 2019
2. D.K Academy, Python – практическо програмиране. Асеневци, С 2022
3. Донълдсън, Тоби. Бързо ръководство:Програмиране с Python.АлексСофт, С 2017
4. Есканази, Аврам. Софтуерни техологии. КЛМН, С 2006
5. https://softuni.bg/blog/what-is-django
6. 2. https://cynoteck.com/bg/blog-post/flask-vs-django/
7. 3.https://bg.savtec.org/articles/coding/the-basics-of-rest-and-restful-api-development.html
8. 4. https://urocibg.eu/
9. 5. https://www.django-rest-framework.org/
10. 6. https://softuni.bg/blog/vue-js-january-2021
11. 7. https://bg.wikipedia.org/wiki/MySQL
12. 8. https://priobshti.se/article/strategii-v-pomosht-na-prepodavaneto/taksonomiya-na-blum-ili-kak-da-napravim-uroka-v
13. 9. https://html.w3schools.bg
14. 10. https://www.inventum.bg/web-design/css/