

АБДРАЗАКОВА ГУЛМИРА АБДЫЖАПАРОВНА
ИСМАНОВА ТЫНАРА КУБАНЫЧБЕКОВНА

Бэкенд иштеп чыгуучу (МББС, структуралык программалоо)

ОКУУ КУРАЛЫ



Манас 2025

**КЫРГЫЗ РЕСПУБЛИКАСЫНЫН ИЛИМ, ЖОГОРКУ БИЛИМ БЕРҮҮ ЖАНА
ИННОВАЦИЯЛАР МИНИСТРЛИГИ
“Б.ОСМОНОВ АТЫНДАГЫ ЖАЛАЛ-АБАД МАМЛЕКЕТТИК УНИВЕРСИТЕТИ”
ИЛИМИЙ БИЛИМ БЕРҮҮ ӨНДҮРҮШТҮК КОМПЛЕКСИ**

ЖАЛАЛ-АБАД КОЛЛЕДЖИ

АВТОМАТТАШТЫРЫЛГАН СИСТЕМАЛАР ЖАНА МАТЕМАТИКА ПЦКсы

**АБДРАЗАКОВА ГУЛМИРА АБДЫЖАПАРОВНА
ИСМАНОВА ТЫНАРА КУБАНЫЧБЕКОВНА**

**Бэкенд иштеп чыгуучу (МББС, структуралык программалоо)
дисциплинасы боюнча**

ОКУУ КУРАЛЫ

**Шифр 230109 “Автоматташтырылган системаларды жана эсептөө техникасын
программалык камсыздоо” адистиги үчүн**

Манас 2025

УДК 004.4 + 004.65

ББК 32.973.26

**А 15 Басууга Б.Осмонов атындагы Жалал-Абад мамлекеттик университетинин
методикалык кеңеши сунуш кылган.**

Рецензенттер: Лек окуу борборунун жетекчиси Эркулова Э.К.

ЖАКтын АСЖМ ПЦКсынын окутуучусу Мусаева Э.

Методикалык кеңештин мүчөсү Бектурсунов Ж.Б.

Абдразакова Г.А. А15. Бэкенд иштеп чыгуучу (маалымат базасын башкаруу системалары жана структуралык программалоо) багыты боюнча адистик алган студенттери менен мектепте биология, химия, экология мугалимдери үчүн окуу куралы.

Б. Осмонов атындагы Жалал-Абад мамлекеттик университети 67 бет.

ISBN _____

Окуу куралында бэкенд иштеп чыгуунун негизги түшүнүктөрү кеңири баяндалат. Маалымат базасын башкаруу системаларынын (МББС) теориялык негиздери, структуралык программалоо принциптери жана Python программалоо тили аркылуу маалыматтар менен иштөө ыкмалары заманбап билим берүү талаптарына ылайык жазылды.

Окуу куралда алгоритм түзүү, структуралык программалоонун негизги конструкциялары, SQLite жана PostgreSQL маалымат базалары менен иштөө, SQL суроолорун жазуу, Python аркылуу МББСке туташуу, ошондой эле практикалык мисалдар жана лабораториялык иштер камтылган.

Бул окуу куралы жогорку жана орто кесиптик окуу жайларынын студенттерине, бэкенд иштеп чыгууну өздөштүрүп жаткан окуучуларга жана информатика, программалоо багытындагы мугалимдерге арналат.

Т 4306011100-06

ISBN _____

УДК 004.4:004.6

ББК 32.973.26

© Б.Осмонов атындагы Жалал-Абад мамлекеттик университети.

© Абдразакова Г. А., 2025

МАЗМУНУ

Киришүү.....	6
I БӨЛҮМ: Python тилине кыскача сереп	7
1.1 Python синтаксиси	8
II БӨЛҮМ: SQLite менен иштеп чыгуу	12
2.1 SQLite'ке киришүү	12
2.2 SQLite таблицасынын структурасы жана негизги операциялар	18
III БӨЛҮМ. Django веб-фреймворку менен иштеп чыгуу	22
3.1 Django фреймворкуна киришүү	22
3.2 Виртуалдык чөйрө (venv) жана пакеттерди башкаруу (pip)	25
3.3 Проект түзүү жана структурасы	28
3.4 Алгачкы view жазуу	33
3.5 Модельдер жана маалымат базасы менен байланыш (Models)	37
3.6 Формалар жана шаблондор менен иштөө (Forms & Templates).....	41
3.7 Формалар жана шаблондор менен иштөө (Forms & Templates).....	46
3.8 Админ панель жана Django'дун автоматтык интерфейси	46
IV БӨЛҮМ: ПРАКТИКАЛЫК ДОЛБООРОЛОР.....	49
4.1 Django менен блог тиркемесин түзүү.....	49
4.2 Django + SQLite менен Student Management системасы	53
4.3 Django'ну серверге жайгаштыруу (деплой).....	60
1. Теориялык негиздер.....	60
1. Деплой деген эмне?.....	60
2. Сервер түрлөрү.....	60
3. Деплойлоодо керектүү компоненттер	60
4. Деплойлоо кадамдары (Ubuntu сервер мисалы).....	61
4.1 Серверге туташуу	61
4.2 Python жана virtual environment орнотуу	61
4.3 Django долбоорун серверге көчүрүү	61
4.4 Пакеттерди орнотуу	61
4.5 Gunicorn орнотуу жана тестирлөө	61
4.6 Статикалык файлдарды башкаруу.....	61
4.7 Nginx менен Gunicorn конфигурациялоо	61
4.8 Серверди автоматтык баштоо	62
5. Heroku аркылуу жеңил деплойлоо (студенттер үчүн ыңгайлуу).....	62
6. Практикалык кеңештер.....	63

7. Практикалык тапшырмалар	63
8. Жыйынтык	63
V БӨЛҮМ. ЖЫЙЫНТЫК ЖАНА СУНУШТАР	64
5.1 Жыйынтыктык корутунду	64
5.2 Карьералык багыттар жана практикалык сунуштар	65
5.3 Кошумча билим берүү ресурстары жана сунуш кылынган булактар	66
Адабияттар	68
Адабияттар	84

Киришүү

Азыркы санариптик өнүгүү шартында программалоо жана веб-технологиялар маалыматтарды иштеп чыгуу, сактоо жана башкаруу процесстеринин негизги компоненти болуп калды. Ар түрдүү тармактарда автоматташтырылган системаларга болгон талаптын өсүшү программалоо тилдерин жана веб-платформаларды өздөштүрүүнүн маанисин жогорулатууда. Бул багытта Python программалоо тили, SQLite маалымат базасы жана Django веб-фреймворку заманбап программалык камсыздоону иштеп чыгууда кеңири колдонулган негизги технологиялар катары өзгөчө орунду ээлейт.

Python программалоо тили дүйнө жүзүндө жогорку популярдуулукка ээ болгон универсалдуу тилдердин катарына кирет. Анын синтаксистик жөнөкөйлүгү, логикалык так түзүлүшү жана кеңири колдонуу чөйрөсү программалоону жаңы үйрөнүп жаткан студенттер үчүн да, практикалык тажрыйбасы бар адистер үчүн да натыйжалуу пайдаланууга шарт түзөт. Python автоматташтыруу, маалыматтарды талдоо, веб-тиркемелерди иштеп чыгуу, жасалма интеллект жана башка көптөгөн прикладдык тармактарда ийгиликтүү колдонулуп келет.

SQLite маалымат базасы жеңил жана компакттуу маалымат сактоо каражаты болуп саналат. Анын негизги өзгөчөлүгү — өзүнчө сервердик инфраструктураны талап кылбагандыгында. SQLite толук кандуу SQL функционалын колдойт жана Python программалоо тили менен оңой интеграцияланат. Ушул себептен улам, аталган маалымат базасы студенттерге маалымат базаларынын негизги принциптерин түшүндүрүүдө жана практикалык көндүмдөрдү калыптандырууда ыңгайлуу чечим катары колдонулат.

Django — Python тилинин негизинде иштелип чыккан жогорку деңгээлдеги веб-фреймворк. Ал веб-тиркемелерди тез, коопсуз жана структураланган түрдө иштеп чыгууга мүмкүнчүлүк берет. Django фреймворку MVT архитектуралык моделине таянат жана объект-ориентирленген маалыматтар менен иштөөгө арналган ORM, автоматташтырылган админ-панель, колдонуучуларды башкаруу жана аутентификация механизмдери сыяктуу маанилүү инструменттерди камтыйт. Бул мүмкүнчүлүктөр студенттерге веб-иштеп чыгуунун толук циклын өздөштүрүүгө шарт түзөт.

Бул окуу куралы Python программалоо тилинин негизги түшүнүктөрүнөн баштап, SQLite маалымат базасы менен иштөө ыкмаларын жана Django фреймворкун колдонуу аркылуу толук функционалдуу веб-тиркемелерди түзүү процесстерин камтыйт. Теориялык материалдар практикалык мисалдар, тапшырмалар жана чакан долбоорлор менен бекемделген. Окуу куралынын негизги максаты — студенттердин практикалык багыттагы билимдерин өнүктүрүү жана аларды реалдуу программалык долбоорлорду ишке ашырууга даярдоо.

Бул окуу куралы төмөнкү категориядагы окурмандарга арналат:

- программалоо тармагын жаңыдан өздөштүрүп баштаган студенттерге;
- Python программалоо тили жана веб-иштеп чыгуу боюнча баштапкы билимге ээ болгон окурмандарга;
- SQLite маалымат базасы жана Django фреймворку менен иштөөнү үйрөнүүгө кызыккандарга;
- практикалык долбоорлорду иштеп чыгуу көндүмдөрүн калыптандырууну көздөгөндөргө.

Окуу материалын толук өздөштүргөндөн кийин окурман төмөнкү компетенцияларга ээ болот:

- Python программалоо тилинин негизги синтаксистик конструкцияларын колдонуу;
- SQLite маалымат базасында таблицаларды түзүү, маалыматтарды кошуу, өзгөртүү жана башкаруу;
- Django фреймворкунун негизги компоненттерин колдонуп веб-тиркеме иштеп чыгуу;
- моделдер, шаблондор, көргөзмө логикасы жана URL конфигурациялары менен иштөө;
- чакан практикалык долбоорлорду иштеп чыгып, жеке портфолиону толуктоо.

I БӨЛҮМ: Python тилине кыскача сереп

Бул бөлүмдө Python программалоо тилинин жалпы мүнөздөмөсү, негизги өзгөчөлүктөрү жана колдонуу чөйрөлөрү каралат. Python — учурда маалыматтык технологиялар тармагында кеңири таралган жана билим берүү процессинде активдүү колдонулган универсалдуу программалоо тили болуп саналат.

Python программалоо тили өзүнүн жөнөкөй жана логикалык жактан так синтаксиси менен айырмаланат. Коддун түзүлүшү англис тилине жакын болгондуктан, ал программалоону жаңы үйрөнүп жаткан студенттер үчүн да оңой өздөштүрүлөт. Ошол эле учурда, Python татаал программалык системаларды иштеп чыгууда да жогорку натыйжалуулукту камсыздайт.

Python тили көп максаттуу колдонууга ээ. Ал веб-тиркемелерди иштеп чыгууда, маалыматтарды талдоо жана визуалдаштырууда, ар кандай процесстерди автоматташтырууда, жасалма интеллект жана машиналык үйрөнүү тармактарында, ошондой эле билим берүүчү жана прикладдык программаларды түзүүдө кеңири колдонулат.

Тилдин маанилүү артыкчылыктарынын бири — анын кеңири стандарттык китепканасы. Бул китепкана программалоодо көп кездешкен маселелерди кошумча программалык каражаттарды орнотпостон чечүүгө мүмкүнчүлүк берет. Мындан тышкары, Python көп платформада иштөө мүмкүнчүлүгүн камсыз кылат жана Windows, Linux, macOS операциялык системаларында колдоого алынат.

Python программалоо тилинин өнүгүшүнө эл аралык программисттер коомчулугу чоң салым кошуп келет. Ачык булактуу мүнөзү жана жеткиликтүү окуу-методикалык материалдардын көптүгү бул тилди окутууда жана практикалык колдонууда натыйжалуу пайдаланууга шарт түзөт.

1.1 Python синтаксиси

1. Өзгөрмөлөр жана маалымат типтери

Python’до өзгөрмө (variable) түзүү үчүн тип көрсөтүүнүн кереги жок:

```
name = "Ines"
age = 20
height = 1.68
is_student = True
```

Негизги маалымат типтери:

Тип	Мисал	Түшүндүрмө
int	5, 100	Бүтүн сан
float	5.5, 3.14	Ондук сан
str	"Hello"	Текст
bool	True/False	Логикалык маанилер
list	[1, 2, 3]	Тизмектер
dict	{"name": "Ines"}	Ачкыч-маани жуптары

2. Операторлор Арифметикалык операторлор $a + b$ $a - b$ $a * b$ a / b $a // b$ бүтүн бөлүү $a \% b$ калдыгы $a ** b$ даражаландыруу	Салыштыруулар $x == y$ $x != y$ $x > y$ $x < y$ $x >= y$ $x <= y$ Логикалык операторлор and, or, not
3. Шарттык конструкциялар (if / elif / else) <code>age = 18</code> <code>if age >= 18:</code> <code> print("Катталсаңыз болот")</code> <code>elif age == 17:</code> <code> print("Бир жыл күтүңүз")</code> <code>else:</code> <code> print("Катталбайсыз")</code> Python’до блоктор чыгыш менен (отступ) белгиленет — бул өтө маанилүү.	4. Циклдер (for, while) for цикли <code>for i in range(5):</code> <code> print(i)</code> while цикли <code>count = 0</code> <code>while count < 3:</code> <code> print("Сан:", count)</code> <code> count += 1</code>

5. Функциялар

Функциялар коду кайталоону азайтып, тартипке келтирет.

```
def greet(name):
    return f"Салам, {name}!"
```

```
print(greet("Ines"))
```

6. Модульдар менен иштөө

Башка файлдагы же китепканадан коду импорттоо:

```
import math
print(math.sqrt(25))
```

7. Файлдар менен иштөө

```
with open("test.txt", "w") as f:
```

```
f.write("Hello, world!")
```

№1 Практикалык мисалдар

Тапшырманын шарттары

Студенттин аты, жашы жана 3 сабак боюнча алган бааларын өзгөрмөлөргө сактап, төмөнкү иштерди аткаргыла:

1. Өзгөрмөлөрдү колдонуп маалыматтарды сактагыла
2. Орточо бааны эсептегиле
3. Орточо баага жараша жыйынтык чыгаргыла:
 - 90 жана андан жогору → «**Эң жакшы**»
 - 70–89 → «**Жакшы**»
 - 50–69 → «**Канаттандыраарлык**»
 - 49дан төмөн → «**Канаттандыраарлык эмес**»

Үлгү аткаруу коду

```
name = "Ines"
```

```
age = 20
```

```
score1 = 85
```

```
score2 = 90
```

```
score3 = 80
```

```
average = (score1 + score2 + score3) / 3
```

```
print("Аты:", name)
```

```
print("Жашы:", age)
```

```
print("Орточо баа:", average)
```

```
if average >= 90:
```

```
    print("Жыйынтык: Эң жакшы ")
```

```
elif average >= 70:
```

```
    print("Жыйынтык: Жакшы")
```

```
elif average >= 50:
```

```
    print("Жыйынтык: Канаттандыраарлык")
```

```
else:
```

```
    print("Жыйынтык: Канаттандыраарлык эмес")
```

Студент өздөштүргөн түшүнүктөр

- Өзгөрмөлөр жана маалымат типтери
- Арифметикалык операторлор
- Шарттык операторлор (if / elif / else)
- print() функциясы

№2 Практикалык тапшырма

Тапшырманын шарты:

Студент өзүнүн аты-жөнүн, жашын жана баасын (0–100) өзгөрмөлөргө сактап, төмөнкү талаптарды аткарсын:

1. Атын жана жашын экранга чыгарсын
2. Баага жараша жыйынтык чыгарсын:
 - 90–100 → «A»
 - 70–89 → «B»
 - 50–69 → «C»
 - 0–49 → «F»
3. for цикли менен 1ден 5ке чейинки сандарды чыгарсын

Аткарууга үлгү код:

```
name = "Ines"
```

```
age = 20
```

```
score = 85
```

```
print("Аты:", name)
```

```
print("Жашы:", age)
```

```
if score >= 90:
```

```
    print("Баасы: A")
```

```
elif score >= 70:
```

```
    print("Баасы: B")
```

```
elif score >= 50:
```

```
    print("Баасы: C")
```

```
else:
```

```
    print("Баасы: F")
```

```
for i in range(1, 6):
```

```
    print(i)
```

ТЕСТ



<http://qrcoder.ru/code/?https%3A%2F%2Fforms.gle%2Fzmtfo4pF4bGa9k5h8&4&0>

II БӨЛҮМ: SQLite менен иштеп чыгуу

SQLite — бул жеңил, сервер талап кылбаган, файлдык негизде иштеген маалымат базасы. Ал кичи жана орто деңгээлдеги долбоорлор үчүн эң ыңгайлуу жана Python ичинде демейки модул катары келет. SQLite колдонууда ар бир база — жөн гана бир *.db* файлы, бул аны өтө мобилдүү жана жөнөкөй кылат.

Бул бөлүмдө SQLite базасын түзүп, таблицалар кошуп, маалымат кошуу, өчүрүү, өзгөртүү жана тандоо операцияларын үйрөнөбүз.

2.1 SQLite'ке киришүү

SQLite деген эмне?

SQLite — бул:

- **Серверсиз** маалымат базасы (MySQL, PostgreSQL сыяктуу сервер талап кылбайт)
- **Бир файлда сакталат** (мисалы: database.db)
- **Тез жана жеңил**
- **Python менен автоматтык интеграцияланган (sqlite3)**
- **SQL стандартын жакшы колдогон база**

Python менен окуу процесстерине, мини-проекттерге, десктоп жана мобилдик тиркемелерге кеңири колдонулат.

SQLite файлын түзүү

Python ичинде база түзүү үчүн төмөнкүдөй кат жазылат:

```
import sqlite3
```

```
conn = sqlite3.connect("mydatabase.db")
```

Эгер mydatabase.db жок болсо — Python автоматтык түрдө аны түзөт.

Таблица түзүү (CREATE TABLE)

```
import sqlite3
```

```
conn = sqlite3.connect("mydatabase.db")
cursor = conn.cursor()
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    age INTEGER
)
""")
```

```
conn.commit()
conn.close()
```

Маалымат кошуу (INSERT)

```
conn = sqlite3.connect("mydatabase.db")
cursor = conn.cursor()
```

```
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Ines", 20))
```

```
conn.commit()
conn.close()
```

? — бул SQL injection болбошу үчүн колдонулган коопсуз параметр.

Маалымат алуу (SELECT)

```
conn = sqlite3.connect("mydatabase.db")
cursor = conn.cursor()
```

```
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()
```

```
for row in rows:
    print(row)
```

```
conn.close()
```

Маалымат жаңылоо (UPDATE)

```
cursor.execute("UPDATE students SET age = ? WHERE id = ?", (21, 1))
```

Маалымат өчүрүү (DELETE)

```
cursor.execute("DELETE FROM students WHERE id = ?", (1,))
```

Негизги SQL командалар

Команда	Максаты
CREATE TABLE	Таблица түзүү
INSERT INTO	Маалымат кошуу
SELECT	Маалымат алуу
UPDATE	Маалыматты өзгөртүү
DELETE	Маалыматты өчүрүү
ALTER TABLE	Таблицаны өзгөртүү
DROP TABLE	Таблицаны толук жок кылуу

Практикалык мисал

Максат: Студенттер таблицасын түзүп, маалымат кошуп, алып, жаңылап, өчүрүү.

```
import sqlite3
```

1. Базага туташуу

```
conn = sqlite3.connect("college.db")
```

```
cursor = conn.cursor()
```

2. Таблица түзүү

```
cursor.execute("""
```

```
CREATE TABLE IF NOT EXISTS students (
```

```
    id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
    name TEXT,
```

```
    age INTEGER,
```

```
    major TEXT
```

```
)
```

```
""")
```

3. Маалымат кошуу

```
cursor.execute(
```

```
    "INSERT INTO students (name, age, major) VALUES (?, ?, ?)",
```

```
    ("Ines", 20, "IT")
```

```
)
```

4. Маалымат алуу

```
cursor.execute("SELECT * FROM students")
for row in cursor.fetchall():
    print(row)
```

5. Маалымат жаңылоо

```
cursor.execute(
    "UPDATE students SET age = ? WHERE name = ?",
    (21, "Ines")
)
```

6. Маалымат өчүрүү

```
cursor.execute(
    "DELETE FROM students WHERE name = ?",
    ("Ines",)
)
```

```
conn.commit()
```

```
conn.close()
```

Практикалык тапшырмалар

Тапшырма 1

school.db деген база түзүп, teachers таблицасын түзгүлө:

- id
- fullname
- subject

Тапшырма 2

teachers таблицасына 3 мугалимдин маалыматын кош.

Тапшырма 3

Бардык мугалимдердин тизмесин SELECT менен чыгар.

Тапшырма 4

Бир мугалимдин предметин UPDATE аркылуу өзгөрт.

Тапшырма 5

Бир мугалимди DELETE аркылуу өчүр.

Интерактивдүү тапшырма 1 — Кодду толукта

Берилген коддо бош жерлерди толтур:

```

import sqlite3

conn = sqlite3._____("school.db")
cursor = conn.______()

cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    age INTEGER
)
""")
conn.______()
conn.______()

```

Текшерүү (жообу):

```

connect
cursor
commit
close

```

Интерактивдүү тапшырма 2 — Туура же туура эмес

№	Билдирүү	Жооп
1	SQLite сервер талап кылат	
2	SQLite маалыматты файлда сактайт	
3	INSERT маалымат өчүрөт	
4	SELECT маалыматты алат	
5	DROP TABLE таблицаны толук өчүрөт	

Интерактивдүү тапшырма 3 — Натыйжаны тап

Код:

```

cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Aida", 19))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Nursultan", 20))
cursor.execute("SELECT * FROM students")
print(cursor.fetchall())

```

Суроо: Канча жазуу чыгат?

Интерактивдүү тапшырма 4 — Команданы танда

Суроо: 20 жаштагы студенттерди алуу үчүн кайсы команда туура?

- A) SELECT * FROM students;
- B) SELECT * FROM students WHERE age = 20;
- C) ``sql
DELETE FROM students WHERE age = 20;

Интерактивдүү тапшырма 5 — Катаны тап

Коддо ката бар:

```
cursor.execute("SELECT name age FROM students")
```

Кайсы ката?

Интерактивдүү тапшырма 6 — Өзүң аткар

- 1 college.db база түз
- 2 groups таблицасын түз (id, group_name)
- 3 2 группа кош
- 4 Бардык группаларды чыгар

Өзүн-өзү текшерүү суроосу:

Тест



<http://qrcoder.ru/code/?https%3A%2F%2Fforms.gle%2Fy6cmcP8tLxRBMNnp6&4&0>

2.2. SQLite таблицасынын структурасы жана негизги операциялар

SQLite маалымат базасы таблицалардан турат. Таблица — бул саптар жана тилкелерден (row, column) турган структура. Ар бир таблицада маалыматтын түрү, аталышы жана чектөөлөрү болот.

Бул бөлүмдө биз таблицаны туура түзүү, өзгөртүү жана анын негизги операцияларын аткарууну үйрөнөбүз.

1. Таблица структурасы деген эмне?

Таблицанын структурасы төмөнкүлөрдү камтыйт:

- Тилке аталышы (column name)
- Маалымат тиби (type)
- Чектөөлөр (constraints)
- Башкы ачкыч (PRIMARY KEY)
- Автоматтык сан (AUTOINCREMENT)

Мисалы, студенттер таблицасы:

Тилке	Тип	Түшүндүрмө
id	INTEGER	Башкы ачкыч, өсүп бараткан
name	TEXT	Студенттин аты
age	INTEGER	Жашы

2. Негизги маалымат типтери (SQLite)

SQLite төмөнкү жалпы типтерди колдойт:

Тип	Мисал	Түшүндүрмө
INTEGER	5, 100	Бүтүн сан
REAL	3.14	Ондук сан
TEXT	"Hello"	Текст
BLOB	Файл, сүрөт	Бинардык маалымат
NULL	None	Маани жок

3. Чектөөлөр (Constraints)

Чектөөлөр таблицаны туура жана коопсуз кылат.

Constraint	Түшүндүрмө
PRIMARY KEY	Башкы идентификатор
AUTOINCREMENT	Автоматтык сан кошуу

Constraint	Түшүндүрмө
NOT NULL	Бош калтырууга болбойт
UNIQUE	Кайталанбас маанилер
DEFAULT	Демейки маани

4. Таблица түзүү (CREATE TABLE)

Мисалы, студенттер таблицасы:

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER DEFAULT 18
)
""")
```

5. Маалымат кошуу (INSERT)

```
cursor.execute(
    "INSERT INTO students (name, age) VALUES (?, ?)",
    ("Aizada", 19)
)
```

Бир нече студентти бир эле убакта кошуу:

```
students = [
    ("Maksat", 20),
    ("Ines", 18),
    ("Adil", 21)
]
cursor.executemany("INSERT INTO students (name, age) VALUES (?, ?)", students)
```

6. Маалыматты тандоо (SELECT)

Бардык жазуулар:

```
cursor.execute("SELECT * FROM students")
print(cursor.fetchall())
```

Тек айрым тилкелерди тандоо:

```
cursor.execute("SELECT name, age FROM students")
```

Шарт менен тандоо:

```
cursor.execute("SELECT * FROM students WHERE age > 18")
```

Бир сап алуу:

```
cursor.execute("SELECT * FROM students WHERE id = ?", (1,))  
print(cursor.fetchone())
```

7. Маалыматты өзгөртүү (UPDATE)

```
cursor.execute(  
    "UPDATE students SET age = ? WHERE name = ?",  
    (22, "Ines")  
)
```

8. Маалыматты өчүрүү (DELETE)

```
cursor.execute(  
    "DELETE FROM students WHERE id = ?",  
    (3,)  
)
```

Бардык таблицаны тазалоо:

```
cursor.execute("DELETE FROM students")
```

9. Таблицанын структурасын өзгөртүү (ALTER TABLE)

Жаңы тилке кошуу:

```
cursor.execute("ALTER TABLE students ADD COLUMN grade TEXT")
```

Тилкени өчүрүү SQLite'те түз колдоого алынбайт

Бирок аны төмөнкү жол менен жасоого болот:

1. Жаңы таблица түзүү
2. Эски таблицадан маалыматты көчүрүү
3. Эски таблицаны өчүрүү
4. Жаңы таблицаны эски атка өзгөртүү

10. Толугу менен таблицаны өчүрүү (DROP TABLE)

```
cursor.execute("DROP TABLE IF EXISTS students")
```

Практикалык мисал

Мисал: *Students* таблицасын түзүп, 1 студент кошуп, көрсөтүү

```
import sqlite3
```

Базага туташуу

```
conn = sqlite3.connect("example.db")
```

```
cursor = conn.cursor()
```

Таблица түзүү

```

cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER
)
""")

```

1 студент кошуу

```

cursor.execute(
    "INSERT INTO students (name, age) VALUES (?, ?)",
    ("Ines", 20)
)

```

Студентти көрсөтүү

```

cursor.execute("SELECT * FROM students")
print(cursor.fetchall())

```

Сактоо жана жабуу

```

conn.commit()
conn.close()

```

Натыйжа (чыгышы):

(1, 'Ines', 20)

Бул мисалда:

- таблица түзүлдү
- 1 маалымат кошулду
- SELECT менен көрсөтүлдү

Практикалык тапшырмалар

Тапшырма 1

example.db база менен иште.

1 students таблицасын түз (id, name, age)

2 1 студент кош: **Ines, 20**

3 Бардык студенттерди чыгар

Тапшырма 2

Ошол эле таблицада:

- 1 Дагы **2 студент** кош
- 2 Жашы **18ден жогору** студенттерди чыгар
- 3 Бир студенттин жашын өзгөрт (UPDATE)

Тапшырма 3

students таблицасы боюнча:

- 1 Бир студентти id аркылуу өчүр
- 2 Калган студенттерди экранга чыгар

Күтүлгөн жыйынтык:

→ Таблицада кеминде **1 студент** калышы керек.

Тест



<http://qrcoder.ru/code/?https%3A%2F%2Fforms.gle%2FMd222QQjNXD8FC9G8&4&0>

III БӨЛҮМ. Django веб-фреймворку менен иштеп чыгуу

Бул бөлүмдө Django веб-фреймворкун колдонуу аркылуу веб-тиркемелерди иштеп чыгуунун негизги принциптери каралат. Django — Python программалоо тилинин негизинде иштелип чыккан, жогорку деңгээлде структураланган жана кеңири функционалдык мүмкүнчүлүктөргө ээ веб-фреймворк болуп саналат. Ал иштеп чыгуучуга кыска убакыт ичинде коопсуз жана ишенимдүү веб-тиркемелерди түзүүгө шарт түзөт.

Django фреймворкунун өзгөчөлүгү — көптөгөн керектүү компоненттердин алдын ала даяр болушу. Ушул себептен ал “batteries included” принциби менен иштеген фреймворк катары белгилүү. Бул принцип веб-долбоор түзүүдө негизги функцияларды нөлдөн баштап жазуунун зарылдыгын жоёт.

Бул бөлүмдө Django фреймворкуна киришүү, анын түзүмү, архитектуралык модели жана колдонуу чөйрөлөрү жөнөкөй жана түшүнүктүү мисалдар менен баяндалат.

3.1 Django фреймворкуна киришүү

Django — веб-тиркемелерди иштеп чыгууга багытталган, Python тилинде жазылган заманбап веб-фреймворк. Ал тез иштеп чыгууга, коопсуздук талаптарын сактоого жана

коддун иреттүү болушуна өзгөчө көңүл бурат. Django дүйнө жүзү боюнча кеңири колдонулуп, чоң жана татаал веб-долбоорлорду ишке ашырууда ишенимдүү платформа катары таанылган.

Django фреймворку 2005-жылы веб-журналистика тармагында колдонуу үчүн иштелип чыгып, кийинчерээк универсалдуу веб-платформага айланган. Бүгүнкү күндө фреймворк активдүү өнүгүп, эл аралык иштеп чыгуучулар коомчулугу тарабынан колдоого алынып келет.

Django фреймворкунун негизги өзгөчөлүктөрү

1. Жогорку деңгээлдеги коопсуздук

Django веб-тиркемелерди түзүүдө негизги коопсуздук коркунучтарын эске алат. Фреймворк төмөнкү коркунучтардан коргоо механизмдерин автоматтык түрдө камсыз кылат:

- сайт аралык скрипттер (XSS);
- жасалма сурамдар (CSRF);
- SQL сурамдарды бурмалоо;
- Clickjacking сыяктуу чабуулдар.

Бул өзгөчөлүк студенттер үчүн коопсуз веб-тиркеме түзүүнүн негиздерин үйрөнүүгө мүмкүнчүлүк берет.

2. Ылдам иштеп чыгуу мүмкүнчүлүгү

Django'до веб-тиркеме иштеп чыгууда колдонулуучу көптөгөн компоненттер алдын ала даярдалган. Аларга төмөнкүлөр кирет:

- URL багыттоо системасы;
- форма түзүү механизмдери;
- автоматтык админ-панель;
- ORM аркылуу маалымат базасы менен иштөө.

Мындай мүмкүнчүлүктөр иштеп чыгуучунун убактысын үнөмдөп, долбоорду тез ишке ашырууга жардам берет.

3. Django ORM механизми

Django ORM (Object-Relational Mapping) маалымат базасы менен иштөөнү жөнөкөйлөтөт. Иштеп чыгуучу SQL суроолорду түздөн-түз жазбастан, Python класстары аркылуу маалымат базасы менен иш алып барат.

Мисалы, жаңы жазууну базага кошуу төмөнкүдөй аткарылат:

Book.objects.create(title="Django Basics", author="John Doe")

4. Автоматташтырылган админ-панель

Django'нун эң маанилүү артыкчылыктарынын бири — админ-панелдин автоматтык түрдө түзүлүшү. Бул панель аркылуу администраторлор маалыматтарды кошуу, өзгөртүү жана өчүрүү операцияларын кошумча код жазбастан эле аткара алышат.

5. Масштабдуулук жана туруктуулук

Django ири көлөмдөгү колдонуучуларга кызмат көрсөткөн веб-тиркемелерди иштеп чыгууга ылайыктуу. Көптөгөн ири платформалар бул фреймворкту тандап келишет, анын ичинде:

- социалдык тармактар;
- маалымат порталдары;
- электрондук соода системалары;
- корпоративдик веб-платформалар.

Бул факт Django'нун ишенимдүүлүгүн далилдейт.

Django архитектурасы (MVT модели)

Django фреймворку MVT (Model–View–Template) архитектуралык моделине негизделет. Бул модель веб-тиркеменин ички түзүмүн логикалык бөлүктөргө бөлүүгө мүмкүндүк берет.

Model — маалымат базасы менен иштөөгө жооп берет жана таблицалардын түзүмүн аныктайт.

View — колдонуучудан келген сурамды кабыл алып, аны иштетет жана жооп даярдайт.

Template — колдонуучуга көрсөтүлүүчү HTML интерфейсти камсыз кылат.

MVT модели коддун иреттүү болушуна жана долбоорду кеңейтүүгө ыңгайлуу шарт түзөт.

Django колдонулуучу негизги тармактар

Django төмөнкү типтеги веб-тиркемелерди иштеп чыгууда колдонулат:

- маалыматтык жана жаңылык сайттары;
- блог платформалары;
- онлайн-дүкөндөр;
- социалдык сервистер;
- билим берүү жана тестирлөө системалары;
- корпоративдик маалыматтык системалар.

Django менен иштөөнү баштоо үчүн талап кылынган билимдер

Django фреймворкун үйрөнүү үчүн төмөнкү базалык билимдер жетиштүү:

- Python программалоо тилинин негиздери;
- функциялар жана модулдар менен иштөө;
- файл түзүмүн түшүнүү;
- HTML негизги элементтери.

Жыйынтык

Django — заманбап веб-иштеп чыгууда кеңири колдонулган, коопсуз жана функционалдуу фреймворк. Анын даяр компоненттери, автоматтык админ-панели жана архитектуралык түзүмү студенттерге веб-тиркемелерди иштеп чыгууну тез жана натыйжалуу өздөштүрүүгө жардам берет.

Тест



<http://qrcoder.ru/code/?https%3A%2F%2Fforms.gle%2FrpFNuyLYRpcPBXpo6&4&0>

3.2 Виртуалдык чөйрө (venv) жана пакеттерди башкаруу (pip)

Python менен иштегенде ар бир долбоор өз алдынча пакеттерге, китепканаларга жана версияларга муктаж болот. Дал ушундан улам **виртуалдык чөйрө (virtual environment, venv)** колдонулат. Ал ар бир долбоорду бөлүп, башка долбоорлорго таасир этпей иштөөгө мүмкүнчүлүк берет.

1. Виртуалдык чөйрө (venv) деген эмне?

Venv — бул ар бир долбоор үчүн өзүнчө Python чөйрөсү, анын ичинде:

- Өзүнчө китепканалар
- Өзүнчө pip орнотуулары
- Өзүнчө Python версиялары

Бул долбоорлор ортосунда пакет конфликттерин болтурбайт.

Мисал:

Бир долбоор Django 4.2 колдонот, башкасы Django 5.0 колдонот. Эгер виртуалдык чөйрө болбосо, бул экөөнү бир системада иштетүү мүмкүн эмес болмок.

2. Виртуалдык чөйрө түзүү

Python 3 версиясында `venv` демейки болуп келет.

Windows үчүн

```
python -m venv venv
```

Linux/macOS үчүн

```
python3 -m venv venv
```

Бул команда **venv** аттуу папканы түзөт, анын ичинде Python интерпретатору жана `pip` камтылган.

3. Виртуалдык чөйрөнү активдештирүү

Windows

```
venv\Scripts\activate
```

Linux/macOS

```
source venv/bin/activate
```

Активдештирилгенде командалык саптын алдында (`venv`) деген жазуу пайда болот.

4. Виртуалдык чөйрөнү өчүрүү (деактивация)

```
deactivate
```

5. Пакеттерди башкаруу (pip)

Пакет орнотуу

```
pip install django
```

Катталган пакетти жаңылоо

```
pip install --upgrade django
```

Пакетти өчүрүү

```
pip uninstall django
```

Орнотулган бардык пакеттерди көрүү

```
pip list
```

6. requirements.txt — долбоордун пакет тизмеси

Долбоорду башка компьютерге көчүргөндө бардык пакеттерди кайра орнотуу керек. Бул үчүн талаптар тизмеси колдонулат.

Тизмени түзүү

```
pip freeze > requirements.txt
```

Тизмедеги бардык пакеттерди орнотуу

```
pip install -r requirements.txt
```

7. Виртуалдык чөйрө менен иштөөдөгү жакшы тажрыйбалар

- Ар бир долбоорго өзүнчө venv түзүү
- Пакеттерди дайыма requirements.txt аркылуу башкаруу
- venv папкасын GitHub'га жүктөбөө
- Чөйрөнү активдештирмейинче пакет орнотпоо

Практикалык мисал (Venv жана pip)

Тапшырманын шарты

Жаңы Python долбоор үчүн виртуалдык чөйрө түзүп, аны активдештирип, **Django** пакетин орнотуңуз. Андан кийин орнотулган пакеттердин тизмесин сактаңыз.

Аткаруу кадамдары

1 Долбоор папкасын түзүү

```
mkdir my_project  
cd my_project
```

2 Виртуалдык чөйрө түзүү

```
python -m venv venv
```

3 Виртуалдык чөйрөнү активдештирүү

Windows: venv\Scripts\activate

Linux / macOS: source venv/bin/activate

Активдештирилгенде терминалда (venv) көрүнөт.

4 pip аркылуу пакет орнотуу

```
pip install django
```

5 Орнотулган пакеттерди текшерүү

```
pip list
```

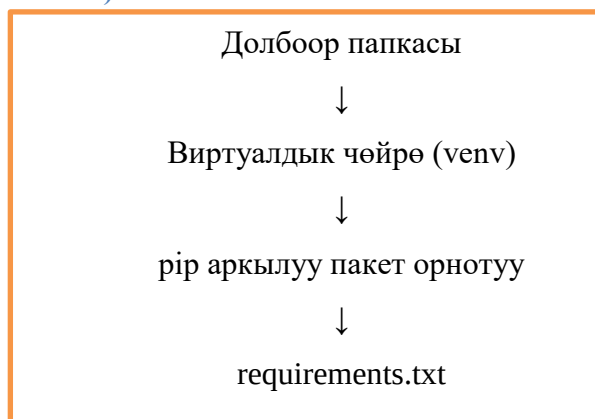
6 requirements.txt файлын түзүү

```
pip freeze > requirements.txt
```

Күтүлгөн жыйынтык

- venv папкасы түзүлөт
- Django пакети виртуалдык чөйрөгө орнотулат
- requirements.txt файлында орнотулган пакеттер сакталат

Тексттик схема (логикасы)



Жыйынтык

Бул практикалык мисал аркылуу студент:

- Виртуалдык чөйрө түзүүнү
- Аны активдештирүүнү
- рір менен пакет орнотууну
- Долбоор пакеттерин башкарууну үйрөнөт.

Практикалык тапшырма

Тапшырманын шарты

Жаңы Python долбоор түзүп, виртуалдык чөйрө (venv) аркылуу **requests** пакетин орнотунуз. Андан кийин орнотулган пакеттердин тизмесин файлга сактаңыз.

ТЕСТ



<http://qrcoder.ru/code/?https%3A%2F%2Fforms.gle%2FYZRXqhhM2fg8XV15A&4&0>

3.3 Проект түзүү жана структурасы

Django фреймворку менен иштөөнүн биринчи кадамы — жаңы веб-долбоорду түзүү. Django долбоору — бул бир же бир нече веб-тиркемелерди (app) камтыган структураланган система. Ар бир долбоор белгилүү бир ережелерге жана каталогдордун түзүлүшүнө ылайык келет. Бул бөлүмдө жаңы Django проектин түзүү жолдору жана анын ички структурасы каралат.

1. Жаңы Django проектин түзүү

Django орнотулгандан кийин төмөнкү команда аркылуу жаңы проект түзө аласыз:

`django-admin startproject mysite`

Бул команда *mysite* аттуу жаңы папканы түзүп, анын ичинде бир катар файлдарды жана каталогдорду камтыйт. Бул жерде *mysite* — проекттин аталышы, аны каалаганыңызча өзгөртсөңүз болот.

Проект түзүлгөндө структура төмөнкүдөй болот:

```
mysite/  
  manage.py  
  mysite/  
    __init__.py  
    settings.py  
    urls.py  
    asgi.py  
    wsgi.py
```

2. Django проекттиндеги негизги файлдар

Ар бир файл белгилүү бир ролду аткарат. Төмөндө негизги файлдардын функциялары кеңири түшүндүрүлгөн:

2.1 manage.py

- Проектти башкаруучу файл
- Серверди иштетүү, миграцияларды башкаруу, колдонмолорду түзүү жана башка Django буйруктарын аткарууга колдонулат

Мисалы, серверди иштетүү:

```
python manage.py runserver
```

2.2 settings.py

- Долбоордун бардык конфигурацияларын камтыйт
- Орнотулган app'тердин тизмеси (INSTALLED_APPS)
- Маалымат базасынын жөндөөлөрү
- Статикалык файлдар жана шаблондар
- Коопсуздук жөндөөлөрү (SECRET_KEY)
- Убакыт жана тил параметрлери

2.3 urls.py

- Веб-сайттын маршруттарын (URL) башкарат
- Колдонуучу кайсы URL'га кайрылса, кайсы view ишке кирерин аныктайт

Мисалы:

```
from django.urls import path
from . import views
```

```
urlpatterns = [
    path("", views.home),
]
```

2.4 init.py

- Бул файл директорийди Python модулуна айлантат
- Көбүнчө бош болот жана автоматтык түзүлөт

2.5 wsgi.py жана asgi.py

- Projectти серверге жайгаштырганда колдонулат
- **wsgi.py** — салттуу синхрондук серверлерге ылайыктуу
- **asgi.py** — асинхрондук серверлер жана WebSocket’тер үчүн колдонулат

3. Django проект менен арртын айырмасы

Түшүндүрмө Проект		Арр
Максат	Жалпы система	Белгилүү бир функция
Мисал	Веб-сайт	Блог, комментарий, каттоо
Колдонуу	Бир нече арр болушу мүмкүн Көп долбоорлордо кайра колдонсо болот	

Проект — бүт система

Арр — система ичиндеги өзүнчө модул же функция

4. Жаңы арр түзүү

Проект түзүлгөндөн кийин бир же бир нече application түзүүгө болот:

```
python manage.py startapp blog
```

Арр структурасы төмөнкүдөй:

blog/

admin.py

apps.py

models.py

tests.py

views.py

migrations/

- **admin.py** — админ панелин конфигурациялоо
- **apps.py** — арр жөндөөлөрү

- **models.py** — маалымат базасынын структурасы
- **views.py** — логика жана HTTP жооптор
- **tests.py** — тесирлөө коду
- **migrations/** — базага өзгөртүүлөрдү сактоо

5. Структуранын мааниси

- Кодду иреттүү жана масштабдуу кармоо
- Командалык иштөөгө ылайыктуу
- Долбоорду кеңейтүү жана жаңылоо оңой

Django структурасы долбоорду профессионалдуу жана коопсуз кылып көрсөтөт.

Практикалык мисал

Мисал: Жаңы Django проект түзүү жана биринчи app кошуу

1. Жаңы Django проект түзүү

```
django-admin startproject mysite
```

2. Проект структурасы

```
mysite/
```

```
manage.py
```

```
mysite/
```

```
__init__.py
```

```
settings.py
```

```
urls.py
```

```
wsgi.py
```

```
asgi.py
```

3. Серверди иштетүү

```
python manage.py runserver
```

4. Жаңы app түзүү (мисалы blog)

```
python manage.py startapp blog
```

App структурасы

```
blog/
```

```
admin.py
```

```
apps.py
```

```
models.py
```

views.py
tests.py
migrations/

Мисалдын максаты:

- Django проект түзүүнү көрсөтүү
- App түзүүнү көрсөтүү
- Структураны таанышуу

Практикалык тапшырмалар

Тапшырма 1 — Жаңы проект түзүү

- 1 Django орнотулганын текшер
- 2 myproject аттуу жаңы проект түз
- 3 Серверди иштетип браузерде ач (<http://127.0.0.1:8000>)

Тапшырма 2 — App түзүү

- 1 shop аттуу жаңы app түз
- 2 App структурасын карап чык
- 3 shop артти INSTALLED_APPS тизмесине кош

Тапшырма 3 — URL жана view түзүү

- 1 shop/views.py ичинде төмөнкү view түз:
from django.http import HttpResponse

```
def home(request):  
    return HttpResponse("Welcome to Shop App")
```

- 2 mysite/urls.py ичинде маршрут кош:
from django.urls import path
from shop import views

```
urlpatterns = [  
    path("", views.home),  
]
```

- 3 Серверди иштетип, браузерде <http://127.0.0.1:8000/> ачып, текстти көр



<http://qrcoder.ru/code/?https%3A%2F%2Fforms.gle%2FqvNP5DKrnNVYv6v36&6&0>

3.4 Алгачкы view жазуу

Django веб-тиркемесинде **view** — бул колдонуучунун суроосуна жооп берип, маалыматтарды даярдоочу жана аларды веб-баракка жөнөтүүчү логика бөлүгү. View'лар Python функциялары же класстар болушу мүмкүн. Бул бөлүмдө view түзүүнүн негиздери жана URL менен байланыштыруу практикалык түрдө көрсөтүлөт.

1. View деген эмне?

- View — колдонуучу браузерден келген HTTP суроосун иштетет
- Маалыматты базадан алып, шаблонго жөнөтөт
- Натыйжада HTML, JSON, же башка форматтагы жооп даярдайт

Мисалы, жөнөкөй view:

```
blog/views.py
```

```
from django.http import HttpResponse
```

```
def home(request):
```

```
    return HttpResponse("Салам, бул менин биринчи view!")
```

Бул view колдонуучуга текст түрүндө жооп кайтарат.

2. URL менен байланыштыруу

View'ды иштетүү үчүн аны URL менен байланыштыруу керек. Бул үчүн urls.py файлын колдонобуз.

```
mysite/urls.py
```

```
from django.contrib import admin
```

```
from django.urls import path
```

```
from blog import views
```

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path("", views.home), башкы баракка view туташтыруу  
]
```

- path("", views.home) — башкы баракка (http://127.0.0.1:8000/) колдонуучу кайрылганда home view ишке кирет
- Ар кандай URL'дерди кошуп, башка view'ларды туташтырууга болот

3. Template колдонуу

View'лар HTML шаблондор менен иштей алат. Мисалы:

1. Шаблон түзүү:

```
blog/  
    templates/  
        blog/  
            home.html
```

2. HTML файлдын ичинде:

```
<!-- blog/templates/blog/home.html -->  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Менин биринчи сайтым</title>  
</head>  
<body>  
    <h1>Салам, бул менин биринчи Django сайты!</h1>  
</body>  
</html>
```

3. View'да шаблонду колдонуу:

```
from django.shortcuts import render
```

```
def home(request):  
    return render(request, 'blog/home.html')
```

4. Практикалык кеңештер

- Бардык view'ларды views.py файлына жайгаштыруу

- Күрөө view'лар үчүн класстар (Class-Based Views) колдонсо болот
- URL'дерди urls.py аркылуу туура конфигурациялоо
- Template'терди templates/app_name/ папкасында сактоо

5. Колдонуучу суроолоруна жооп берүү

View'дар ар кандай маалыматтарды көрсөтө алат:

- Текст (HttpResponse)
- HTML (шаблон аркылуу)
- JSON (API үчүн)

Мисалы, JSON жооп:

```
from django.http import JsonResponse
```

```
def api_data(request):
    data = {"message": "Бул JSON жооп!"}
    return JsonResponse(data)
```

6. Жыйынтык

Алгачкы view жазуу — Django'догу негизги кадамдардын бири. View түзүп, аны URL менен байланыштыруу аркылуу веб-тиркеменин негизги логикасы иштеп баштайт. Кийин шаблондор, форма, база менен иштөө жана динамикалык контент кошуу аркылуу тиркеме толук функционалдуу веб-сайтка айланат.

Практикалык мисал

Мисал: Жөнөкөй view түзүп, башкы баракка туташтыруу

```
# blog/views.py
```

```
from django.http import HttpResponse
```

```
def home(request):
    return HttpResponse("Салам, бул менин биринчи view!")
```

```
# mysite/urls.py
```

```
from django.contrib import admin
```

```
from django.urls import path
```

```
from blog import views
```

```
urlpatterns = [
```

```
    path('admin/', admin.site.urls),
```

```
    path("", views.home), # башкы баракка view туташтыруу
```

]

Натыйжа:

Браузерде `http://127.0.0.1:8000/` ачканда:

Салам, бул менин биринчи view!

Практикалык тапшырмалар (3)

Тапшырма 1 — Жөнөкөй view

1 about деген view түз:

```
def about(request):
```

```
    return HttpResponse("Бул менин Django сайтымдын 'About' барагы")
```

2 URL'ге туташтыр: `http://127.0.0.1:8000/about/`

Тапшырма 2 — HTML шаблон колдонуу

1 home.html деген шаблон түз:

```
<h1>Салам, бул менин биринчи Django сайты!</h1>
```

2 views.py ичиндеги home view'да render колдон:

```
from django.shortcuts import render
```

```
def home(request):
```

```
    return render(request, 'blog/home.html')
```

3 Серверди иштетип, браузерде текшер

Тапшырма 3 — JSON жооп

1 api_data деген view түз:

```
from django.http import JsonResponse
```

```
def api_data(request):
```

```
    data = {"message": "Бул JSON жооп!"}
```

```
    return JsonResponse(data)
```

2 URL'ге туташтыр: `http://127.0.0.1:8000/api/`

3 Браузерде JSON пайда болгонун текшер



<http://qrcoder.ru/code/?https%3A%2F%2Fforms.gle%2FXadv4CHNUs2FSfB47&6&0>

3.5 Модельдер жана маалымат базасы менен байланыш (Models)

Django'до **Model** — бул маалымат базасынын структурасын аныктоочу жана базадагы маалыматтар менен иштөөчү класстар. Модельдер ORM (Object-Relational Mapping) аркылуу SQL суроолорду жазбай эле Python класстары менен маалымат базасы менен иштөөгө мүмкүнчүлүк берет.

Бул бөлүмдө биз моделдерди түзүүнүн негиздерин, аларды база менен кантип байланыштыруу керектигин жана практикалык мисалдарды карайбыз.

1. Model деген эмне?

- Model — таблица сыяктуу структура
- Ар бир класс **таблицага** туура келет
- Ар бир класстын атрибуту **таблица тилкесине** туура келет
- Django ORM автоматтык түрдө SQL суроолорду түзөт

Мисалы, студенттер таблицасы:

```
blog/models.py
```

```
from django.db import models
```

```
class Student(models.Model):
```

```
    name = models.CharField(max_length=100)
```

```
    age = models.IntegerField()
```

```
    email = models.EmailField(unique=True)
```

```
    def __str__(self):
```

```
        return self.name
```

- CharField — текст тилкеси
- IntegerField — бүтүн сан тилкеси
- EmailField — уникалдуу электрондук почта
- __str__ — объектти текст катары көрсөтүү

2. Миграциялар (Migrations)

Модель түзүлгөндөн кийин, базага өзгөртүүлөрдү киргизүү керек. Django миграциялар аркылуу бул процессин автоматташтырат.

1. Миграция файлдарын түзүү:

```
python manage.py makemigrations
```

2. Өзгөртүүлөрдү базага колдонуу:

```
python manage.py migrate
```

Мунун натыйжасында Django **SQLite** же башка базада таблицаны түзөт.

3. Админ панел аркылуу модел менен иштөө

Модельди админ панелде көрсөтүү үчүн, admin.py файлын колдонобуз:

```
blog/admin.py
```

```
from django.contrib import admin
```

```
from .models import Student
```

```
admin.site.register(Student)
```

Бул менен админ панелде студенттерди кошуу, өчүрүү жана өзгөртүү мүмкүнчүлүгү пайда болот.

4. Маалымат кошуу жана алуу

4.1 Жаңы объект кошуу

```
student = Student(name="Aizada", age=20, email="aizada@example.com")
```

```
student.save()
```

4.2 Бардык объекттерди алуу

```
students = Student.objects.all()
```

```
for s in students:
```

```
    print(s.name, s.age)
```

4.3 Шарт менен тандоо

```
young_students = Student.objects.filter(age__lt=21)
```

- filter() — шарт менен тандоо
- age__lt=21 — жашы 21ден кичине болгон студенттер

4.4 Объектти өзгөртүү

```
student = Student.objects.get(id=1)
```

```
student.age = 21
```

```
student.save()
```

4.5 Объекти өчүрүү

```
student = Student.objects.get(id=1)
```

```
student.delete()
```

5. Модельдердин артыкчылыктары

- SQL жазбай эле база менен иштөө
- ORM аркылуу Python объекттери менен иштөө жеңил
- База логикасын код менен интеграциялоо
- Админ панелде автоматтык көрсөткүч

5. Практикалык мисал

Мисал: *Студенттер модели түзүү жана базага колдонуу*

```
# blog/models.py
```

```
from django.db import models
```

```
class Student(models.Model):
```

```
    name = models.CharField(max_length=100)
```

```
    age = models.IntegerField()
```

```
    email = models.EmailField(unique=True)
```

```
    def __str__(self):
```

```
        return self.name
```

```
# Миграция түзүү жана базага колдонуу
```

```
python manage.py makemigrations
```

```
python manage.py migrate
```

```
# Жаңы студент кошуу
```

```
student = Student(name="Aizada", age=20, email="aizada@example.com")
```

```
student.save()
```

```
# Бардык студенттерди алуу
```

```
students = Student.objects.all()
```

```
for s in students:
```

```
    print(s.name, s.age)
```

```
# Шарт менен тандоо
young_students = Student.objects.filter(age__lt=21)

# Объектти жаңылоо
student = Student.objects.get(id=1)
student.age = 21
student.save()
```

```
# Объектти өчүрүү
student = Student.objects.get(id=1)
student.delete()
```

Максат: Модель түзүү, базага колдонуу, объекттер менен иштөө

7. Практикалык тапшырмалар

Тапшырма 1 — Жаңы модель түзүү

1 Course деген модель түз:

- name — CharField(100)
- duration — IntegerField
- description — TextField

2 Миграция түзүп, базага колдон

Тапшырма 2 — Объект кошуу жана алуу

1 3 курс маалыматтарын кош:

```
Course.objects.create(name="Python", duration=30, description="Python курсу")
```

2 Бардык курстарды алып, атын жана узактыгын басып чыгар

Тапшырма 3 — Объектти өзгөртүү жана өчүрүү

1 Бир курс менен иштөө: узактыгын өзгөртүү

2 Бир курс объекттин өчүрүү

8. Жыйынтык

Django моделдери веб-тиркеменин маалымат базасы менен түз байланышкан негизги компоненти болуп саналат. Model аркылуу таблицалар түзүлөт, маалымат кошулат, жаңыланат жана өчүрүлөт. Миграциялар жана админ панель менен бирге, модельдер долбоорду тез жана натыйжалуу иштеп чыгууга мүмкүндүк берет.

Тест



<https://forms.gle/GiTaK8xGHgooL6MYA>

3.6 Формалар жана шаблондор менен иштөө (Forms & Templates)

Веб-тиркемелерде колдонуучу менен өз ара аракеттешүү өтө маанилүү. Django бул үчүн **формалар (Forms)** жана **шаблондор (Templates)** механизмдерин сунуштайт. Формалар аркылуу колдонуучу маалымат киргизет, шаблондор аркылуу маалыматты браузерде көрсөтөбүз.

Бул бөлүмдө формаларды түзүү, шаблонду колдонуу жана view менен интеграциялоо көрсөтүлөт.

1. Django шаблондору (Templates)

- HTML файлдар, колдонуучуга көрүнгөн интерфейс
- View'дар аркылуу маалымат менен байланышат
- templates папкасында сакталат

1.1 Шаблон түзүү

Структура мисалы:

blog/

templates/

blog/

home.html

student_form.html

home.html мисалы:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>Менин биринчи сайтым</title>
```

```
</head>
```

```
<body>
```

```
<h1>Салам, бул менин биринчи Django сайты!</h1>
</body>
</html>
```

1.2 View аркылуу шаблонду көрсөтүү

```
blog/views.py
from django.shortcuts import render

def home(request):
    return render(request, 'blog/home.html')
    • render() — шаблонду view’дан кайтаруу функциясы
    • Биринчи параметр — HTTP суроо, экинчи — шаблон файлы
```

2. Django формалары (Forms)

Формалар колдонуучудан маалымат алуу үчүн колдонулат.

2.1 Форманы түзүү

```
blog/forms.py
from django import forms
from .models import Student

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'age', 'email']
    • ModelForm — модель менен байланыштырат
    • fields — формада кайсы талаалар көрсөтүлөрүн аныктайт
```

2.2 View’да форма колдонуу

```
blog/views.py
from django.shortcuts import render, redirect
from .forms import StudentForm

def add_student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('home')
```

else:

```
form = StudentForm()
```

```
return render(request, 'blog/student_form.html', {'form': form})
```

- request.method — суроо түрүн текшерет (GET же POST)
- form.is_valid() — маалымат туура келерин текшерет
- form.save() — жаңы студентти базага кошот

2.3 HTML шаблон формасы

```
<!-- blog/templates/blog/student_form.html -->
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>Студент кошуу</title>
```

```
</head>
```

```
<body>
```

```
  <h1>Студент кошуу формасы</h1>
```

```
  <form method="POST">
```

```
    {% csrf_token %}
```

```
    {{ form.as_p }}
```

```
    <button type="submit">Жөнөтүү</button>
```

```
  </form>
```

```
</body>
```

```
</html>
```

- {% csrf_token %} — коопсуздукту камсыз кылат
- {{ form.as_p }} — форма талааларын параграфтарда көрсөтөт

3. URL менен формага туташуу

```
blog/urls.py
```

```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [
```

```
  path("", views.home, name='home'),
```

```
  path('add/', views.add_student, name='add_student'),
```

```
]
```

- URL аркылуу view туташтырылат
- Колдонуучу http://127.0.0.1:8000/add/ дарегине кирсе, форма ачылат

4. Практикалык кеңештер

- Бардык шаблондор templates/app_name/ папкасында болсун
- Формаларды forms.py файлына жайгаштырыңыз
- View'дар жана шаблондор ортосундагы маалымат алмашууну туура пландаштырыңыз
- CSRF токени унутпаңыз

5. Практикалык мисал

Мисал: Студент кошуу формасы жана шаблонду колдонуу

```
# blog/forms.py
```

```
from django import forms
```

```
from .models import Student
```

```
class StudentForm(forms.ModelForm):
```

```
    class Meta:
```

```
        model = Student
```

```
        fields = ['name', 'age', 'email']
```

```
# blog/views.py
```

```
from django.shortcuts import render, redirect
```

```
from .forms import StudentForm
```

```
def add_student(request):
```

```
    if request.method == 'POST':
```

```
        form = StudentForm(request.POST)
```

```
        if form.is_valid():
```

```
            form.save()
```

```
            return redirect('home')
```

```
    else:
```

```
        form = StudentForm()
```

```
    return render(request, 'blog/student_form.html', {'form': form})
```

```
<!-- blog/templates/blog/student_form.html -->
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <title>Студент кошуу</title>
```

```
</head>
```

```

<body>
  <h1>Студент кошуу формасы</h1>
  <form method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Жөнөтүү</button>
  </form>
</body>
</html>

```

```

# blog/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path("", views.home, name='home'),
    path('add/', views.add_student, name='add_student'),
]

```

Натыйжа:

Колдонуучу <http://127.0.0.1:8000/add/> дарегине кирсе, студент кошуу формасы ачылат жана маалымат базага кошулат.

7. Практикалык тапшырмалар

Тапшырма 1 — Жаңы форма түзүү

- 1 CourseForm түзгүлө, Course моделине байланыштуу (fields: name, duration, description)
- 2 Форманы шаблон аркылуу көрсөтүп, POST суроосу менен базага кошуңуз

Тапшырма 2 — Шаблонду кеңейтүү

- 1 home.html бетине шилтеме кош:

```
<a href="{% url 'add_student' %}">Жаңы студент кошуу</a>
```

- 2 Шаблонду view менен туташтырыңыз

Тапшырма 3 — CSRF токен жана валидация

- 1 StudentForm аркылуу киргизилген маалыматтын тууралыгын текшериниз
- 2 Валидация туура болбосо, форма кайра көрсөтүлсүн жана каталарды көрсөтсүн

8. Жыйынтык

Формалар жана шаблондор веб-тиркемелердин негизги компоненти болуп эсептелет. Form менен колдонуучу маалымат киргизет, Template аркылуу маалыматты

көрүүгө мүмкүнчүлүк берет. Бул механизмдер view менен бирге интеграцияланып, динамикалык, интерактивдүү веб-сайттарды түзүүгө мүмкүндүк берет.

Тест



<https://forms.gle/C1xCCNsTvpEZZCxeA>

3.7 Админ панель жана Django'дун автоматтык интерфейси

Django'нун уникалдуу артыкчылыктарынын бири — **автоматтык админ панель**. Бул веб-интерфейс иштеп чыгуучуларга жана администраторлорго маалымат базасын түзөтүүгө, өзгөртүүгө жана өчүрүүгө мүмкүнчүлүк берет. Админ панель долбоорду тез иштеп чыгууга жана маалыматтарды башкарууга жеңилдик түзөт.

1. Админ панелге кирүү

- Админ панель стандарттык URL аркылуу жеткиликтүү:
`http://127.0.0.1:8000/admin/`
- Алгач суперколдонуучу түзүү керек:
`python manage.py createsuperuser`
- Колдонуучу аты, email жана сырсөз киргизилип, суперколдонуучу аккаунт түзүлөт
- Андан кийин браузерден админ панелге кирүүгө болот

2. Моделди админ панелде көрсөтүү

Модель админ панелде көрүнүшү үчүн `admin.py` файлын конфигурациялоо керек:

```
blog/admin.py
from django.contrib import admin
from .models import Student
```

```
admin.site.register(Student)
```

- `register()` — моделди админ панелде каттоого алат

- Модел кошулгандан кийин, админ панелде студенттерди кошуп, өчүрүп жана өзгөртүүгө болот

2.1 Админ панельдин интерфейси

Админ панельде ар бир моделге өзүнчө таблица көрсөтүлөт:

- Бардык объекттерди көрүү
- Жаңы объект кошуу
- Барларын түзөтүү же өчүрүү
- Сорттоо жана издөө мүмкүнчүлүктөрү

Бул интерфейс автоматтык түзүлгөндүктөн, веб-разработкада убакытты үнөмдөйт.

3. Кастомизациялоо

Админ панельди колдонуучунун муктаждыгына ылайык оңой өзгөртүүгө болот:

```
blog/admin.py
```

```
from django.contrib import admin
```

```
from .models import Student
```

```
class StudentAdmin(admin.ModelAdmin):
```

```
    list_display = ('name', 'age', 'email')  таблицада кайсы талаалар көрүнөрүн белгилөө
```

```
    search_fields = ('name', 'email')      издөө талаалары
```

```
admin.site.register(Student, StudentAdmin)
```

- list_display — таблицадагы тилкелер
- search_fields — издөө мүмкүнчүлүгү
- Бул аркылуу админ панель функционалдуулугун кеңейтүүгө болот

4. Практикалык кеңештер

- Ар бир жаңы модельди админ панелде көрсөтүү үчүн каттоодон өткөрүү
- Көп моделдер бар болсо, ModelAdmin аркылуу кастомизациялоо
- Колдонуучулар жана уруксаттар менен иштөөнү унутпоо
- Админ панель аркылуу тестирлөөнү жүргүзүү

5. Практикалык мисал

Максат: Student моделин админ панелде көрсөтүү жана кастомизациялоо.

1. **Модель түзүү** – models.py:

```
from django.db import models
```

```
class Student(models.Model):
```

```
    name = models.CharField(max_length=100)
```

```
    age = models.IntegerField()
```

```
email = models.EmailField(unique=True)
```

```
def __str__(self):  
    return self.name
```

2. Миграцияларды түзүү жана базага колдонуу:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

3. Админ панелде көрсөтүү – admin.py:

```
from django.contrib import admin
```

```
from .models import Student
```

```
class StudentAdmin(admin.ModelAdmin):
```

```
    list_display = ('name', 'age', 'email')
```

```
    search_fields = ('name', 'email')
```

```
admin.site.register(Student, StudentAdmin)
```

4. Суперколдонуучу түзүү:

```
python manage.py createsuperuser
```

Браузерде: <http://127.0.0.1:8000/admin/>

Эми Student модели админ панелде көрсөтүлөт, издөө жана тилке көрсөтүү мүмкүнчүлүктөрү менен.

6. Практикалык тапшырма

1. Жаңы модель түзгүлө: Teacher (аты, кафедра, email), аны админ панелде каттаңыз.
2. StudentAdmin классында жаңы тилке кошуп, жашы боюнча сорттоо мүмкүнчүлүгүн кошуңуз.
3. Админ панелде бир нече Student объекттерин кошуп, издөө жана sort функционалын сыноо.

7. Жыйынтык

Админ панель Django'дун күчтүү куралы болуп саналат. Ал маалымат базасын башкарууну жеңилдетет, убакытты үнөмдөйт жана веб-тиркемени профессионалдуу көрсөтөт. Модельдер, view жана админ панель бирге интеграцияланып, долбоорду толук функционалдуу веб-сайтка айлантат.



<https://forms.gle/qTLoP1jJaXgfxCua9>

IV БӨЛҮМ: ПРАКТИКАЛЫК ДОЛБООРОЛОР

4.1 Django менен блог тиркемесин түзүү

Бул бөлүмдө студенттерге **Django фреймворку** менен жөнөкөй блог тиркемесин түзүүнүн кадамдары көрсөтүлөт. Практикалык долбоор аркылуу студенттер теориялык билимдерди колдонуп, толук веб-тиркемени иштеп чыгууну үйрөнөт.

1. Жаңы проект түзүү

Төмөнкү команданы аткарып, жаңы Django проектин түзгүлө:

```
django-admin startproject myblog
```

```
cd myblog
```

Структурасы төмөнкүдөй болот:

```
myblog/
```

```
    manage.py
```

```
    myblog/
```

```
        __init__.py
```

```
        settings.py
```

```
        urls.py
```

```
        asgi.py
```

```
        wsgi.py
```

2. Жаңы app түзүү

Блог функционалын ишке ашыруу үчүн app түзгүлө:

```
python manage.py startapp blog
```

Структурасы:

blog/

admin.py

apps.py

models.py

views.py

urls.py өз алдынча түзсө болот

templates/

migrations/

- Бардык блог менен байланышкан код blog/ папкасында жайгашат

3. Модел түзүү

blog/models.py файлына пост моделин кошуңуз:

```
from django.db import models
```

```
class Post(models.Model):
```

```
    title = models.CharField(max_length=200)
```

```
    content = models.TextField()
```

```
    created_at = models.DateTimeField(auto_now_add=True)
```

```
    def __str__(self):
```

```
        return self.title
```

- title — посттун аталышы
- content — текст
- created_at — түзүлгөн дата автоматтык кошулат

4. Миграциялар жана база

Миграция жасап, өзгөртүүлөрдү базага колдонуу:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

5. Админ панель

blog/admin.py файлын өзгөртүү:

```
from django.contrib import admin
```

```
from .models import Post
```

```
admin.site.register(Post)
```

- Админ панелде блог постторун кошууга, өзгөртүүгө жана өчүрүүгө мүмкүнчүлүк пайда болот

6. View түзүү

blog/views.py файлына төмөнкү view'ларды кошуңуз:

```
from django.shortcuts import render, get_object_or_404
from .models import Post
```

```
def home(request):
```

```
    posts = Post.objects.all()
```

```
    return render(request, 'blog/home.html', {'posts': posts})
```

```
def post_detail(request, post_id):
```

```
    post = get_object_or_404(Post, id=post_id)
```

```
    return render(request, 'blog/post_detail.html', {'post': post})
```

- home — бардык постторду көрсөтөт
- post_detail — жеке постту көрсөтөт

7. URL туташтыруу

blog/urls.py файлын түзгүлө:

```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [
```

```
    path("", views.home, name='home'),
```

```
    path('post/<int:post_id>/', views.post_detail, name='post_detail'),
```

```
]
```

myblog/urls.py файлына блог URL'дарын кошуу:

```
from django.contrib import admin
```

```
from django.urls import path, include
```

```
urlpatterns = [
```

```
    path('admin/', admin.site.urls),
```

```
    path("", include("blog.urls")),
```

```
]
```

8. Шаблон түзүү

blog/templates/blog/home.html:

```

<!DOCTYPE html>
<html>
<head>
  <title>Блог</title>
</head>
<body>
  <h1>Блог посттору</h1>
  <ul>
    {% for post in posts %}
      <li><a href="{% url 'post_detail' post.id %}">{{ post.title }}</a></li>
    {% endfor %}
  </ul>
</body>
</html>

```

blog/templates/blog/post_detail.html:

```

<!DOCTYPE html>
<html>
<head>
  <title>{{ post.title }}</title>
</head>
<body>
  <h1>{{ post.title }}</h1>
  <p>{{ post.content }}</p>
  <p><a href="{% url 'home' %}">Артка</a></p>
</body>
</html>

```

9. Серверди иштетүү

python manage.py runserver

- <http://127.0.0.1:8000/> — блогдун башкы барагы
- <http://127.0.0.1:8000/admin/> — админ панель

10. Практикалык тапшырма

1. Блогго жаңы Category моделин түзгүлө: аталыш жана сүрөт талаалары менен, аны админ панелде көрсөтүңүз.
2. home.html шаблондорунда постторду категориясы менен топтоо (category-wise display) функциясын кошуңуз.

3. Админ панелде постторду түзүү, өчүрүү жана түзөтүү менен тажрыйба жасаңыз, ошондой эле `post_detail` URL аркылуу жеке постту ачып көрүңүз.

11. Практикалык кеңештер

- Модельдерди админ панел аркылуу кошуп, тест жүргүзүңүз
- View'ларды модулдук түрдө жазып, URL менен байланыштырыңыз
- Шаблондорду логикалык папка түзүмүндө сактаңыз
- Серверди иштетип, иштеп жаткан веб-сайтты браузерде текшерип туруңуз

12. Жыйынтык

Бул практикалык долбоор студенттерге:

- Django проект жана app түзүүнү
- Модель менен база байланышын
- View жана шаблон аркылуу маалыматты көрсөтүүнү
- URL аркылуу навигацияны уюштурууну үйрөтөт.

Бул кадамдарды аткаруу менен студент толук функционалдуу блог веб-сайтын түзө алат.

Тест



<https://forms.gle/HnWMCCcea7BDgdCL7>

4.2 Django + SQLite менен Student Management системасы

Бул бөлүмдө студенттерге **Django** фреймворку жана **SQLite** маалымат базасын колдонуп, жөнөкөй **Student Management System (SMS)** түзүүнүн кадамдары көрсөтүлөт. Бул система студенттерди кошууга, өзгөртүүгө, өчүрүүгө жана көрсөтүүгө мүмкүнчүлүк берет.

1. Жаңы Django проект түзүү

```
django-admin startproject student_mgmt
```

```
cd student_mgmt
```

Структурасы:

```
student_mgmt/  
  manage.py  
student_mgmt/  
  __init__.py  
  settings.py  
  urls.py  
  asgi.py  
  wsgi.py
```

2. Жаңы app түзүү

python manage.py startapp students

Структурасы:

```
students/  
  admin.py  
  apps.py  
  models.py  
  views.py  
  urls.py  өз алдынча түзсө болот  
  templates/  
  migrations/
```

- Бардык студенттер менен байланышкан код students/ папкасында жайгашат

3. Модель түзүү

students/models.py файлына студент моделин кошуңуз:

```
from django.db import models
```

```
class Student(models.Model):
```

```
    first_name = models.CharField(max_length=50)
```

```
    last_name = models.CharField(max_length=50)
```

```
    email = models.EmailField(unique=True)
```

```
    age = models.IntegerField()
```

```
    enrollment_date = models.DateField(auto_now_add=True)
```

```
    def __str__(self):
```

```
        return f"{self.first_name} {self.last_name}"
```

- CharField — текст

- EmailField — уникалдуу электрондук почта
- IntegerField — жаш
- DateField(auto_now_add=True) — автоматтык каттоо датасы

4. Миграциялар жана база

```
python manage.py makemigrations
```

```
python manage.py migrate
```

- SQLite базага таблица түзүлөт

5. Админ панель

students/admin.py файлын өзгөртүү:

```
from django.contrib import admin
```

```
from .models import Student
```

```
admin.site.register(Student)
```

- Админ панелде студенттерди кошууга, өзгөртүүгө жана өчүрүүгө мүмкүнчүлүк пайда болот

6. Формалар түзүү

students/forms.py:

```
from django import forms
```

```
from .models import Student
```

```
class StudentForm(forms.ModelForm):
```

```
    class Meta:
```

```
        model = Student
```

```
        fields = ['first_name', 'last_name', 'email', 'age']
```

- ModelForm аркылуу модел менен форма байланыштырылат

7. View түзүү

students/views.py:

```
from django.shortcuts import render, redirect, get_object_or_404
```

```
from .models import Student
```

```
from .forms import StudentForm
```

Бардык студенттерди көрсөтүү

```
def student_list(request):
```

```
    students = Student.objects.all()
```

```
    return render(request, 'students/student_list.html', {'students': students})
```

Жаңы студент кошуу

```
def student_create(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('student_list')
    else:
        form = StudentForm()
    return render(request, 'students/student_form.html', {'form': form})
```

Студентти жаңылоо

```
def student_update(request, pk):
    student = get_object_or_404(Student, pk=pk)
    if request.method == 'POST':
        form = StudentForm(request.POST, instance=student)
        if form.is_valid():
            form.save()
            return redirect('student_list')
    else:
        form = StudentForm(instance=student)
    return render(request, 'students/student_form.html', {'form': form})
```

Студентти өчүрүү

```
def student_delete(request, pk):
    student = get_object_or_404(Student, pk=pk)
    if request.method == 'POST':
        student.delete()
        return redirect('student_list')
    return render(request, 'students/student_confirm_delete.html', {'student': student})
```

8. URL туташтыруу

students/urls.py:

```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [  
    path("", views.student_list, name='student_list'),  
    path('new/', views.student_create, name='student_create'),  
    path('edit/<int:pk>/', views.student_update, name='student_update'),  
    path('delete/<int:pk>/', views.student_delete, name='student_delete'),  
]
```

student_mgmt/urls.py файлына кошуу:

```
from django.contrib import admin  
from django.urls import path, include
```

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path("", include('students.urls')),  
]
```

9. Шаблондор түзүү

9.1 Студенттер тизмеси

students/templates/students/student_list.html:

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Student Management</title>  
</head>  
<body>  
    <h1>Студенттер тизмеси</h1>  
    <a href="{% url 'student_create' %}">Жаңы студент кошуу</a>  
    <ul>  
        {% for student in students %}  
        <li>  
            {{ student.first_name }} {{ student.last_name }} -  
            <a href="{% url 'student_update' student.pk %}">Өзгөртүү</a> |  
            <a href="{% url 'student_delete' student.pk %}">Өчүрүү</a>  
        </li>
```

```

        {% endfor %}
    </ul>
</body>
</html>

```

9.2 Студент формасы

students/templates/students/student_form.html:

```

<!DOCTYPE html>
<html>
<head>
    <title>Студент формасы</title>
</head>
<body>
    <h1>Студент кошуу/өзгөртүү</h1>
    <form method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Сактоо</button>
    </form>
</body>
</html>

```

9.3 Студентти өчүрүү тастыктоо

students/templates/students/student_confirm_delete.html:

```

<!DOCTYPE html>
<html>
<head>
    <title>Студентти өчүрүү</title>
</head>
<body>
    <h1>{{ student.first_name }} {{ student.last_name }} өчүрүлөтбү?</h1>
    <form method="POST">
        {% csrf_token %}
        <button type="submit">Ооба, өчүрүү</button>
        <a href="{% url 'student_list' %}">Жок, артка кайтуу</a>
    </form>
</body>

```

</html>

10. Серверди иштетүү

`python manage.py runserver`

- `http://127.0.0.1:8000/` — студенттер тизмеси
- `http://127.0.0.1:8000/admin/` — админ панель

11. Практикалык тапшырма

1. Студенттерге `grade` талаасын кошуп, админ панелде жана формада көрсөтүңүз.
2. Студенттер тизмесин жашы боюнча сорттоого жана издөөгө мүмкүнчүлүк кошуңуз.
3. Студентти өчүрүүдө тастыктоо барагынын дизайнун Bootstrap менен жакшыртыңыз.

12. Практикалык кеңештер

- Моделдерди админ панелде текшерип туруңуз
- Формаларды `view` менен туура интеграциялоо
- Шаблондорду логикалык структурада сактоо
- URL аркылуу навигацияны ыңгайлуу уюштуруу

13. Жыйынтык

Бул практикалык долбоор студенттерге:

- Django проект жана `app` түзүүнү
- Модель менен SQLite базасын колдонуп иштөө
- View жана форма аркылуу CRUD операцияларын аткарууну
- Шаблон жана URL аркылуу веб-интерфейс түзүүнү үйрөтөт

Бул кадамдарды аткаруу менен студент толук функционалдуу **Student Management System** түзө алат.

Тест



4.3 Django'ну серверге жайгаштыруу (деплой)

1. Теориялык негиздер

Django'ну иштеп чыккан соң, локалдык компьютерде гана эмес, интернет аркылуу колдонуучулар кире ала турган серверге жайгаштыруу керек. Бул процесс **деплой** деп аталат. Бул бөлүмдө Django'ну серверге жайгаштыруу, сервер конфигурациялоо жана негизги практикалык кадамдар көрсөтүлөт.

1. Деплой деген эмне?

- Деплой – веб-тиркемени локалдык компьютерден **продакшн серверге** көчүрүү жана иштетүү процесси.
- Продакшн сервер – интернетке туташкан, колдонуучулар кире ала турган сервер.
- Django'ну деплойлоо аркылуу веб-сайт **туруктуу, коопсуз жана тез** иштей алат.

Деплой — Django долбоорун локалдык чөйрөдөн серверге өткөрүп, интернеттен жеткиликтүү кылуу процесси.

Негизги этаптар:

1. **Сервер тандоо** — VPS (DigitalOcean, AWS EC2, Hetzner), же shared hosting.
2. **СУБД орнотуу** — PostgreSQL, MySQL ж.б.
3. **Виртуалдык чөйрө түзүү** — Python пакеттерин башкаруу (venv).
4. **Сервердик WSGI колдонуу** — Gunicorn же uWSGI.
5. **Веб-сервер орнотуу** — Nginx же Apache.
6. **Статикалык жана медиа файлдарды тейлөө** — collectstatic.
7. **Коопсуздук жана SSL сертификат** — HTTPS аркылуу кирүү.

2. Сервер түрлөрү

Django долбоорун жайгаштыруу үчүн ар кандай серверлер колдонулат:

Сервер	Кыскача баяндоо
VPS (Virtual Private Server)	Толук башкарууга мүмкүнчүлүк, Linux орнотулган, Ubuntu көп колдонулат
PaaS (Platform as a Service)	Мисалы, Heroku, PythonAnywhere – жеңил жана тез деплойлоо, серверди башкаруунун кереги жок
Shared Hosting	Азыр көп колдонулбайт, бирок арзан вариант

Студенттер үчүн эң жөнөкөй – **Heroku же PythonAnywhere**.

3. Деплойлоодо керектүү компоненттер

1. **Python жана Django** – серверде орнотулушу керек

2. **WSGI сервер** – веб-сервер жана Django ортосунда көпүрө (мисалы: Gunicorn)
3. **Веб-сервер** – Apache же Nginx, статикалык файлдарды берүү үчүн
4. **Маалымат базасы** – SQLite же PostgreSQL (продакшн үчүн PostgreSQL сунушталат)
5. **Virtual Environment (venv)** – долбоорду изоляциялоо үчүн

4. Деплойлоо кадамдары (Ubuntu сервер мисалы)

4.1 Серверге туташуу

```
ssh username@server_ip
```

4.2 Python жана virtual environment орнотуу

```
sudo apt update
```

```
sudo apt install python3-pip python3-venv
```

```
python3 -m venv myenv
```

```
source myenv/bin/activate
```

4.3 Django долбоорун серверге көчүрүү

- Git колдонсо болот:

```
git clone https://github.com/username/project.git
```

```
cd project
```

- Же FTP аркылуу файлдарды көчүрүүгө болот.

4.4 Пакеттерди орнотуу

```
pip install -r requirements.txt
```

requirements.txt – долбоор үчүн керектүү бардык пакеттердин тизмеси.

4.5 Gunicorn орнотуу жана тестирлөө

```
pip install gunicorn
```

```
gunicorn projectname.wsgi
```

- Серверде веб-сайт иштеп жатканын текшерүү.

4.6 Статикалык файлдарды башкаруу

Django’до CSS, JavaScript жана сүрөттөр **статикалык файлдар** деп аталат. Продакшндо аларды сервер аркылуу берүү керек.

```
python manage.py collectstatic
```

- Бул команда бардык статикалык файлдарды STATIC_ROOT папкасына көчүрөт.

4.7 Nginx менен Gunicorn конфигурациялоо

- Nginx сервер аркылуу интернет колдонуучуларга сайт жеткиликтүү болот:

```
server {
```

```
    server_name example.com;
```

```
location /static/ {  
    alias /home/username/project/static/  
}
```

```
location / {  
    proxy_pass http://127.0.0.1:8000;  
}  
}
```

- Nginx статикалык файлдарды берет, Gunicorn динамикалык Django view'ларды иштетет.

4.8 Серверди автоматтык баштоо

- Systemd файл түзүп Gunicorn серверди автоматтык иштетүүгө болот:

```
sudo nano /etc/systemd/system/projectname.service
```

```
[Unit]
```

```
Description=Gunicorn Django Project
```

```
After=network.target
```

```
[Service]
```

```
User=username
```

```
Group=www-data
```

```
WorkingDirectory=/home/username/project
```

```
ExecStart=/home/username/myenv/bin/gunicorn --access-logfile - --workers 3  
projectname.wsgi:application
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
sudo systemctl start projectname
```

```
sudo systemctl enable projectname
```

5. Heroku аркылуу жеңил деплойлоо (студенттер үчүн ыңгайлуу)

1. Heroku аккаунт түзүү: <https://heroku.com>
2. Heroku CLI орнотуу:

```
curl https://cli-assets.heroku.com/install.sh | sh
```

3. Проектти Git репозиторийге кошуу:

```
git init
```

git add .

git commit -m "Initial commit"

4. Heroku app түзүү:

heroku create my-student-app

git push heroku main

5. Миграцияларды колдонуу:

heroku run python manage.py migrate

6. Сайтты ачуу:

heroku open

6. Практикалык кеңештер

- Продакшндо **DEBUG = False** кылуу керек (settings.py)
- Сырсөздөрдү жана сырткы ачкычтарды **environment variables** аркылуу сактоо
- HTTPS сертификат орнотуу (Let's Encrypt)
- SQLite ордуна **PostgreSQL** колдонуу сунушталат

7. Практикалык тапшырмалар

1. Локалдык Django долбооруңузду Heroku же PythonAnywhere серверине деплой кылыңыз.
2. Статикалык файлдарды туура көрсөтүңүз (CSS, сүрөттөр).
3. Админ панелге кирип, жаңы студент кошуп, серверде текшериңиз.
4. DEBUG режимин өчүрүп, HTTPS аркылуу сайтка кирүүнү текшериңиз.

8. Жыйынтык

- Django деплойлоо – веб-сайтты интернет колдонуучуларга жеткиликтүү кылган процесс.
- Веб-сайт **Gunicorn/Nginx** же **Heroku/PythonAnywhere** аркылуу иштей алат.
- Деплойлоодо **коопсуздук, статикалык файлдар, маалымат базасы** жана **суперколдонуучу** маанилүү.
- Практикалык кадамдар менен студенттер реалдуу интернетте иштеген веб-сайт түзүүнү үйрөнөт.



<https://forms.gle/MC7dr7A1VJeKi8FY6>

V БӨЛҮМ. ЖЫЙЫНТЫК ЖАНА СУНУШТАР

5.1 Жыйынтыктык корутунду

Бул окуу куралы Python программалоо тилинин негизинде веб-тиркемелерди иштеп чыгуу багытында баштапкы жана орто деңгээлдеги билим алуучулар үчүн методикалык негиз түзүү максатын көздөдү. Материалды түзүүдө заманбап программалык камсыздоону иштеп чыгуу талаптары, билим берүү стандарттары жана практикалык багыт эске алынды.

Окуу процессинде Python тилинин универсалдуу мүнөзү жана веб-иштеп чыгуудагы ролу кеңири каралып, Django жана FastAPI сыяктуу фреймворктордун функционалдык мүмкүнчүлүктөрү салыштырмалуу талданды. Айрыкча Django фреймворкуна басым жасалып, анын модель–көрүнүш–шаблон (MVT) архитектурасына негизделген иштөө логикасы, ORM механизми, административдик интерфейси жана коопсуздук мүмкүнчүлүктөрү этап-этабы менен түшүндүрүлдү.

Мындан тышкары, виртуалдык чөйрө менен иштөө, пакеттерди башкаруу, маалымат базаларына туташуу жана PostgreSQL сыяктуу өнүккөн СУБДларды колдонуу маселелери каралды. Docker технологиясына арналган кошумча бөлүмдөр студенттерге заманбап деплой жана контейнерлөө түшүнүктөрүн өздөштүрүүгө шарт түзөт.

Жыйынтыгында, бул окуу куралы Python тилинин негиздеринен тартып, толук кандуу веб-тиркемелерди иштеп чыгууга чейинки билимди системалуу түрдө берүүгө багытталган деп айтууга болот. Материал теория менен практиканын айкалышына негизделип, студенттердин өз алдынча иштөө жөндөмдүүлүгүн жана практикалык көндүмдөрүн өнүктүрүүгө шарт түзөт.

Окуу курсунун алкагында төмөнкү логикалык иштөө тизмеги өздөштүрүлдү:

Python → SQLite → Django

- *Python* — негизги программалоо тили катары колдонулуп, алгоритмдик ой жүгүртүүнү калыптандырат;
- *SQLite* — жеңил салмактуу маалымат базасы катары окутуунун баштапкы этаптарында маалыматтар менен иштөөнү түшүндүрүү үчүн пайдаланылат;
- *Django* — веб-тиркемелердин сервердик бөлүгүн ишке ашырууда, маалымат базалары жана REST API менен иштөөдө колдонулат.

Бул ыкма студенттерге программалык чечимдерди комплекстүү түрдө иштеп чыгуу көндүмүн калыптандырууга өбөлгө түзөт.

5.2 Карьералык багыттар жана практикалык сунуштар

Алынган билимдер келечекте төмөнкү кесиптик багыттарда иштөөгө негиз боло алат:

1. Junior Python Developer

- Python тилинин негизги синтаксиси жана стандарттык китепканаларын өздөштүрүү;
- Практикалык тапшырмалар жана чакан долбоорлор аркылуу тажрыйба топтоо.

2. Backend Developer (Django)

- Django фреймворку аркылуу веб-сайттарды жана REST API иштеп чыгуу;
- Маалымат базалары жана ORM технологиялары менен иштөө;
- Серверге жайгаштыруу жана контейнерлөө негиздерин билүү.

3. Маалыматтарды автоматташтыруу адиси

- Python тили аркылуу маалыматтарды жыйноо, иштетүү жана талдоо;
- Pandas, NumPy жана файл форматтары менен иштөө көндүмдөрү.

4. Портфолио үчүн практикалык долбоорлор

- Веб-платформа же блог системасы;
- Маалыматтарды талдоо жана визуалдаштыруу долбоорлору;
- Автоматташтыруу багытындагы пайдалуу скрипттер.

Практикалык сунуш катары студенттерге аткарылган долбоорлорун GitHub платформасына жайгаштыруу, портфолио түзүү жана резюмеин сапатына өзгөчө көңүл буруу сунушталат.

Мындан тышкары, веб-иштеп чыгуу тармагында төмөнкү карьералык жолдор мүмкүн: backend же full-stack иштеп чыгуучу, API адиси, DevOps инженери, маалымат инженери, машиналык окутуу боюнча адис, коопсуздук инженери жана техникалык архитектор.

Ийгиликтүү адис болуу үчүн үзгүлтүксүз билим алуу, практика жүзүндө иштөө, командада иштөө жөндөмүн өнүктүрүү, чет тилдерди өздөштүрүү жана IT-коомчулукка активдүү катышуу маанилүү экендиги белгиленет.

5.3 Кошумча билим берүү ресурстары жана сунуш кылынган булактар

Бул бөлүмдө студенттердин өз алдынча билим алуусун колдоо, практикалык көндүмдөрүн тереңдетүү жана заманбап веб-иштеп чыгуу технологияларын үзгүлтүксүз өздөштүрүү максатында сунуш кылынган негизги маалымат булактары келтирилет. Тизмектелген ресурстар окуу курсунун мазмунуна шайкеш келип, теорияны практика менен айкалыштырууга өбөлгө түзөт.

1. Расмий техникалык документациялар

Расмий документациялар программалоо тилдерин жана фреймворкторду терең жана так өздөштүрүүнүн негизги булагы болуп саналат. Алар акыркы жаңыртууларды, стандарттарды жана мыкты тажрыйбаларды камтыйт.

- Python программалоо тилинин расмий документациясы — тилдин синтаксиси, стандарттык китепканалары жана колдонуу мисалдары берилет;
- SQLite маалымат базасы боюнча расмий колдонмолор — SQL командалары жана файлдык база менен иштөө принциптери;
- Django веб-фреймворкунун расмий документациясы — архитектурасы, ORM, коопсуздук жана админ-панель боюнча кеңири маалымат;
- Django REST Framework — RESTful сервистерди иштеп чыгуу ыкмалары;
- Кошумча түрдө, окуу процессинде каралган болсо, PostgreSQL, Docker, Pydantic, SQLAlchemy же Tortoise ORM боюнча расмий булактарды колдонуу сунушталат.

Бул документациялар студенттерге профессионалдык иштеп чыгууда зарыл болгон көндүмдөрдү калыптандырууга жардам берет.

2. Видео сабактар жана билим берүүчү каналдар

Видео форматындагы материалдар практикалык мисалдар аркылуу теманы тез кабыл алууга шарт түзөт. Төмөнкү каналдар билим берүү чөйрөсүндө кеңири таанылган:

- Python жана Django боюнча системалуу сабактарды сунуштаган адистештирилген каналдар;
- Жалпы программалоо логикусын жана практикалык долбоорлорду түшүндүргөн авторлор;
- Баштапкы деңгээлдегилер үчүн түшүнүктүү, этап-этабы менен баяндалган видеосабактар.

Бул ресурстар аудиториядан тышкары өз алдынча окууда пайдалуу.

3. Практикалык көндүмдөрдү өнүктүрүүчү онлайн платформалар

Программалоо көндүмдөрүн бекемдөө үчүн теория менен катар практикалык тапшырмаларды аткаруу өзгөчө маанилүү. Бул багытта төмөнкү платформалар сунушталат:

- Алгоритмдик ой жүгүртүүнү өнүктүрүүгө багытталган интерактивдүү сайттар;
- Реалдуу программалоо тапшырмалары аркылуу тажрыйба топтоого мүмкүнчүлүк берген ресурстар;
- Python жана веб-иштеп чыгуу боюнча практикалык мисалдарга бай билим берүү порталдары.

Бул платформалар студенттердин логикалык ой жүгүртүүсүн жана код жазуу ыкмаларын өркүндөтөт.

4. Онлайн курстар жана билим берүү платформалары

Заманбап онлайн курстар программалоону өз алдынча терең үйрөнүүгө ыңгайлуу шарт түзөт. Алардын арасында:

- Университеттер жана эл аралык уюмдар тарабынан сунушталган академиялык курстар;
- Практикага багытталган видеокурстар жана интерактивдүү сабактар;
- Python, Django жана веб-технологиялар боюнча адистештирилген платформалар.

Бул курстар кошумча билим алуу жана кесиптик деңгээлди жогорулатуу үчүн пайдалуу.

5. IT-коомчулуктар жана кесиптик форумдар

Программалоо тармагында суроо-жооп, тажрыйба алмашуу жана командалык иш көндүмдөрүн өнүктүрүү үчүн IT-коомчулуктардын ролу чоң. Студенттерге:

- Эл аралык программалоо форумдарына катышуу;
- Ачык булактуу долбоорлордун талкууларын байкоо жана салым кошуу;
- Адистештирилген чат-платформалар аркылуу тажрыйбалуу иштеп чыгуучулар менен байланыш түзүү сунушталат.

6. Окуу адабияттары жана илимий булактар

Теориялык базаны бекемдөө үчүн Python жана веб-иштеп чыгууга арналган заманбап окуу китептерин колдонуу маанилүү. Бул китептерде программалоонун негиздери, фреймворктор менен иштөө жана мыкты тажрыйбалар системалуу берилет.

7. Блогдор жана аналитикалык макалалар

Технологиялык компаниялардын жана жеке адистердин блогдору заманбап тенденциялар, практикалык кейстер жана реалдуу долбоорлор боюнча актуалдуу маалыматтарды берет. Алар студенттердин аналитикалык ой жүгүртүүсүн өнүктүрүүгө жардам берет.

Жыйынтык сунуш:

Берилген ресурстар тизмеси окуу курсунун мазмунуна ылайык түзүлгөн. Окутуучу бул булактарды окуу программасына жараша кеңейтип же кыскартып, студенттер үчүн эң актуалдуу деп эсептелгендерин тандай алат. Ар бир ресурска кыскача мүнөздөмө берүү студенттердин өз алдынча окуусун дагы да натыйжалуу кылат.

Адабияттар

1. Бекмамбетов, А. «Структуралык программалоо жана алгоритмдер», Бишкек, 2020
2. Сыдык уулу, Ж. «Python программалоо тили боюнча практикалык колдонмо», Бишкек, 2021
3. Турсунбаев, М. «Компьютердик программалоо негиздери», Бишкек, 2019
4. Асанов, К. «Маалыматтар базасы жана SQL тили», Бишкек, 2020
5. Мамбеталиев, Д. «Python тили менен маалыматтар базасы жана веб-технологиялар», Бишкек, 2022
6. Жакшылык уулу, С. «Программалоодо структуралык принциптер», Бишкек, 2021
7. Кыргыз Республикасынын Билим берүү жана Илим министрлигинин окуу программалары, Бишкек, 2023
8. Онлайн ресурстар:
 - Python тили боюнча кыргызча колдонмо — <https://python.kg>
 - SQLite жана SQL боюнча кыргызча маалымат — <https://sqlite.kg>
9. Любановский, Д. «Python: программирование для начинающих», Москва, 2021
10. Лутц, М. «Learning Python», 5th Edition, O'Reilly, 2019
11. Sweigart, A. «Automate the Boring Stuff with Python», 2nd Edition, No Starch Press, 2019
12. Beazley, D. «Python Cookbook», 3rd Edition, O'Reilly, 2013
13. Бекмамбетов, А. «Структуралык программалоо жана алгоритмдер», Бишкек, 2020
14. Мартин, Р. «Clean Code: A Handbook of Agile Software Craftsmanship», 2008
15. Django Project Documentation — <https://docs.djangoproject.com/en/4.2/>
16. Python Documentation — <https://docs.python.org/3/>
17. SQLite Documentation — <https://www.sqlite.org/docs.html>
18. PEP 8 — Python коддоо стандарттары: <https://peps.python.org/pep-0008/>