Съдържание

[1. Увод 4](#_Toc196701637)

[2. Основни понятия и принципи в уеб сигурността 6](#_Toc196701638)

[2.1. Въведение в уеб сигурността 6](#_Toc196701639)

[2.2. Удостоверяване (Authentication) и оторизация (Authorization) 6](#_Toc196701640)

[2.3. Значение на сигурността в съвременните уеб приложения 7](#_Toc196701641)

[3. Удостоверяване на самоличността (Authentication) 7](#_Toc196701642)

[3.1. Основни подходи за удостоверяване 7](#_Toc196701643)

[3.1.1 Еднофакторно удостоверяване (Single-Factor Authentication) 7](#_Toc196701644)

[3.1.2 Двуфакторно удостоверяване (Two-Factor Authentication, 2FA) 8](#_Toc196701645)

[3.1.3 Многофакторно удостоверяване (Multi-Factor Authentication, MFA) 8](#_Toc196701646)

[3.2. Механизми и протоколи за удостоверяване 8](#_Toc196701647)

[3.2.1 Потребителско име и парола 8](#_Toc196701648)

[3.2.2 Сертификати 8](#_Toc196701649)

[3.2.3 OAuth и SSO (Single Sign-On) 8](#_Toc196701650)

[3.3. Типични уязвимости и добри практики 9](#_Toc196701651)

[3.3.1 Уязвимости 9](#_Toc196701652)

[3.3.2 Добри практики 9](#_Toc196701653)

[4. Оторизация (Authorization) 10](#_Toc196701654)

[4.1. Понятие за роля, право и политика за достъп 10](#_Toc196701655)

[4.2. Видове оторизация 10](#_Toc196701656)

[4.2.1 Оторизация по роли (Role-Based Access Control – RBAC) 10](#_Toc196701657)

[4.2.2 Оторизация по атрибути (Attribute-Based Access Control – ABAC) 11](#_Toc196701658)

[4.2.3 Примери за различни нива на достъп 11](#_Toc196701659)

[5. Архитектурни модели и стандарти в уеб удостоверяването 11](#_Toc196701660)

[5.1. Сесии 11](#_Toc196701661)

[5.2. Бисквитки (Cookies) 11](#_Toc196701662)

[5.3. Токени (JWT – JSON Web Token) 11](#_Toc196701663)

[5.4. Инструменти и библиотеки: сравнение между различни подходи и технологии 12](#_Toc196701664)

[5.4.1 Django инструменти 12](#_Toc196701665)

[5.4.2 Сравнение с други технологии 12](#_Toc196701666)

[6. Средства за удостоверяване и оторизация в Django 13](#_Toc196701667)

[6.1. Вградена автентикация в Django (django.contrib.auth) 14](#_Toc196701668)

[6.2. Моделът на потребителя, групи и права 14](#_Toc196701669)

[6.2.1 Моделът User и профилът на потребителя 14](#_Toc196701670)

[6.2.2 Групи и права (permissions) 15](#_Toc196701671)

[6.3. Готови изгледи (views) и форми за логин, регистрация, смяна на парола 15](#_Toc196701672)

[6.3.1 Форми за регистрация и вход 15](#_Toc196701673)

[6.3.2 Изгледи (views) за аутентификация 16](#_Toc196701674)

[6.3.3 Използване на маршрутите 17](#_Toc196701675)

[6.4. Разширяване и персонализация на логиката за управление на достъпа 17](#_Toc196701676)

[7. Обща архитектура и постановка на разработеното приложение 19](#_Toc196701677)

[7.1. Цели и структура на приложението 19](#_Toc196701678)

[7.2. Основни файлове и тяхното предназначение 19](#_Toc196701679)

[8. Реализация на моделите за потребителски данни 20](#_Toc196701680)

[8.1. Описание на използвания User модел 20](#_Toc196701681)

[8.2. Групи, роли и права за достъп 20](#_Toc196701682)

[8.3. Административни инструменти за управление 21](#_Toc196701683)

[9. Реализация на формите за регистрация и вход 21](#_Toc196701684)

[9.1. Видове форми и тяхната роля 21](#_Toc196701685)

[9.2. Примерни реализации 21](#_Toc196701686)

[9.3. Интерфейс и съобщения 22](#_Toc196701687)

[10. Реализация на изгледите за удостоверяване и управление на профил 22](#_Toc196701688)

[10.1. Основни изгледи (views.py) 22](#_Toc196701689)

[10.2. Сесии, пренасочвания и съобщения 23](#_Toc196701690)

[11. Оторизация и управление на достъпа 23](#_Toc196701691)

[11.1. Ограничаване на достъпа – декоратори и проверки 23](#_Toc196701692)

[11.2. Интерфейсни ограничения според ролята 24](#_Toc196701693)

[12. Представяне на потребителския интерфейс 24](#_Toc196701694)

[12.1. Интерфейс – изгледи и интеракции 24](#_Toc196701695)

[12.2. Обработка на състояния и съобщения 25](#_Toc196701696)

[13. Тестване и демонстрация 25](#_Toc196701697)

[13.1. Оценка и препоръки 25](#_Toc196701698)

[14. Заключение 26](#_Toc196701699)

[14.1. Обобщение на основните резултати 26](#_Toc196701700)

[14.2. Анализ на постигнатите цели 27](#_Toc196701701)

[14.3. Ограничения и предизвикателства 27](#_Toc196701702)

[14.4. Възможности за бъдещо развитие 28](#_Toc196701703)

[14.5. Практически препоръки 28](#_Toc196701704)

[14.6. Заключителни думи 29](#_Toc196701705)

[15. Литература: 30](#_Toc196701706)

# Увод

В последните десетилетия уеб технологиите се превърнаха в неизменна част от ежедневието на съвременния човек и бизнеса. Интернет пространството предлага безпрецедентни възможности за обмен на информация, електронна търговия, социални контакти и предоставяне на услуги. С нарастването на броя и сложността на уеб приложенията, въпросите, свързани със сигурността, стават все по-актуални и критични. Защитата на личните данни, предотвратяването на неоторизиран достъп и гарантирането на надеждност и доверие са сред основните предизвикателства пред всеки разработчик и собственик на уеб платформа.

В основата на уеб сигурността стоят два ключови процеса – удостоверяване (authentication) и оторизация (authorization). Удостоверяването представлява процесът, чрез който системата проверява самоличността на потребителя, докато оторизацията определя какви права и достъп има вече удостовереният потребител. Тези два процеса са тясно свързани и са в основата на изграждането на сигурни уеб приложения, които защитават както потребителите, така и самата система от злоупотреби и атаки.

С нарастването на изискванията към сигурността, съвременните уеб приложения трябва да отговарят на редица стандарти и добри практики. Това включва не само използването на силни пароли и криптиране на комуникацията, но и внедряването на многофакторна автентикация, управление на сесии, контрол на достъпа до ресурси, както и защита срещу типични уязвимости като SQL инжекции, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) и други. В допълнение, законодателни изисквания като Общия регламент за защита на личните данни (GDPR) налагат още по-високи стандарти за съхранение и обработка на лична информация.

В този контекст изборът на подходяща технологична платформа за разработка на уеб приложения е от съществено значение. Django – един от най-популярните уеб фреймуъркове за Python – се отличава със своята богата функционалност, сигурност и лесна разширяемост. Django предоставя вградена система за удостоверяване и оторизация, която включва модели за потребители, групи, права, както и готови изгледи и форми за обработка на автентикацията. Това позволява на разработчиците да изграждат сигурни приложения с минимални усилия, като същевременно имат възможност за персонализация и разширяване на логиката според специфичните нужди на проекта.

Настоящата дипломна работа има за цел да изследва и анализира основните принципи и средства за уеб сигурност, с фокус върху процесите на удостоверяване и оторизация в контекста на Django. В теоретичната част са разгледани фундаменталните понятия, архитектурните модели и стандартите, които определят съвременните практики в областта на уеб сигурността. Описани са различните подходи за удостоверяване на самоличността – от еднофакторно до многофакторно удостоверяване, както и механизмите за контрол на достъпа чрез роли, права и политики за достъп.

В практическата част е реализирано уеб приложение, което демонстрира процесите на регистрация, вход, управление на профил и контрол на достъпа до ресурси. Използвани са основните компоненти на Django – модели, форми, изгледи, маршрутизация и административен панел. Особено внимание е отделено на разширяването и персонализацията на логиката за управление на достъпа, както и на интеграцията на добри практики за сигурност. Чрез реални примери и кодови фрагменти са илюстрирани основните стъпки по изграждане на сигурна система за удостоверяване и оторизация.

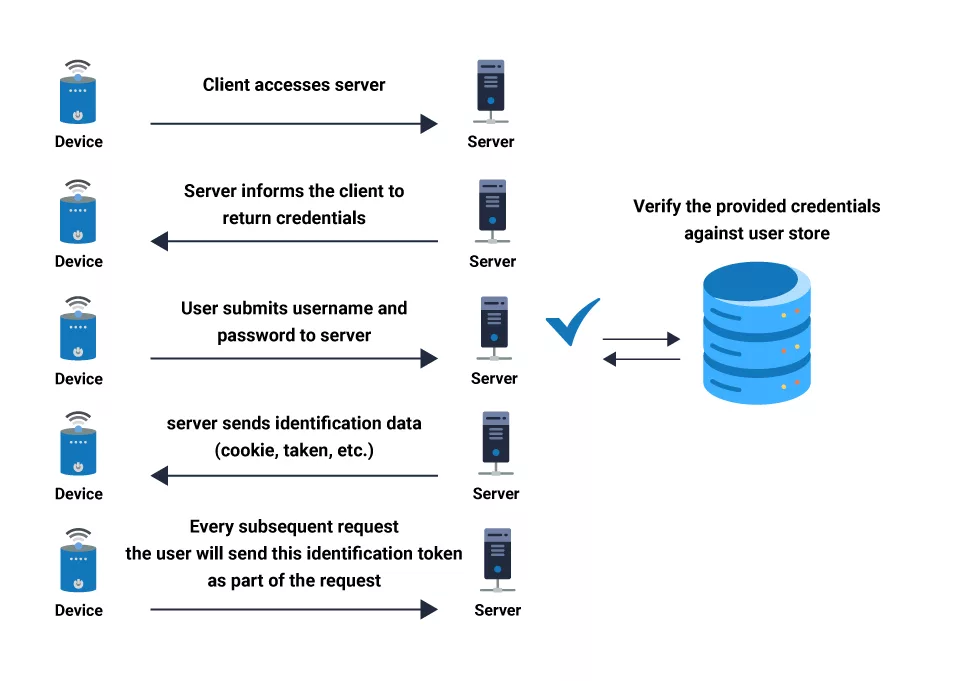
В заключение, темата за уеб сигурността и управлението на достъпа е изключително актуална и многопластова. С развитието на технологиите и нарастването на киберзаплахите, необходимостта от надеждни и сигурни решения става все по-голяма. Чрез анализа на теоретичните основи и практическата реализация с помощта на Django, настоящата работа цели да даде цялостен поглед върху процесите на удостоверяване и оторизация, както и да предложи практически насоки за изграждане на сигурни уеб приложения.

# Основни понятия и принципи в уеб сигурността

## Въведение в уеб сигурността

Уеб сигурността е съвкупност от технологии, процеси и практики, които целят да защитят уеб приложенията и техните потребители от неоторизиран достъп, злоупотреби и атаки. В съвременния дигитален свят, където голяма част от личната и бизнес информация се съхранява и обработва онлайн, осигуряването на сигурност е от ключово значение за доверието на потребителите и за спазването на законовите изисквания.

## Удостоверяване (Authentication) и оторизация (Authorization)



**Удостоверяване** (authentication) е процесът, чрез който системата проверява самоличността на потребителя. Най-често това става чрез въвеждане на потребителско име и парола, но могат да се използват и други методи като биометрични данни, еднократни кодове и др. Удостоверяването отговаря на въпроса: „Кой си ти?“.

**Оторизация** (authorization) е процесът, чрез който системата определя какви права и достъп има вече удостовереният потребител. Оторизацията отговаря на въпроса: „Какво може да прави този потребител?“. Например, след като даден потребител се е идентифицирал успешно, системата може да му позволи достъп само до определени ресурси или функционалности.

## Значение на сигурността в съвременните уеб приложения

Сигурността е критичен аспект на всяко уеб приложение поради следните причини:

* **Защита на лични данни:** Нарушаването на сигурността може да доведе до изтичане на чувствителна информация.
* **Спазване на законови изисквания:** Регулации като GDPR изискват защита на личните данни.
* **Доверие на потребителите:** Сигурните приложения печелят доверието на клиентите и партньорите.
* **Предотвратяване на финансови загуби:** Атаките могат да доведат до директни или индиректни финансови щети.

# Удостоверяване на самоличността (Authentication)

## Основни подходи за удостоверяване

### Еднофакторно удостоверяване (Single-Factor Authentication)

Това е най-разпространеният метод, при който потребителят предоставя само един фактор за удостоверяване – обикновено потребителско име и парола. Този подход е лесен за реализация, но не е достатъчно сигурен при самостоятелна употреба.

**Примерен код (Django):**

from django.contrib.auth import authenticate, login

def login\_view(request):

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

### Двуфакторно удостоверяване (Two-Factor Authentication, 2FA)

Този метод изисква два различни фактора за удостоверяване, например:

* Нещо, което потребителят знае (парола)
* Нещо, което потребителят има (код, изпратен по SMS или генериран от приложение)

2FA значително повишава сигурността, тъй като компрометирането на един фактор не е достатъчно за достъп.

### Многофакторно удостоверяване (Multi-Factor Authentication, MFA)

Този подход използва повече от два фактора, като може да включва биометрични данни (пръстов отпечатък, лицево разпознаване) или хардуерни токени.

## Механизми и протоколи за удостоверяване

### Потребителско име и парола

Най-често използваният механизъм. Важно е паролите да се съхраняват хеширани и със сол (salt), за да се предотврати тяхното разкриване при пробив.

**Примерен код (Django):**

from django.contrib.auth.models import User

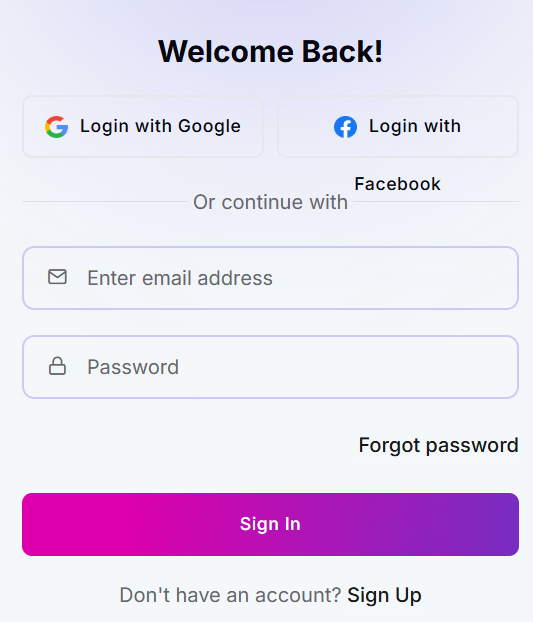
user = User.objects.create\_user('username', password='password')

### Сертификати

Използват се в по-сигурни среди, където удостоверяването се извършва чрез цифрови сертификати (X.509), издадени от доверен орган.

### OAuth и SSO (Single Sign-On)

OAuth е протокол, който позволява на потребителите да се удостоверяват чрез външни доставчици (Google, Facebook и др.), без да споделят паролата си с приложението. SSO позволява достъп до множество приложения с еднократно удостоверяване.



## Типични уязвимости и добри практики

### Уязвимости

* **Слаби пароли:** Лесни за отгатване или пробиване чрез brute-force атаки.
* **SQL инжекции:** Ако входните данни не се валидират правилно.
* **Фишинг:** Потребителите могат да бъдат подмамени да въведат данните си на фалшив сайт.
* **Съхранение на пароли в чист текст:** Това е сериозна грешка в сигурността.

### Добри практики

* Използване на силни пароли и изискване за минимална дължина и сложност.
* Хеширане на паролите със съвременни алгоритми (bcrypt, Argon2).
* Ограничаване на броя неуспешни опити за вход.
* Използване на HTTPS за криптиране на трафика.
* Използване на двуфакторна автентикация.

# Оторизация (Authorization)

## Понятие за роля, право и политика за достъп

**Роля** (role) е логическо групиране на права, което определя какви действия може да извършва даден потребител.

**Право** (permission) е конкретно разрешение за достъп до определен ресурс или функционалност.

**Политика за достъп** (access policy) е набор от правила, които определят кой има достъп до какво.

User

│

├────-> Група: "Потребител"

│ │

│ ├─ Право: виж свои данни

│

└────-> Група: "Администратор"

│

├─ Право: виж, редактирай всички данни

## Видове оторизация

### Оторизация по роли (Role-Based Access Control – RBAC)

При този модел на всеки потребител се присвоява една или повече роли, а всяка роля има определени права. Това е най-често използваният модел в уеб приложения.

**Примерен код (Django):**

from django.contrib.auth.decorators import permission\_required

@permission\_required('app.change\_menuitem')

def edit\_menu\_item(request, item\_id):

# Само потребители с това право могат да редактират

pass

### Оторизация по атрибути (Attribute-Based Access Control – ABAC)

Този модел използва атрибути на потребителя, ресурса и средата за вземане на решения за достъп. Например, достъпът може да зависи от часа на деня, местоположението или други характеристики.

### Примери за различни нива на достъп

* **Гост:** Може да разглежда публична информация.
* **Регистриран потребител:** Може да редактира собствения си профил.
* **Администратор:** Има достъп до административния панел и всички данни.

# Архитектурни модели и стандарти в уеб удостоверяването

## Сесии

Сесиите са механизъм за съхраняване на състояние между отделните заявки на потребителя. В Django сесиите се реализират чрез уникален идентификатор, който се съхранява в бисквитка на браузъра.

**Примерен код (Django):**

# Записване на данни в сесията

request.session['user\_id'] = user.id

## Бисквитки (Cookies)

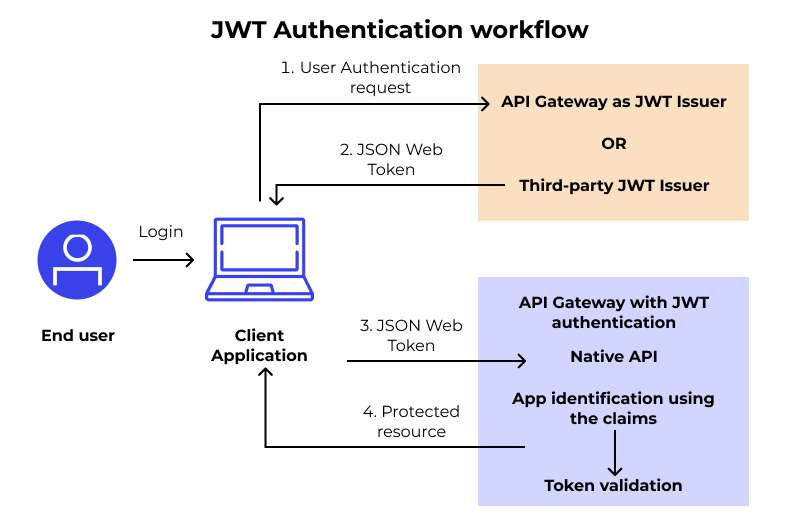
Бисквитките са малки файлове, които се съхраняват на компютъра на потребителя и се използват за съхраняване на информация като идентификатори на сесии, предпочитания и др. Те могат да бъдат обект на атаки като кражба на сесия (session hijacking), затова трябва да се използват сигурни бисквитки (HttpOnly, Secure).

## Токени (JWT – JSON Web Token)

JWT е модерен стандарт за удостоверяване и оторизация, при който информацията за потребителя се съхранява в криптиран токен, който се предава между клиента и сървъра. JWT се използва често при SPA (Single Page Applications) и мобилни приложения.

**Примерен JWT:**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...



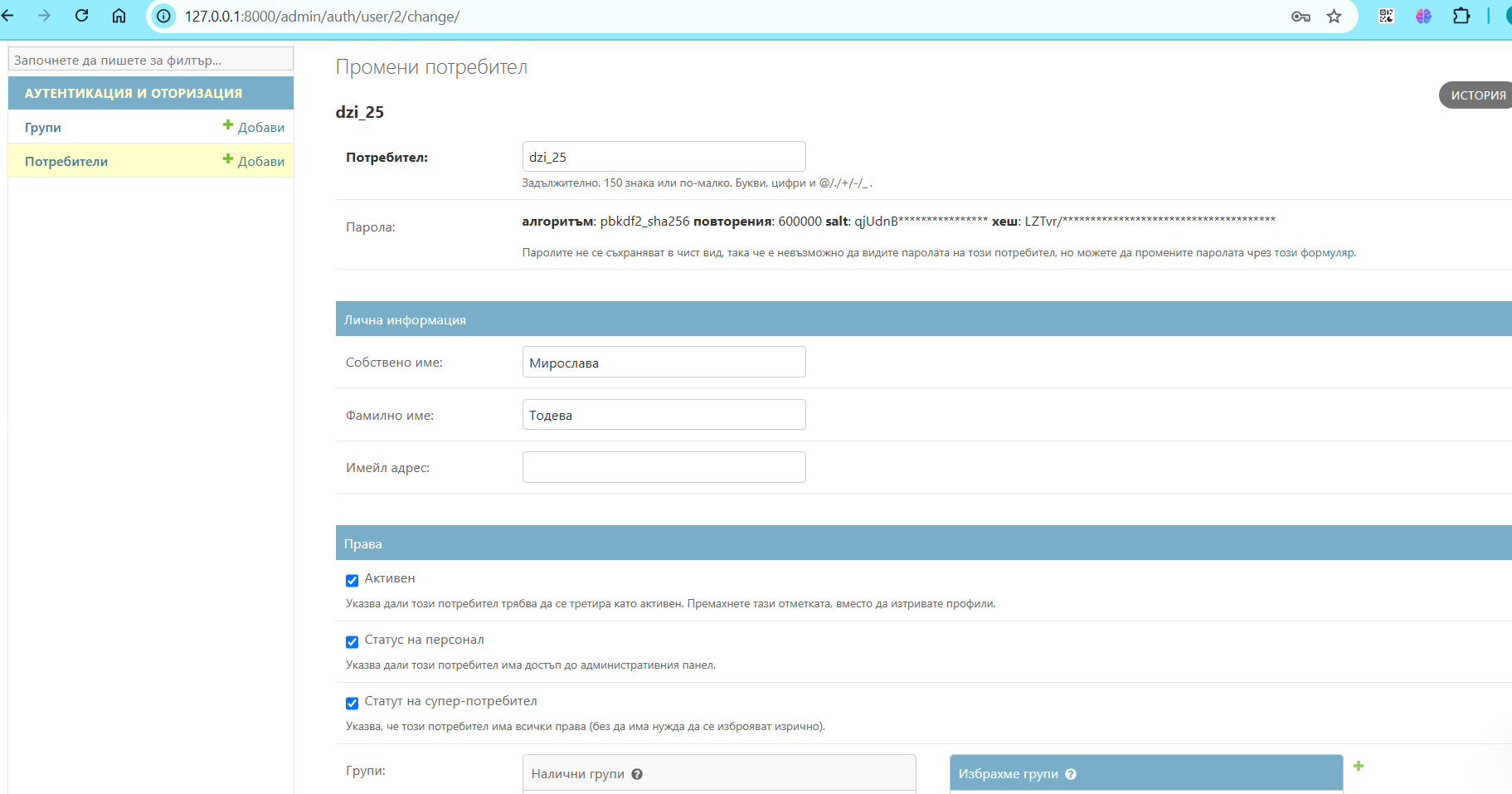
## Инструменти и библиотеки: сравнение между различни подходи и технологии

### Django инструменти

* **django.contrib.auth:** Вградена система за удостоверяване и оторизация, включваща модели за потребители, групи и права.
* **django-allauth:** Разширение за социално удостоверяване (Google, Facebook и др.).
* **django-rest-framework-simplejwt:** Библиотека за работа с JWT в Django REST Framework.

### Сравнение с други технологии

* **Flask:** По-лек фреймуърк, изисква допълнителни разширения за удостоверяване и оторизация.
* **Express.js (Node.js):** Използва middleware за удостоверяване, често с JWT.
* **Spring Security (Java):** Мощна система за управление на сигурността в Java приложения.



В този раздел бяха разгледани основните понятия и принципи на уеб сигурността, подходите за удостоверяване и оторизация, архитектурните модели и инструментите за тяхната реализация. Включени са практически примери с Django, които могат да бъдат илюстрирани със снимки на екрани (login форма, административен панел, различни менюта според ролята) и диаграми, показващи потока на удостоверяване и оторизация. Тези знания са фундаментални за изграждането на сигурни и надеждни уеб приложения.

# Средства за удостоверяване и оторизация в Django

Django e един от най-популярните уеб фреймуъркове за Python и предлага мощна, богата и сигурна система за удостоверяване (authentication) и оторизация (authorization), която е лесно разширима и пригодима към различни нужди на приложенията. Тази система се базира на компонентите на пакета django.contrib.auth, който осигурява модели на потребителя, групи, права, както и готови изгледи и формуляри за обработка на автентикацията. По-долу са разгледани основните средства, които Django предоставя, заедно с конкретни примери от проекта.

## Вградена автентикация в Django (django.contrib.auth)

Системата за автентикация на Django е изградена така, че да разделя логическите аспекти на удостоверяване и оторизация, като осигурява средства за:

* създаване, идентификация и валидиране на потребители,
* управление на пароли,
* присвояване на права и контрол на достъпа.

В центъра на системата стои моделът на потребителя (User), заедно с набор от помощни функции, които се използват в изгледите и формите.

* **Пример:**

В кода на приложението е използван стандартният User модел на Django, което се удостоверява с импортирането му в forms.py, models.py и други файлове:

from django.contrib.auth.models import User

Това позволява използването на всички стандартни методи и атрибути, дефинирани от Django.

## Моделът на потребителя, групи и права

Вградената реализация на User модела позволява бърза и сигурна интеграция със системата за удостоверяване:

### Моделът User и профилът на потребителя

Във файла models.py може да се види пример за разширяване на стандартния User модел чрез допълнителния модел Profile:

from django.db import models

from django.contrib.auth.models import User

class Profile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

full\_name = models.CharField(max\_length=100, blank=True)

bio = models.TextField(blank=True)

def \_\_str\_\_(self):

return self.user.username

Този подход позволява съхраняване на разширена информация за всеки потребител, без да се променя директно структурата на User.

### Групи и права (permissions)

Django осигурява два важни механизма за оторизация: групи и права. Групите са логическо обединение на потребители с едни и същи права, докато permissions дефинират конкретни действия (като добавяне, промяна, изтриване на записи).

**Управление чрез административния панел:**

С добавянето на Profile в admin.py:

from django.contrib import admin

from .models import Profile

admin.site.register(Profile)

се предоставя на администраторите възможност да създават, редактират и изтриват потребителски профили, но също така и да управляват групи и права през стандартното Admin меню.

## Готови изгледи (views) и форми за логин, регистрация, смяна на парола

Django предоставя комплект от вградени класове и функции, които реализират стандартните операции по удостоверяване – вход, изход, смяна/възстановяване на парола и регистрация.

### Форми за регистрация и вход

Във файла forms.py са реализирани:

* Регистрационна форма на базата на UserCreationForm:

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

class UserRegisterForm(UserCreationForm):

email = forms.EmailField()

class Meta:

model = User

fields = ['username', 'email', 'password1', 'password2']

* Форма за редакция на профил:

from .models import Profile

class ProfileUpdateForm(forms.ModelForm):

class Meta:

model = Profile

fields = ['full\_name', 'bio']

Тези форми се използват във view-тата за улесняване и управление на процеса по регистрация, вход и редакция.

### Изгледи (views) за аутентификация

Във views.py се откриват типичните примери за обработка на регистрация и вход:

**Регистрация на потребител:**

def register(request):

if request.method == 'POST':

user\_form = UserRegisterForm(request.POST)

if user\_form.is\_valid():

user = user\_form.save()

Profile.objects.create(user=user)

messages.success(request, f'Регистрацията е успешна.')

return redirect('login')

else:

user\_form = UserRegisterForm()

return render(request, 'register.html', {'user\_form': user\_form})

**Вход на потребител:**

from django.contrib.auth import authenticate, login

def login\_view(request):

if request.method == 'POST':

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

return redirect('profile')

else:

messages.error(request, 'Грешно потребителско име или парола.')

return render(request, 'login.html')

Използват се вградените функции authenticate и login, което гарантира сигурна обработка на данни за вход.

Форма за смяна на парола може да бъде реализирана чрез интегриране на PasswordChangeView и допълнителни шаблони, като това може да бъде разширено при нужда.

### Използване на маршрутите

Във файла urls.py са декларирани маршрутите към основните страници за register, login, logout, profile и profile-edit:

from django.urls import path

from . import views

urlpatterns = [

path('register/', views.register, name='register'),

path('login/', views.login\_view, name='login'),

path('logout/', views.logout\_view, name='logout'),

path('profile/', views.profile, name='profile'),

path('profile/edit/', views.profile\_edit, name='profile\_edit'),

]

Това улеснява достъпа до основните стъпки свързани с удостоверяване.

## Разширяване и персонализация на логиката за управление на достъпа

Вграденият механизъм на Django позволява допълнителна персонализация чрез:

* Декоратори върху изгледите като @login\_required:

from django.contrib.auth.decorators import login\_required

@login\_required

def profile(request):

...

Този декоратор гарантира, че достъп до профила имат само идентифицирани потребители.

* Контрол на групи и права в изгледите/шаблоните:

{% if user.is\_superuser %}

<a href="{% url 'admin:index' %}">Администрация</a>

{% endif %}

Така само потребители с адм. права виждат определени менюта/бутони.

* Филтриране на права на Python ниво:

from django.contrib.auth.decorators import permission\_required

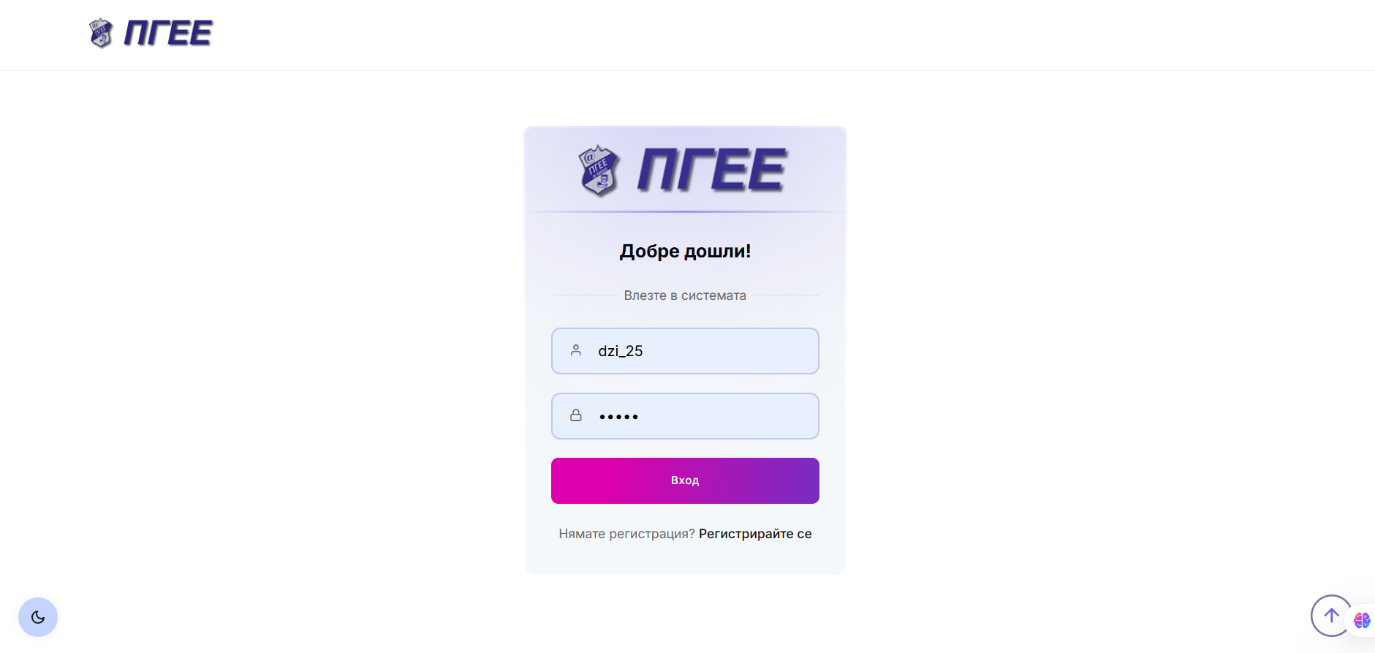
@permission\_required('app.change\_profile', raise\_exception=True)

def admin\_profile\_edit(request, user\_id):

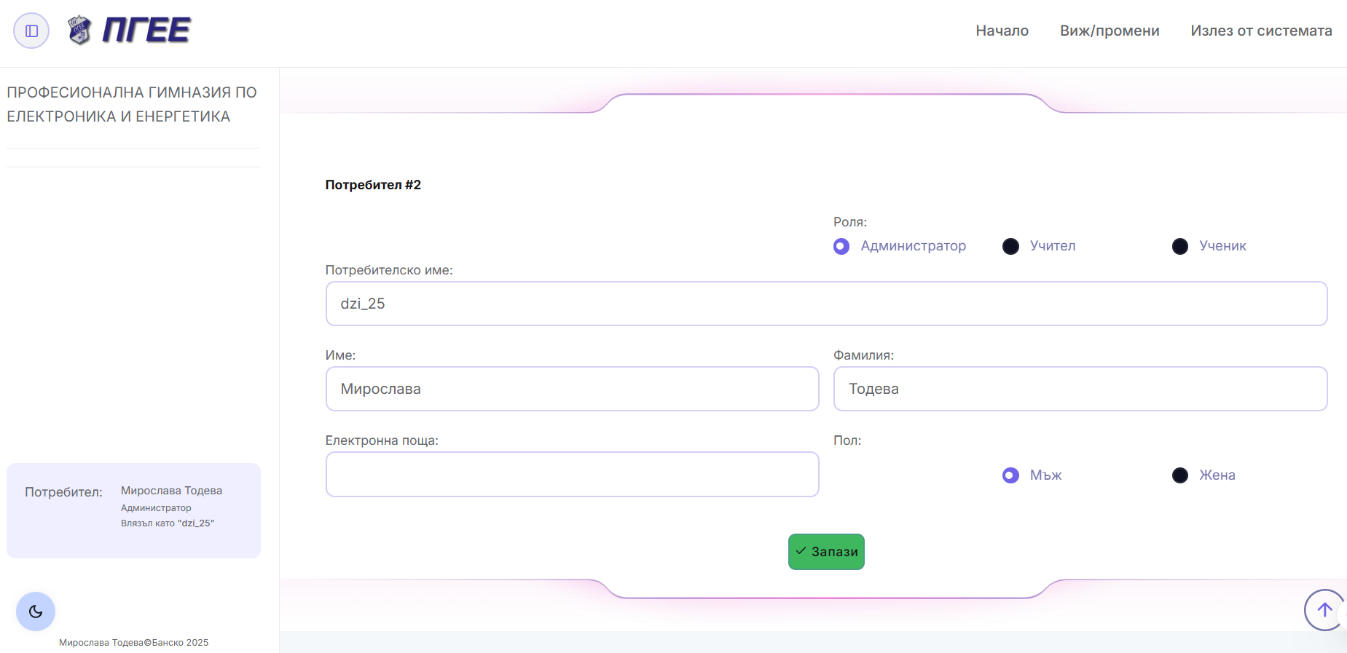
...

Това осигурява защита на чувствителни страници и данни.

**Персонализация на формите и изгледите.** Чрез унаследяване и разширяване на вградените форми и view класове, както и чрез добавяне на собствени валидации и потребителски съобщения, логиката на удостоверяване и оторизация може да се пригоди максимално според спецификите на приложението и възнасването на нови бизнес изисквания.



Вградената система за удостоверяване и оторизация в Django, с помощта на моделите, групите, правата, готовите форми и view-та, както и множеството възможности за персонализация, дава отлична основа за изграждане на сигурни уеб приложения с контролиран и диференциран достъп до ресурсите.



# Обща архитектура и постановка на разработеното приложение

## Цели и структура на приложението

Основната цел на създаденото уеб приложение е да демонстрира процесите на идентификация, регистрация и оторизация на потребители, както и управление на роли и права за достъп до ресурси. Реализацията е извършена чрез Django – популярен Python уеб фреймуърк, предоставящ готови средства за автентикация и контрол на достъпа.

Архитектурата се базира на класическата Django структура – изгледите (views), формите (forms), моделите (models), административното управление (admin), маршрутизацията (urls), и настройки на приложението (apps). Всеки от тези компоненти носи отговорност за конкретна част от логиката на приложението.

## Основни файлове и тяхното предназначение

* **models.py** – определя моделите (структурата на данните и потребителя);
* **forms.py** – съдържа формите за регистрация, вход и редакция на профил;
* **views.py** – обработва логиката за вход, регистрация, изход, управление на профила и интерфейсните действия;
* **urls.py** – описва маршрутите към отделните страници във външния интерфейс;
* **admin.py** – съдържа административна логика за управление на модели през Django admin;
* **apps.py** – дефинира конфигурацията на самото приложение.

# Реализация на моделите за потребителски данни

## Описание на използвания User модел

В настоящата реализация се използва стандартният Django потребителски модел, без създаване на custom User клас. Това позволява интеграция с вградените механизми за автентикация и управление на права.

Във файла **models.py** се намира дефиницията на един допълнителен модел, който разширява функционалността:

from django.db import models

from django.contrib.auth.models import User

class Profile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

full\_name = models.CharField(max\_length=100, blank=True)

bio = models.TextField(blank=True)

def \_\_str\_\_(self):

return self.user.username

Този модел Profile използва OneToOne връзка с основния User модел на Django и предоставя допълнителни атрибути: "пълно име" и "описание за себе си" (bio). Това улеснява разширяването на потребителските данни без да се нарушава цялостта на автентикационната система на фреймуърка.

## Групи, роли и права за достъп

В решенията на Django управлението на роли и права се извършва чрез "groups" и "permissions". Конкретни групи и права могат да бъдат създавани и управлявани през административния панел, като за wygoda се добавя следното в **admin.py**:

from django.contrib import admin

from .models import Profile

admin.site.register(Profile)

Чрез такава регистрация в admin-а могат да се преглеждат и редактират профилите на потребителите.

## Административни инструменти за управление

Django Admin предоставя възможност за създаване, редакция и изтриване както на основните потребители, така и на допълнителните профили и групи. В случай на нужда могат да се добавят персонализирани филтри, търсене и други административни функции на моделите, като се разширява конфигурацията в admin.py.

# Реализация на формите за регистрация и вход

## Видове форми и тяхната роля

В **forms.py** са реализирани три основни форми:

* Регистрационна форма за създаване на нови потребители
* Форма за вход (автентикация)
* Форма за редакция на профил

## Примерни реализации

**Регистрационна форма:**

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

from .models import Profile

class UserRegisterForm(UserCreationForm):

email = forms.EmailField()

class Meta:

model = User

fields = ['username', 'email', 'password1', 'password2']

Тази форма разширява стандартната UserCreationForm и добавя поле за имейл.

**Форма за редакция на профил:**

class ProfileUpdateForm(forms.ModelForm):

class Meta:

model = Profile

fields = ['full\_name', 'bio']

По този начин се осигурява гъвкава възможност за обновяване на данните от разширения потребителски профил.

**Валидация и обработка на данни.**

Django автоматично обработва валидацията на UserCreationForm – проверява дали паролите съвпадат, дали потребителското име е заето, и т.н. Допълнителна логика може да се добавя чрез методите clean\_...() във формите.

## Интерфейс и съобщения

В шаблоните, въз основа на тези форми, лесно се изграждат user-friendly екрани за регистрация, вход и редакция на профил, както и съобщения при грешка или успех (например с Django messages framework):

from django.contrib import messages

messages.success(request, f'Успешна регистрация!')

messages.error(request, f'Грешка при вход: ...')

# Реализация на изгледите за удостоверяване и управление на профил

## Основни изгледи (views.py)

В **views.py** са дефинирани логиката за регистрация, вход, изход и редакция на профил:

**Регистрация:**

def register(request):

if request.method == 'POST':

user\_form = UserRegisterForm(request.POST)

if user\_form.is\_valid():

user = user\_form.save()

Profile.objects.create(user=user)

messages.success(request, f'Регистрацията е успешна.')

return redirect('login')

else:

user\_form = UserRegisterForm()

return render(request, 'register.html', {'user\_form': user\_form})

**Вход:**

from django.contrib.auth import authenticate, login

def login\_view(request):

if request.method == 'POST':

username = request.POST['username']

password = request.POST['password']

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

return redirect('profile')

else:

messages.error(request, 'Грешно потребителско име или парола.')

return render(request, 'login.html')

**Редакция на профил:**

@login\_required

def profile\_edit(request):

if request.method == 'POST':

p\_form = ProfileUpdateForm(request.POST, instance=request.user.profile)

if p\_form.is\_valid():

p\_form.save()

messages.success(request, f'Профилът е обновен успешно!')

return redirect('profile')

else:

p\_form = ProfileUpdateForm(instance=request.user.profile)

return render(request, 'profile\_edit.html', {'p\_form': p\_form})

Вижда се използването на декоратора @login\_required за контрол на достъпа до потребителските профили.

## Сесии, пренасочвания и съобщения

При успешно удостоверяване на потребител, сесията бива инициирана автоматично. За събития като успешна регистрация, грешка при вход и успешна редакция на профил, се използват Django съобщения, които се извеждат в интерфейса.

# Оторизация и управление на достъпа

## Ограничаване на достъпа – декоратори и проверки

В кода на представените изгледи се използва редовно декораторът @login\_required, осигуряващ достъп само за аутентифицирани потребители. С декоратор като @permission\_required (или чрез проверки на групите и правата в Python логиката на изгледа) може да се реализира допълнителна контролираща логика:

from django.contrib.auth.decorators import permission\_required

@permission\_required('app.change\_profile', raise\_exception=True)

def admin\_profile\_edit(request, user\_id):

...

Чрез подобни решения, само потребители с определени права – например администратори – имат достъп до критични функции.

## Интерфейсни ограничения според ролята

В навигационните шаблони (или в страничния панел на профила) се проверяват групите и правата на текущия потребител, за да се показват динамично определени бутони или менюта.

**Пример:**

{% if user.is\_superuser %}

<a href="{% url 'admin:index' %}">Администрация</a>

{% endif %}

По този начин само потребители със специални права виждат и използват административните опции.

# Представяне на потребителския интерфейс

## Интерфейс – изгледи и интеракции

Потребителският интерфейс ясно разграничава публичната част (достъпна за всички) от защитената част (достъпна само след вход). След успешна регистрация или вход потребителят вижда персонализирана страница с профилна информация и навигация според ролята.

Картините на екрана (скрийншотове) могат да илюстрират:

* Формата за регистрация с валидиращи съобщения;
* Екрана за вход с грешно въведена парола;
* Профилната страница с възможност за редакция;
* Панел с административни функции (само за администратор).

## Обработка на състояния и съобщения

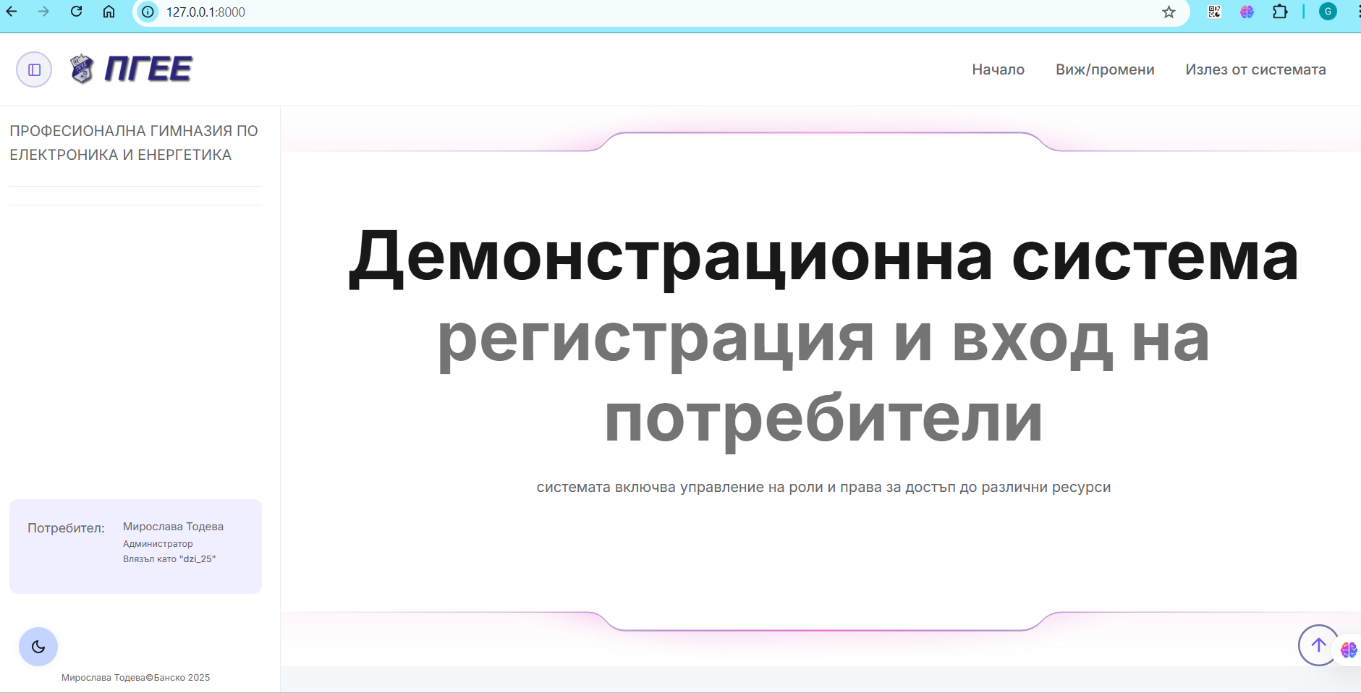
Показването на успех, грешка или предупреждение е реализирано с помощта на Django messages framework, като на всеки екран ясно се различават състоянията: регистрация (успех), вход (грешка при парола), редакция (успех).

# Тестване и демонстрация

* **12.1. Тестови сценарии**

Тестването обхваща:

* Регистрация на нов потребител (валидни и невалидни данни);
* Вход с коректна и грешна парола/потребителско име;
* Редакция на профил и проверка на запазените данни;
* Опит за достъп до защитена страница без вход и пренасочване към login.



## Оценка и препоръки

Кодът и интерфейсът покриват основните сценарии за сигурно управление на потребители. Препоръчва се използване на HTTPS, двуфакторна автентикация за по-критични приложения и регулярна проверка на кода за уязвимости.

# Заключение

В съвременната дигитална епоха уеб приложенията са неразделна част от ежедневието на хората и бизнеса. С нарастването на тяхната роля и значение, въпросите, свързани със сигурността, стават все по-актуални и критични. Настоящата дипломна работа беше посветена на един от най-важните аспекти на уеб сигурността – процесите на удостоверяване и оторизация, с фокус върху тяхната реализация в среда на Django.

## Обобщение на основните резултати

В теоретичната част на работата бяха разгледани фундаменталните понятия и принципи на уеб сигурността. Подчертано беше значението на защитата на личните данни, спазването на законовите изисквания и изграждането на доверие у потребителите. Бяха анализирани основните подходи за удостоверяване на самоличността – еднофакторно, двуфакторно и многофакторно удостоверяване, както и различните механизми и протоколи, използвани в съвременните уеб приложения. Особено внимание беше отделено на типичните уязвимости и добрите практики за тяхното предотвратяване, като използване на силни пароли, хеширане, ограничаване на опитите за вход и криптиране на комуникацията.

Важен акцент беше поставен върху оторизацията – процесът, чрез който се определят правата и достъпът на вече удостоверените потребители. Разгледани бяха различните модели за контрол на достъпа, включително оторизация по роли (RBAC) и по атрибути (ABAC), както и примери за прилагането им в реални уеб приложения. Бяха анализирани и архитектурните модели и стандарти, които определят начина на управление на сесиите, бисквитките и токените (JWT), както и инструментите и библиотеките, използвани в различни технологични платформи.

В практическата част беше реализирано уеб приложение, базирано на Django, което демонстрира на практика процесите на регистрация, вход, управление на профил и контрол на достъпа до ресурси. Използвани бяха основните компоненти на Django – модели, форми, изгледи, маршрутизация и административен панел. Особено внимание беше отделено на разширяването и персонализацията на логиката за управление на достъпа, както и на интеграцията на добри практики за сигурност. Чрез реални примери и кодови фрагменти бяха илюстрирани основните стъпки по изграждане на сигурна система за удостоверяване и оторизация.

## Анализ на постигнатите цели

Поставените в началото на работата цели бяха изпълнени успешно. Беше направен задълбочен анализ на теоретичните основи на уеб сигурността, удостоверяването и оторизацията. Бяха представени и сравнени различни подходи и технологии, като специален фокус беше поставен върху възможностите на Django. В практическата част беше реализирано работещо приложение, което покрива всички основни сценарии за управление на потребители, роли и права.

Реализацията на приложението показа, че Django предоставя мощна и гъвкава платформа за изграждане на сигурни уеб системи. Вградената система за удостоверяване и оторизация, съчетана с възможностите за разширяване и персонализация, позволява бързо и ефективно разработване на приложения с различни нива на достъп и контрол. Използването на стандартните механизми за валидация, хеширане на пароли, управление на сесии и права гарантира високо ниво на сигурност, дори и при по-сложни бизнес изисквания.

## Ограничения и предизвикателства

Въпреки постигнатите резултати, трябва да се отбележат и някои ограничения на реализираното решение. В рамките на дипломната работа беше реализирана основна функционалност за удостоверяване и оторизация, но не бяха внедрени по-сложни механизми като двуфакторна или многофакторна автентикация, интеграция със социални мрежи или външни доставчици на идентичност (OAuth, SSO). Също така, приложението не включваше пълна защита срещу всички възможни уязвимости (например CSRF, XSS, brute-force атаки), въпреки че бяха използвани основните защитни механизми, предоставени от Django.

Друго предизвикателство е свързано с мащабируемостта и управлението на сложни политики за достъп в по-големи системи. В такива случаи може да се наложи използването на по-сложни модели за контрол на достъпа, интеграция с външни системи за управление на идентичност и прилагане на допълнителни мерки за мониторинг и одит на действията на потребителите.

## Възможности за бъдещо развитие

Въз основа на направения анализ и реализираното приложение могат да се очертаят няколко основни насоки за бъдещо развитие и усъвършенстване:

* **Внедряване на многофакторна автентикация (MFA):** Добавянето на втори или трети фактор за удостоверяване (например еднократни кодове, изпратени по имейл или SMS, или използване на мобилни приложения за генериране на кодове) значително повишава сигурността на системата.
* **Интеграция със социални мрежи и външни доставчици:** Използването на протоколи като OAuth и SSO позволява на потребителите да се удостоверяват чрез профилите си в Google, Facebook и други платформи, което улеснява достъпа и повишава удобството.
* **Разширяване на модела за контрол на достъпа:** При по-големи и сложни приложения може да се наложи внедряване на атрибутно-базиран контрол на достъпа (ABAC) или други гъвкави модели, които позволяват по-прецизно управление на правата.
* **Автоматизирано тестване и мониторинг на сигурността:** Внедряването на инструменти за автоматизирано тестване на уязвимости, мониторинг на логове и одит на действията на потребителите допринася за по-високо ниво на сигурност и бърза реакция при инциденти.
* **Обучение на потребителите:** Провеждането на обучения и предоставянето на ясни инструкции за сигурна работа с приложението намалява риска от човешки грешки и социално инженерство.

## Практически препоръки

Въз основа на направения анализ и реализираното приложение могат да се формулират следните практически препоръки за разработчици и администратори на уеб приложения:

* **Използвайте вградените механизми на Django:** Те са добре тествани, поддържат се активно и покриват основните изисквания за сигурност.
* **Редовно обновявайте зависимостите и следете за уязвимости:** Използването на актуални версии на фреймуърка и библиотеките намалява риска от експлоатация на известни уязвимости.
* **Внедрете HTTPS и криптирайте чувствителната информация:** Това е задължително изискване за всички съвременни уеб приложения.
* **Ограничете правата на потребителите до необходимия минимум:** Принципът на най-малките права (least privilege) е ключов за предотвратяване на злоупотреби.
* **Използвайте силни пароли и изисквайте регулярна смяна:** Внедрете политики за минимална дължина и сложност на паролите.
* **Следете и анализирайте логовете за подозрителна активност:** Това позволява ранно откриване на атаки и инциденти.
* **Провеждайте регулярни обучения по сигурност:** Информираността на потребителите е важен фактор за цялостната защита на системата.

## Заключителни думи

Темата за уеб сигурността, удостоверяването и оторизацията е изключително актуална и многопластова. С развитието на технологиите и нарастването на киберзаплахите, необходимостта от надеждни и сигурни решения става все по-голяма. Чрез анализа на теоретичните основи и практическата реализация с помощта на Django, настоящата работа даде цялостен поглед върху процесите на удостоверяване и оторизация, както и практически насоки за изграждане на сигурни уеб приложения.

В заключение може да се каже, че сигурността не е еднократен акт, а непрекъснат процес, който изисква постоянни усилия, внимание и адаптация към новите предизвикателства. Само чрез съчетаване на добри практики, подходящи технологии и информираност на всички участници може да се постигне високо ниво на защита и доверие в съвременните уеб системи.

# Литература:

1. W3Schools. HTML, CSS, JavaScript, Python, Django – онлайн уроци и документация. https://html.w3schools.com
2. D.K Academy. Python – практическо програмиране. Асеневци, София, 2022.
3. Django REST Framework – официална документация. https://www.django-rest-framework.org/
4. Колисниченко, Денис. Адаптивен уеб дизайн с Bootstrap. Асеневци, София, 2019.
5. SoftUni Blog. Какво е Django? https://softuni.bg/blog/what-is-django
6. Wikipedia. MySQL – енциклопедична статия. https://bg.wikipedia.org/wiki/MySQL
7. Донълдсън, Тоби. Бързо ръководство: Програмиране с Python. АлексСофт, София, 2017.
8. Уроци BG – онлайн ресурси по програмиране. https://urocibg.eu/
9. SoftUni Blog. Vue.js – въведение. https://softuni.bg/blog/vue-js-january-2021
10. Есканази, Аврам. Софтуерни технологии. КЛМН, София, 2006.
11. Cynoteck. Flask vs Django – сравнение на Python уеб фреймуъркове. https://cynoteck.com/bg/blog-post/flask-vs-django/
12. Inventum. CSS – уеб дизайн ресурси. https://www.inventum.bg/web-design/css/
13. Savtec. Основи на REST и RESTful API разработка. https://bg.savtec.org/articles/coding/the-basics-of-rest-and-restful-api-development.html
14. Django Project. Официална документация на Django. https://docs.djangoproject.com/bg/4.2/
15. OWASP Foundation. OWASP Top Ten – най-честите уеб уязвимости. https://owasp.org/www-project-top-ten/
16. Django Allauth – документация за социална автентикация. https://django-allauth.readthedocs.io/
17. Django REST Framework SimpleJWT – документация. https://django-rest-framework-simplejwt.readthedocs.io/
18. Python Software Foundation. Официална документация на Python. https://docs.python.org/3/
19. Mozilla Developer Network (MDN). Уеб сигурност – ръководства и статии. https://developer.mozilla.org/bg/docs/Web/Security