Съдържание

[1. Увод (въведение) 4](#_Toc196211407)

[2. Django – кратък преглед на фреймуърка 4](#_Toc196211408)

[2.1 Значение 4](#_Toc196211409)

[2.2 Основни характеристики на Django 5](#_Toc196211410)

[2.3 Структура на Django проект 5](#_Toc196211411)

[2.4 Развитие и общност 6](#_Toc196211412)

[2.5 Предимства на Django 6](#_Toc196211413)

[2.6 Django и образователната среда 6](#_Toc196211414)

[3. Django Forms 7](#_Toc196211415)

[3.1 Въведение 7](#_Toc196211416)

[3.2 Основни концепции 7](#_Toc196211417)

[3.3 Използване на форми във view 7](#_Toc196211418)

[3.4 Рендиране на формата в шаблон 8](#_Toc196211419)

[3.5 Видове полета във форми 8](#_Toc196211420)

[3.6 Персонализиране на формите 9](#_Toc196211421)

[3.7 Валидация 10](#_Toc196211422)

[3.8 Съобщения за грешки 10](#_Toc196211423)

[4. ModelForms 10](#_Toc196211424)

[4.1 Въведение 10](#_Toc196211425)

[4.2 Създаване на ModelForm 11](#_Toc196211426)

[4.3 Използване във View 12](#_Toc196211427)

[4.4 Полета и конфигурация 12](#_Toc196211428)

[4.5 Предимства на ModelForms 13](#_Toc196211429)

[4.6 Валидация при ModelForms 13](#_Toc196211430)

[4.7 Пример от практиката: форма за добавяне на продукт 14](#_Toc196211431)

[5. Валидиране на данни 14](#_Toc196211432)

[5.1 Въведение 14](#_Toc196211433)

[5.2 Вградена валидация в Django 14](#_Toc196211434)

[5.3 Валидация с метода is\_valid() 15](#_Toc196211435)

[5.4 Персонализирана валидация с clean\_<поле>() 15](#_Toc196211436)

[5.5 Глобална валидация с clean() 16](#_Toc196211437)

[5.6 Валидиране на ModelForms 16](#_Toc196211438)

[5.7 Използване на валидатори от django.core.validators 17](#_Toc196211439)

[5.8 Съобщения за грешки 17](#_Toc196211440)

[5.9 Добри практики 18](#_Toc196211441)

[6. Избор на технологии и аргументация 18](#_Toc196211442)

[6.1 Въведение 18](#_Toc196211443)

[6.2 Аргументация за избор на Django 18](#_Toc196211444)

[6.3 Избор на език за програмиране: Python 19](#_Toc196211445)

[6.4 Използвани допълнителни технологии 19](#_Toc196211446)

[6.5 Сравнение с алтернативни технологии 19](#_Toc196211447)

[6.6 Практическа приложимост 20](#_Toc196211448)

[6.7 Възможност за разширение 20](#_Toc196211449)

[6.8 Образователен аспект на избраните технологии 20](#_Toc196211450)

[6.9 Практика, ориентирана към реалността 21](#_Toc196211451)

[7. Анализ на съществуващи решения (други фреймуърци или библиотеки) 21](#_Toc196211452)

[7.1 Въведение 21](#_Toc196211453)

[7.2 Сравнение с Flask (Python) 22](#_Toc196211454)

[7.3 Сравнение с Laravel (PHP) 22](#_Toc196211455)

[7.4 Сравнение с React + Formik / React Hook Form 22](#_Toc196211456)

[7.5 Алтернативни библиотеки за форми и валидация 23](#_Toc196211457)

[8. Избор на технологии и аргументация 23](#_Toc196211458)

[8.1 Въведение 23](#_Toc196211459)

[8.2 Защо е избран Django 24](#_Toc196211460)

[8.2.1 Пълноценен фреймуърк „всичко в едно“ 24](#_Toc196211461)

[8.2.2 Прост и четим синтаксис 24](#_Toc196211462)

[8.2.3 Високо ниво на сигурност 24](#_Toc196211463)

[8.2.4 Отлична поддръжка и документация 25](#_Toc196211464)

[8.2.5 Подходящ за мащабиране 25](#_Toc196211465)

[8.3 Защо формите са подходящи за онлайн магазин 25](#_Toc196211466)

[8.3.1 Формите като интерфейс за взаимодействие 25](#_Toc196211467)

[8.3.2 Вградено валидиране на данни 25](#_Toc196211468)

[8.3.3 Връзка с базата данни чрез ModelForms 26](#_Toc196211469)

[8.4 Персонализиране и стил 26](#_Toc196211470)

[8.5 Контрол върху процеса 26](#_Toc196211471)

[9. Приноси на дипломния проект 27](#_Toc196211472)

[9.1 Въведение 27](#_Toc196211473)

[9.2 Технологични приноси 27](#_Toc196211474)

[9.3 Образователни и лични приноси 27](#_Toc196211475)

[9.4 Значимост и приложимост 28](#_Toc196211476)

[10. Практическо изпълнение 28](#_Toc196211477)

[10.1 ModelForm за събиране на информация за потребителя и адреса. 28](#_Toc196211478)

[10.2 Добавяне на метод за валидация на имейл 29](#_Toc196211479)

[10.3 Добавяне на продукт към количката 30](#_Toc196211480)

[10.4 Изчистване на количката 32](#_Toc196211481)

[11. Заключение 32](#_Toc196211482)

[12. Литература: 34](#_Toc196211483)

# ****Увод (въведение)****

С развитието на уеб технологиите и нарастващата нужда от дигитализация в различни сфери на живота, създаването на динамични, интерактивни и сигурни уеб приложения се превръща във все по-важна част от съвременното програмиране. В центъра на всяко уеб приложение стои обменът на информация между потребителя и системата – процес, който обикновено се реализира чрез форми. Те позволяват събиране на данни, извършване на заявки, обработка на поръчки, регистрация на потребители и много други действия.

Настоящият дипломен проект е посветен на изграждането на **динамична форма за онлайн магазин** чрез използване на **Django Forms** – част от мощната екосистема на уеб фреймуърка **Django**, базиран на езика Python. Django предоставя добре структуриран и сигурен начин за създаване на уеб приложения, а системата му за форми предлага удобен механизъм за валидиране на входни данни, обработка на заявки и взаимодействие с база от данни.

Проектът е **теоретико-практическа разработка**, която обединява изучените технологии в специалността „Приложно програмиране“ и демонстрира способността за изграждане на пълноценен уеб интерфейс с реална функционалност. В теоретичната част се разглеждат принципите на Django Forms, типовете форми, механизмите за валидиране на данни и обработка на грешки. Практическата част включва разработването на модул за електронен магазин, чрез който потребителят може да избира продукти и да ги добавя в кошница.

Целта на дипломната работа е да покаже не само усвояването на конкретни технологии, но и умението за тяхното приложение в реална разработка, с акцент върху добри практики, сигурност и потребителско изживяване.

# Django – кратък преглед на фреймуърка

## Значение

Django е високонативен, безплатен и с отворен код уеб фреймуърк, написан на езика за програмиране Python. Той позволява бързото разработване на уеб приложения, като предоставя набор от готови инструменти и компоненти за изграждане на стабилни, сигурни и поддържащи се проекти. Django следва принципа **"Don't Repeat Yourself" (DRY)** – не повтаряй себе си – което насърчава писането на чист, модулен и многократно използваем код.

Django е създаден през 2005 година от група разработчици, работещи по новинарски уеб сайтове. Техният основен стремеж е бил да автоматизират и ускорят изграждането на функционални уеб страници, без да се жертва сигурността и добрият дизайн. Оттогава насам фреймуъркът е прераснал в едно от най-популярните решения за разработка на уеб приложения на Python.

## Основни характеристики на Django

* **Модел-Шаблон-Изглед (MTV) архитектура**: Django използва архитектурен модел, подобен на MVC (Model-View-Controller), като разделя логиката на приложението на три основни компонента – модел (Model), шаблон (Template) и изглед (View). Тази структура улеснява поддръжката и разширението на проектите.
* **Вграден администраторски панел**: Един от най-силните инструменти в Django е автоматично генерираният административен интерфейс. Той позволява лесно управление на данни чрез уеб браузър без необходимост от писане на собствен код.
* **ORM (Object-Relational Mapping)**: Django включва мощен механизъм за работа с бази данни чрез обекти. ORM позволява взаимодействие с базата данни чрез Python код, без необходимост от директни SQL заявки.
* **Сигурност**: Фреймуъркът предоставя вградена защита срещу основни уеб уязвимости като SQL инжекции, cross-site scripting (XSS), cross-site request forgery (CSRF) и clickjacking.
* **Гъвкавост и разширяемост**: Django разполага с богата екосистема от разширения (пакети), които позволяват добавяне на нова функционалност с минимални усилия – потребителска регистрация, REST API, плащания и др.

## Структура на Django проект

При създаване на нов Django проект, се генерира основна структура от директории и файлове, които организират логиката и съдържанието на приложението. Основните компоненти включват:

* manage.py – основен управляващ скрипт за администрация на проекта;
* settings.py – конфигурационен файл за настройка на проекта (база данни, инсталирани приложения, език и др.);
* urls.py – файл за конфигуриране на маршрутите на приложението;
* models.py – дефиниции на модели (таблици) и тяхната логика;
* views.py – изгледи, които определят какво да се показва при даден URL;
* templates/ – директория с HTML шаблони;
* static/ – директория за статични ресурси (CSS, JavaScript, изображения).

Благодарение на добре организираната си структура, Django позволява разделение на отговорностите и улеснява работата в екип.

## Развитие и общност

Django се поддържа от активна общност от разработчици, както и от фондацията Django Software Foundation (DSF), която следи за развитието и стандартизацията на проекта. Благодарение на редовни обновления, документация и ресурси за обучение, фреймуъркът е предпочитан избор не само от начинаещи, но и от големи компании.

Множество добре познати уебсайтове и платформи използват Django – сред тях са **Instagram, Mozilla, Disqus, Pinterest** и **The Washington Post**. Това доказва, че фреймуъркът е подходящ не само за малки приложения, но и за мащабни системи с милиони потребители.

## Предимства на Django

* **Бързина на разработка** – благодарение на предварително изградени компоненти и библиотеки, разработването на уеб приложение с Django е значително по-бързо.
* **Ясна документация** – Django е известен със своята изчерпателна и подредена документация, която улеснява усвояването на фреймуърка.
* **Модулност** – всеки Django проект се състои от отделни приложения (apps), които могат да се използват повторно в други проекти.
* **Добра интеграция с други технологии** – може лесно да се комбинира с JavaScript библиотеки като React или Vue, както и с REST API чрез Django REST Framework.

## Django и образователната среда

Django е особено подходящ за ученици и студенти, тъй като е лесен за усвояване и предоставя реални практически умения. Работата с този фреймуърк запознава учащите с основни концепции от уеб разработката като шаблониране, маршрутизиране, обработка на заявки, бази данни и форми. Това го прави изключително ценен инструмент в професионалната подготовка на приложни програмисти.

# Django Forms

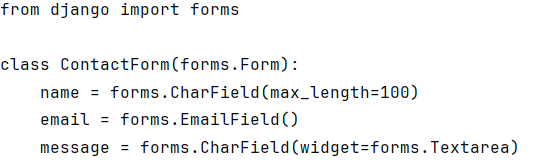
## Въведение

Една от най-мощните и полезни функционалности на Django е системата за работа с **форми**. Django Forms предоставя структуриран и сигурен начин за събиране, обработка и валидиране на входни данни от потребителите. Вместо ръчно да се пише HTML код, да се проверява входът и да се изгражда логика за грешки, Django автоматизира по-голямата част от процеса. Това улеснява разработчиците и намалява риска от уязвимости.

## Основни концепции

Формата в Django представлява Python клас, който дефинира полета, типове данни и методи за валидация. Django автоматично може да генерира HTML код, да обработва POST заявки и да визуализира съобщения за грешки.

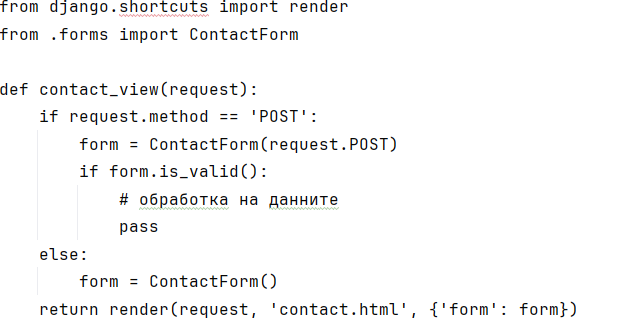
**Пример за обикновена форма:**



Тук се създава форма за контакт с три полета: име, имейл и съобщение. Django ще се погрижи за автоматичното им визуализиране и валидиране.

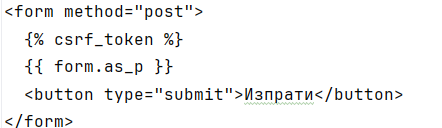
## Използване на форми във view

След като формата е създадена, тя се използва във **view** функция, за да се обработват GET и POST заявки. Пример:



## Рендиране на формата в шаблон

Формите могат да се рендират автоматично с помощта на Django шаблони:

****

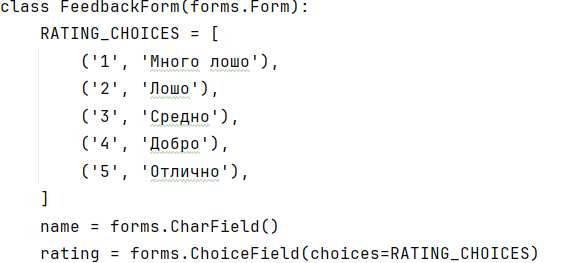
Методът {{ form.as\_p }} визуализира формата като HTML с <p> тагове. Алтернативни методи са as\_table() и as\_ul().

## Видове полета във форми

Django предлага голямо разнообразие от типове полета, в зависимост от типа на данните:

* CharField – текстово поле
* EmailField – имейл адрес
* IntegerField – цели числа
* BooleanField – чекбокс
* ChoiceField – падащо меню
* DateField – дата
* FileField и ImageField – файлове и изображения

**Пример за форма с избор:**

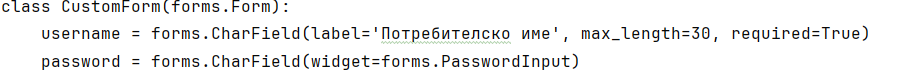
****

## Персонализиране на формите

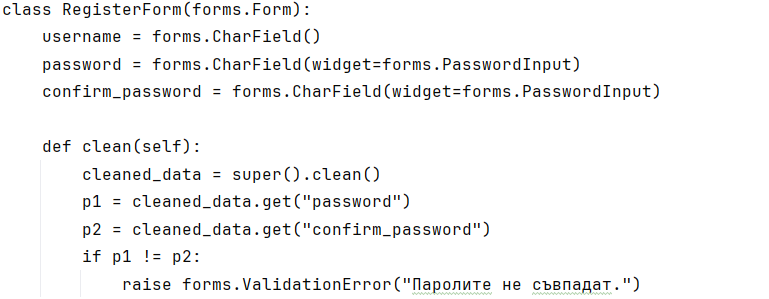
Освен автоматичното поведение, формите могат да бъдат персонализирани с помощта на параметри:

* label – променя текста на етикета
* initial – стойност по подразбиране
* required – дали полето е задължително
* widget – за избор на HTML елемент

Пример:

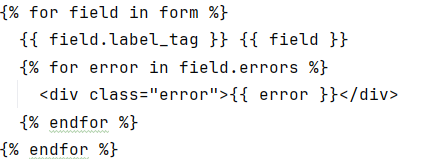


## Валидация

Django автоматично валидира стойности според типа на полето. Освен това, могат да се добавят **потребителски валидации** с методите clean\_<име на поле> или общия clean():

## Съобщения за грешки

Ако формата е невалидна, Django автоматично показва грешки до съответните полета. В шаблоните можем да визуализираме съобщенията с:



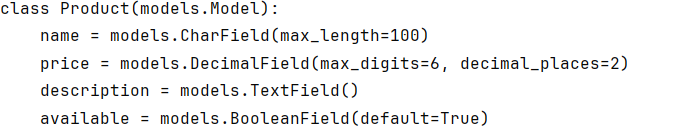
# ModelForms

## Въведение

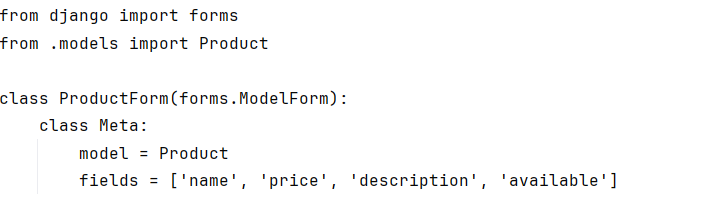
В предходната точка разгледахме стандартните Django форми, които се изграждат ръчно чрез дефиниране на всяко поле. Django обаче предоставя още по-удобен начин за създаване на форми, когато те трябва да отразяват структурата на база данни – чрез **ModelForms.** ModelForm е клас във Django, който автоматично създава форма въз основа на вече дефиниран модел. Това спестява време, премахва дублиране на код и гарантира, че формата ще бъде в синхрон с базата данни.

## Създаване на ModelForm

За да създадем ModelForm, първо трябва да имаме модел, дефиниран чрез Django ORM. Пример:

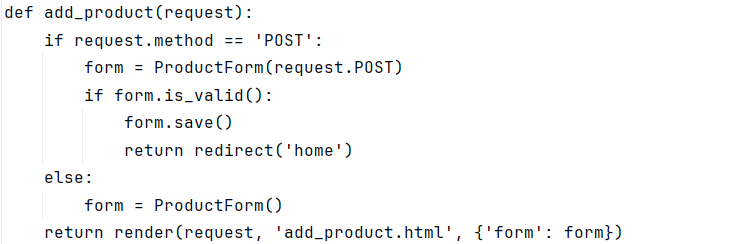


След това създаваме ModelForm, свързан с този модел:



Django автоматично ще създаде HTML форма с полетата от модела и ще приложи съответните валидации.

## Използване във View

ModelForm се използва по същия начин, както обикновените форми:

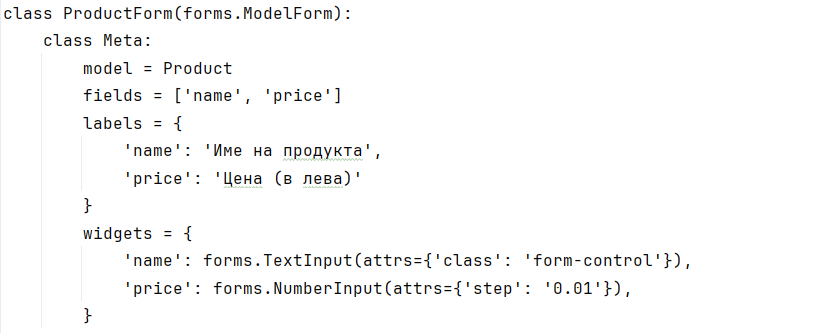
Методът form.save() автоматично създава или обновява запис в базата данни на базата на въведените стойности.

## Полета и конфигурация

**ModelForm поддържа следните опции в Meta класа:**

* model – моделът, към който е свързана формата
* fields – списък с полета, които да се включат във формата
* exclude – полета, които да се изключат (алтернатива на fields)
* labels, widgets, help\_texts, error\_messages – персонализация на визуализацията

Пример с персонализиране:

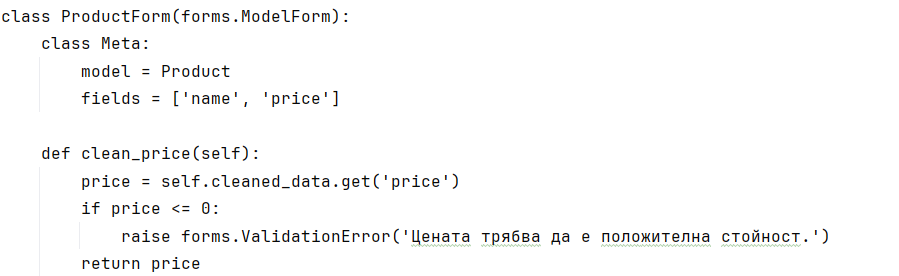


## Предимства на ModelForms

* **Съкращават кода** – автоматично създаване на полета.
* **Синхронизация с моделите** – ако се промени моделът, формата автоматично отразява промените.
* **Вградена валидация** – валидират се типове данни, дължина, уникалност и други ограничения.
* **Съвместимост със save()** – автоматично създаване и запазване на обекти.

## Валидация при ModelForms

Както при стандартните форми, можем да добавим допълнителна валидация чрез clean\_<field>() или clean() методи.



## Пример от практиката: форма за добавяне на продукт

В контекста на електронен магазин, можем да създадем ModelForm за добавяне на продукти в системата. Това позволява администраторът да въвежда нови продукти чрез уеб интерфейс без нужда от достъп до базата данни директно.

Тази функционалност е част от практическата част на дипломния проект, където се използва ModelForm за динамично добавяне на продукти в клиентската кошница.

# Валидиране на данни

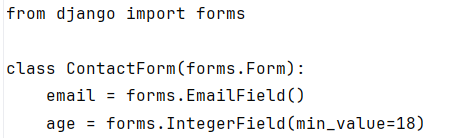
## Въведение

Валидирането на данни е изключително важна част от всеки уеб проект. То гарантира, че потребителският вход е правилен, логически коректен и безопасен за обработка. Django предлага мощна и гъвкава система за валидация, която може да се прилага както на ниво форма, така и на ниво модел. С помощта на Django Forms и ModelForms, валидирането се извършва автоматично за повечето стандартни типове данни, но също така позволява и създаване на персонализирани правила.

## Вградена валидация в Django

Когато се използват стандартни полета от forms.Form или forms.ModelForm, Django автоматично валидира въведените данни според дефинираните типове:

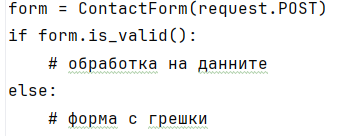
* CharField – изисква текст със зададена максимална (и евентуално минимална) дължина.
* EmailField – проверява дали въведеното съдържа валиден имейл адрес.
* IntegerField – приема само цели числа.
* DecimalField – проверява за коректен числов формат с точност.
* BooleanField – очаква стойности True/False.

Пример:

Тук автоматично ще бъдат валидирани както имейлът, така и възрастта (минимум 18).

## Валидация с метода is\_valid()

Основният метод, който се използва за проверка на валидността на форма в Django, е is\_valid():

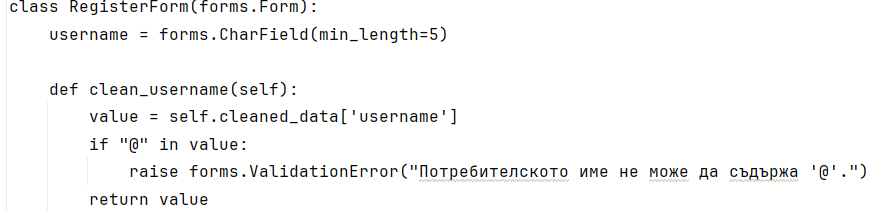


Когато се извика is\_valid(), Django:

* Извлича въведените данни;
* Изпълнява вградената валидация за всеки тип поле;
* Изпълнява всички персонализирани методи за валидация;
* Попълва речника form.errors, ако има проблеми.

## Персонализирана валидация с clean\_<поле>()

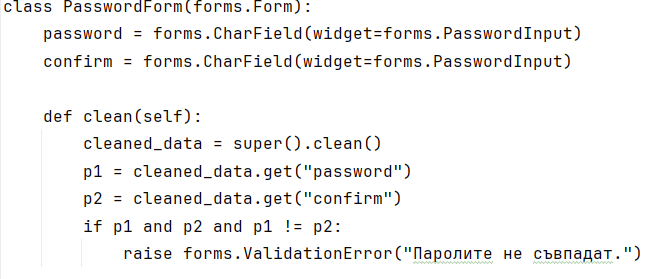
За всяко поле може да се създаде метод с име clean\_<fieldname>, който позволява допълнителна логика за валидация. Пример:



Ако потребителят въведе недопустима стойност, формата ще бъде маркирана като невалидна, а съобщението ще се покаже до съответното поле.

## Глобална валидация с clean()

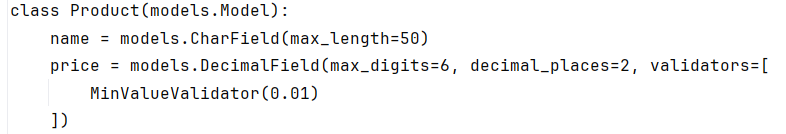
Понякога е необходимо да се сравняват стойности между различни полета. Това се прави в метода clean():



Методът clean() се извиква след всички clean\_<поле>() методи.

## Валидиране на ModelForms

Когато използваме ModelForm, Django автоматично използва валидаторите, дефинирани в модела. Например:



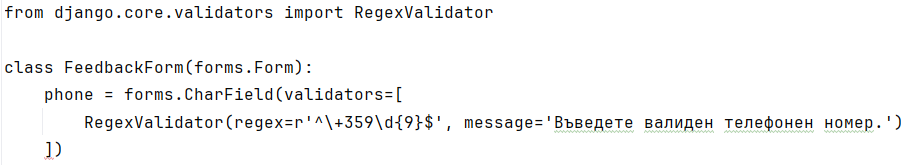
Този валидатор гарантира, че цената не може да бъде по-малка от 0.01. При ModelForm тази проверка ще се извърши автоматично.

## Използване на валидатори от django.core.validators

Django предлага голям брой готови валидатори, които могат да се използват в модели и форми:

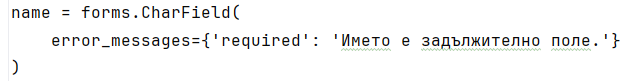
* EmailValidator
* RegexValidator
* URLValidator
* MinValueValidator, MaxValueValidator
* validate\_slug, validate\_email

Пример:

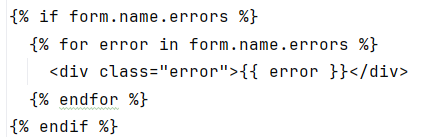


## Съобщения за грешки

Django автоматично показва съобщенията за грешки в шаблона. Те могат да бъдат персонализирани:



В HTML шаблон:



## Добри практики

* Използвай вградени валидатори, когато е възможно.
* За прости проверки използвай clean\_<поле>().
* За зависимости между полета – използвай clean().
* Не забравяй да използваш is\_valid() преди да достъпваш cleaned\_data.
* Използвай смислени съобщения за грешки – това подобрява потребителското преживяване.

# Избор на технологии и аргументация

## Въведение

В процеса на разработка на уеб приложения изборът на технологии играе ключова роля за успеха на проекта. Той определя не само ефективността на изпълнение, но и лекотата на поддръжка, сигурността, възможността за разширение и интеграция с други системи. В този дипломен проект е реализиран потребителски интерфейс на електронен магазин с помощта на **Django** – уеб фреймуърк на езика **Python,** в комбинация с **HTML, CSS** и **Bootstrap** за изграждане на визуалната част на приложението.

## Аргументация за избор на Django

Django е избран като основна платформа за разработка поради следните причини:

* **Бързо прототипиране и развитие**: Django предоставя готови инструменти за управление на потребители, форми, маршрути, администраторски панел и ORM (Object-Relational Mapping), което значително съкращава времето за разработка.
* **Сигурност**: Фреймуъркът включва вградена защита срещу често срещани уеб атаки като SQL инжекции, XSS (cross-site scripting), CSRF (cross-site request forgery) и clickjacking. Това е особено важно при работа с чувствителни данни като лична информация и онлайн поръчки.
* **Гъвкава система за форми**: Django Forms и ModelForms улесняват създаването на динамични и валидирани форми – основен компонент в електронната търговия.
* **Активна общност и документация**: Django е добре документиран, а общността предоставя стотици примери, библиотеки и пакети за допълнителна функционалност.
* **Подходящ за образователни цели**: Django е логически структуриран и предлага ясна разделеност между логика, визуализация и данни – което го прави лесен за разбиране и прилагане от ученици и студенти.

## Избор на език за програмиране: Python

Python е един от най-популярните и четими езици за програмиране. Той е лесен за учене и има широка поддръжка от библиотеки и инструменти. Python се използва активно в образованието, науката, анализа на данни и уеб разработката. Django е създаден именно на Python, което е още една причина за избора на този език за проекта.

## Използвани допълнителни технологии

* **HTML и CSS** – за изграждане на потребителския интерфейс на формите и страниците.
* **Bootstrap** – CSS фреймуърк за по-бързо и лесно създаване на адаптивен и модерен уеб дизайн.
* **SQLite** – базата данни по подразбиране в Django, подходяща за малки проекти и демонстрации.

## Сравнение с алтернативни технологии

В процеса на избор бяха разгледани и други популярни уеб фреймуърци, като Flask, Laravel, Ruby on Rails и ASP.NET. Всеки от тях има своите предимства, но спрямо нуждите на проекта Django се оказа най-подходящ:

* Flask е лек Python фреймуърк, който предоставя повече свобода, но изисква ръчно изграждане на множество функционалности – форми, ORM, администраторски панел и др. Това би довело до по-дълъг процес на разработка и по-голяма сложност за ученик в образователен контекст.
* Laravel (на PHP) и Ruby on Rails също предлагат добра структура и инструменти, но не са толкова популярни в учебните среди в България, а и изискват усвояване на различен език и среда.
* ASP.NET (на C#) е мощен фреймуърк за корпоративни уеб приложения, но се използва предимно в Microsoft среда и има по-висока сложност за начинаещи.

Django, от своя страна, предлага „всичко в едно“ – вграден админ панел, ORM, работа с форми, шаблони, сигурност, и всичко това с минимална нужда от допълнителни настройки.

## Практическа приложимост

Технологиите, избрани в проекта, са широко използвани в реални уеб приложения. Знанията и уменията, придобити чрез работата по проекта, ще бъдат полезни при:

* изграждане на пълноценни уеб системи;
* създаване на REST API за мобилни приложения;
* разработка на административни панели и системи за управление на данни;
* бъдеща професионална реализация като junior web developer.

## Възможност за разширение

Една от причините за избора на Django е и неговата **възможност за разширение**. В бъдеще проектът може да бъде допълнен с:

* потребителска регистрация и профили;
* плащания с PayPal или други методи;
* интеграция на REST API чрез Django REST Framework;
* отделен front-end с React или Vue.js за по-модерна визуализация.

Благодарение на добре структурираната си архитектура, Django позволява постепенно и контролирано разширяване на приложението, без нужда от пренаписване.

## Образователен аспект на избраните технологии

От гледна точка на обучението и подготовката на бъдещ приложен програмист, работата с Django и Python осигурява **силна основа за професионално развитие**. Използваните технологии не само отговарят на съвременните пазарни изисквания, но и спомагат за изграждането на ключови умения:

* **Работа с MVC/MTV архитектура** – концепция, приложима и в други езици и фреймуърци.
* **Работа с бази данни и ORM** – основополагащо умение за съвременните програмисти.
* **Умения за структуриране на проект** – чрез разделяне на логика, шаблони, модели и форми.
* **Разбиране на валидация и сигурност** – важни аспекти при всяко реално уеб приложение.

Освен техническите аспекти, ученикът развива и **умения за самостоятелна работа**, търсене на решения, използване на документация, онлайн ресурси и добри практики – качества, ценени във всяка професионална среда.

## Практика, ориентирана към реалността

Избраният проект е не просто учебна задача, а **мини версия на реално уеб приложение**, каквито се срещат всеки ден в електронната търговия. Това прави работата по него значима и мотивираща, а резултатите – видими и приложими. В процеса на реализация ученикът се докосва до реални казуси:

* потребителско взаимодействие с форма;
* обработка на POST заявки;
* динамично визуализиране на съобщения за грешки;
* съхранение на информация в база данни;
* генериране на HTML съдържание чрез шаблони;
* структуриран код и лесна навигация в проект.

# Анализ на съществуващи решения (други фреймуърци или библиотеки)

## Въведение

В съвременната уеб разработка съществуват множество фреймуърци и библиотеки, които предлагат различни подходи за създаване на динамични форми, обработка на данни и изграждане на потребителски интерфейси. Django е само едно от многото решения, но е важно да се направи сравнителен анализ с други популярни технологии, за да се разберат неговите предимства и ограничения.

## Сравнение с Flask (Python)

**Flask** е друг популярен уеб фреймуърк на Python, който често се сравнява с Django. Основната разлика между тях е в архитектурата и обхвата на функциите:

* Flask е **микрофреймуърк**, който предоставя само основната функционалност за уеб сървър, а всичко останало се добавя чрез разширения.
* Django **е пълноценен фреймуърк** („всичко в едно“), който включва ORM, система за форми, шаблони, сигурност и админ панел по подразбиране.

**От гледна точка на формите:**

* Flask не разполага с вградена система за форми. Обикновено се използва **WTForms** – отделна библиотека за създаване и валидация на формуляри.
* Django предлага **интегрирана система за форми**, която е по-добре документирана и тясно свързана с ORM модела.

Заключение: Flask е подходящ за по-леки и гъвкави проекти, но за учебни и по-структурирани приложения Django е по-ефективен.

## Сравнение с Laravel (PHP)

**Laravel** е популярен фреймуърк на PHP, който има сходна структура с Django:

* Поддържа MVC архитектура;
* Има ORM система (Eloquent);
* Предоставя Blade шаблони и система за маршрути.

## Сравнение с React + Formik / React Hook Form

В по-модерната front-end разработка с JavaScript фреймуърка **React**, съществуват библиотеки като **Formik** и **React Hook Form**, които позволяват лесно изграждане на форми с висока интерактивност.

* Предимства: отлична производителност, голяма гъвкавост, подходящи за SPA (Single Page Applications).
* Недостатъци: изискват сложна конфигурация, разделят front-end и back-end логиката, което увеличава сложността.

Django, в комбинация с Django Forms, позволява **по-лесно изграждане на пълноценна форма с валидиране директно от бекенда**, без нужда от отделен front-end фреймуърк.

## Алтернативни библиотеки за форми и валидация

Освен WTForms и Formik, съществуват и други библиотеки, в зависимост от езика и средата:

* **Express-validator** (Node.js)
* **Yup** (валидираща библиотека за JavaScript)
* **Ruby on Rails – Active Record validations**
* **ASP.NET – Data Annotations**

Всички те предлагат механизми за валидация, но често изискват допълнителна конфигурация и не са така интегрирани както Django Forms и ModelForms.

Laravel също разполага със система за форми, макар че тя често изисква повече конфигурация. Формите могат да бъдат валидирани с помощта на специални request класове или ръчно.

Въпреки че Laravel е силен избор в PHP средата, езикът Python, използван от Django, се счита за по-четим, по-подходящ за начинаещи и по-популярен в образованието.

# Избор на технологии и аргументация

## Въведение

Изборът на подходящи технологии е ключов момент при проектирането и реализацията на софтуерен проект. В настоящия дипломен проект се цели създаването на **динамичен интерфейс за онлайн магазин**, който използва **форми за добавяне на продукти в количка**. За тази цел е избран **Django** – високонадежден Python-базиран уеб фреймуърк, който предоставя вградени механизми за работа с форми, модели и шаблони.

## Защо е избран Django

Django е един от най-популярните фреймуърци за уеб разработка с Python. Той предоставя цялостна платформа за създаване на уеб приложения от началото до края. Причините за избора му в рамките на този проект са както технически, така и образователни.

### Пълноценен фреймуърк „всичко в едно“

**Django предлага готови модули за:**

* работа с бази от данни чрез ORM (Object-Relational Mapping);
* създаване и валидиране на форми;
* изграждане на шаблони и страници;
* управление на URL маршрути;
* защита от най-често срещаните уязвимости;
* автоматично генериране на админ панел.

Този подход „всичко в едно“ значително улеснява работата на разработчика и е особено подходящ за учебни цели, където основната цел е разбиране на концепции, а не писане на нисконивоен код.

### Прост и четим синтаксис

Django използва Python – един от най-лесните за разбиране езици за програмиране. Това го прави чудесен избор за ученици и начинаещи програмисти. Логиката е ясна, структурата е последователна, а документацията – обширна.

### Високо ниво на сигурност

**Django автоматично предпазва от:**

* SQL инжекции;
* XSS (cross-site scripting);
* CSRF (cross-site request forgery);
* Clickjacking.

В контекста на онлайн магазин, където се обменя чувствителна потребителска информация, това е огромно предимство.

### Отлична поддръжка и документация

Django има активна общност, хиляди уроци и огромна база знания. Това позволява на ученика да се ориентира бързо при нужда от помощ или разширяване на проекта.

### Подходящ за мащабиране

С Django могат да се изграждат както малки учебни системи, така и големи търговски платформи. Фактът, че се използва от сайтове като Instagram, Mozilla, Disqus и Pinterest, е доказателство за неговата надеждност и мащабируемост.

## Защо формите са подходящи за онлайн магазин

Един онлайн магазин е интерфейс, който трябва да приема и обработва голямо количество **входна информация от потребителя** – продуктови количества, адреси за доставка, данни за контакт и предпочитания. Именно тук **формите** играят централна роля.

### Формите като интерфейс за взаимодействие

Формите са най-естественият начин за потребителя да комуникира с уеб приложението. В онлайн магазин, това включва:

* избор на продукт и количество;
* добавяне в количка;
* попълване на данни за доставка;
* потвърждаване на поръчка.

С Django Forms това взаимодействие става лесно за изграждане, валидиране и защита.

### **Вградено валидиране на данни**

Django Forms и ModelForms позволяват автоматична проверка дали въведената информация е:

* пълна (required);
* от правилния тип (напр. имейл, число, дата);
* логически валидна (съответствие между полета, допустим диапазон и др.).

В онлайн магазин това гарантира, че потребителят не може да изпрати непълна или невалидна поръчка.

### Връзка с базата данни чрез ModelForms

С ModelForms могат директно да се свържат формите с моделите (напр. Product, Order, CartItem), което позволява автоматично:

* създаване на нови записи;
* обновяване на съществуващи данни;
* обвързване на потребителски действия с реални обекти в базата.

Това е от ключово значение при електронната търговия, където всяка потребителска сесия трябва да бъде обвързана с реални данни.

## Персонализиране и стил

С помощта на HTML шаблони и CSS, Django формите могат да бъдат стилизирани според дизайна на магазина. Могат да се използват библиотеки като Bootstrap, за да се добавят:

* падащи менюта;
* чекбоксове;
* валидиращи съобщения в реално време;
* бутон „добави в количка“ и „изпрати поръчка“.

Това осигурява **добро потребителско изживяване**, което е критично важно за всеки онлайн магазин.

## Контрол върху процеса

**С Django имаме пълен контрол върху:**

* кога формата се счита за валидна;
* какви съобщения за грешки се показват;
* какво се случва след изпращането ѝ.

Това е изключително важно в онлайн пазаруването, където точността и обратната връзка към потребителя са от решаващо значение.

# Приноси на дипломния проект

## Въведение

Дипломният проект на тема **„Работа с Django Forms и валидиране на данни“** съчетава теоретични познания и практически умения, придобити в хода на обучението по специалността „Приложно програмиране“. Чрез реализацията на напълно функциониращ модул от онлайн магазин проектът не само демонстрира разбирането на основни уеб концепции, но и допринася с конкретни резултати и знания, приложими в реална среда.

## Технологични приноси

* **Реализация на уеб интерфейс с динамична форма**  
  Изграден е модул за избор и добавяне на продукти в количка чрез динамични Django форми. Формата позволява въвеждане на информация, извършва валидация и предоставя обратна връзка към потребителя, като се доближава до реалните изисквания на електронната търговия.
* **Интеграция между потребителския интерфейс и базата данни**Проектът показва как потребителски вход (чрез формата) може да се използва за създаване на записи в база от данни с помощта на Django ModelForms и ORM. Това обединява front-end и back-end логиката в една стабилна архитектура.
* **Прилагане на валидиращи механизми**  
  Чрез използване на методите is\_valid(), clean() и clean\_<поле>() е изградена логика, която гарантира, че подадените данни отговарят на изискванията, преди да бъдат обработени или съхранени.

## Образователни и лични приноси

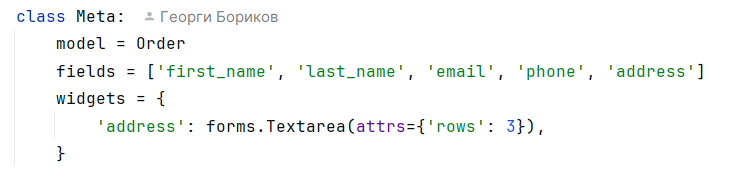
* **Задълбочено усвояване на Django и Python**В процеса на работа по проекта ученикът е овладял работата с един от най-популярните уеб фреймуърци – Django, както и приложението му в реални проекти чрез Python. Това е значим напредък в развитието на програмните умения.
* **Придобиване на практически опит с уеб технологии**Проектът е изграждан с HTML, CSS и Django templates, като в процеса е постигната практическа подготовка за създаване на пълноценни уеб страници и интерфейси.
* **Изграждане на логическо мислене и умения за самостоятелна работа**Учениците се сблъскват с реални казуси – свързани с рендиране на форми, работа с POST заявки, валидиране, маршрутизация и база данни. Решаването им изисква аналитично мислене, търсене на решения и самостоятелна проверка на документация.

## Значимост и приложимост

* **Практическа приложимост на създадения код**  
  Разработеният модул може лесно да бъде разширен и интегриран в по-голяма платформа. Това го прави не просто учебна симулация, а **реален приносен елемент**, приложим в бизнес или учебна среда.
* **Създаване на основа за надграждане**Благодарение на добрата архитектура на Django, проектът може да бъде допълнен с модули за плащане, регистрация на потребители, изпращане на имейли, REST API и други. По този начин се създава основа за бъдещи разработки.

# Практическо изпълнение

## ModelForm за събиране на информация за потребителя и адреса.



**model = Order**

Това указва, че формата е свързана с модела Order. Django ще използва структурата на този модел, за да създаде съответните полета във формата.

**fields = [...]**

Списък от полета от модела, които ще бъдат включени във формата:

* first\_name – собствено име
* last\_name – фамилно име
* email – имейл адрес
* phone – телефонен номер
* address – адрес за доставка

Тези полета ще се появят в HTML формата и ще могат да бъдат попълвани от потребителя.

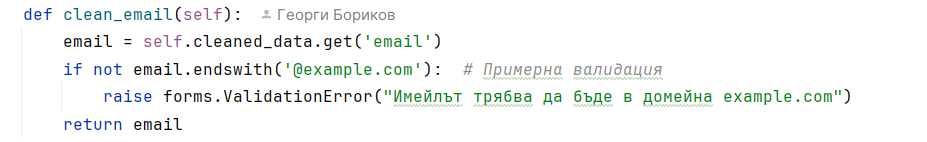
**widgets = {...}**

Позволява персонализиране на HTML елементите (т.нар. **уиджети)** за конкретни полета. В случая:



означава, че полето address ще бъде визуализирано не като едноредово поле (<input>), а като многострочно текстово поле (<textarea>), с 3 реда височина.

## Добавяне на метод за валидация на имейл



**self.cleaned\_data.get('email')**  
Взима стойността, която потребителят е въвел в полето email, след като е преминала базова валидация (тип EmailField и задължителност).

**Проверка с email.endswith('@example.com')**  
Ако имейлът **не** завършва с домейна @example.com, се хвърля грешка чрез:

**raise forms.ValidationError(...)**  
Това прекъсва процеса и отбелязва полето като **невалидно**, показвайки съобщението до потребителя.

Ако всичко е наред, методът връща email.

## Добавяне на продукт към количката



Това е функция, която приема request (заявка от браузъра) и pk – първичния ключ (ID) на продукта, който ще бъде добавен.



С помощта на Django функцията get\_object\_or\_404() се опитваме да вземем продукта от базата. Ако не съществува – връщаме 404 грешка.



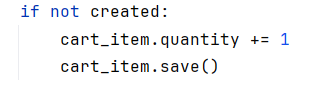
Това е помощна функция (предполага се, че е дефинирана другаде), която:

* **взима текущата количка** от сесията или базата;
* **създава нова**, ако още няма.



Опитваме се да вземем CartItem (продукт в количката). Ако вече има такъв – се връща той. Ако няма – създава се нов.

* created = True → нов елемент в количката.
* created = False → вече съществува – ще увеличим количеството.



Ако продуктът вече е в количката, просто увеличаваме количеството с 1



Използваме Django съобщения (messages) за да уведомим потребителя, че добавянето е успешно. След добавянето, потребителят се пренасочва към началната страница (може да се смени с cart или products според дизайна).



Функция, която обработва заявка (request) и връща HTML страница с информация за текущата количка.



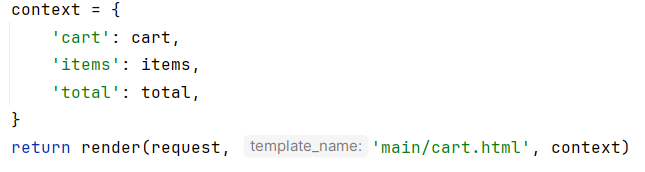
Извиква се помощна функция, която:

* Взима текущата количка на потребителя (от базата или от сесията);
* Ако няма количка – създава нова.

Това гарантира, че всеки потребител има количка, дори и без регистрация (ако се използва session).



Взимаме всички свързани обекти от типа CartItem, тоест – всички продукти в количката.

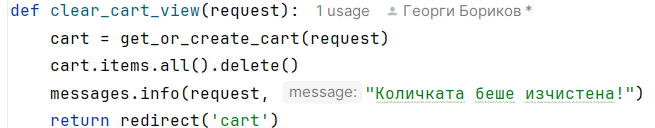


Създаваме речник с данни, който ще бъде предаден на шаблона cart.html. Така шаблонът ще може да достъпва:

* cart – количката;
* items – списък с продукти;
* total – общата сума.

Извикваме шаблона main/cart.html, който визуализира съдържанието на количката, използвайки данните от context.

## Изчистване на количката



* Взима количката (или създава нова, ако няма).
* Изтрива всички CartItem обекти, свързани с количката.
* Показва съобщение „Количката беше изчистена!“.
* Пренасочва потребителя обратно към страницата на количката.

# Заключение

Настоящият дипломен проект разглежда един от най-съществените аспекти в изграждането на съвременни уеб приложения – използването на форми за събиране, валидиране и обработка на потребителски данни. Чрез теоретичен анализ и практическа реализация беше доказано, че **Django Forms** представляват мощен и гъвкав инструмент за изграждане на сигурен и функционален потребителски интерфейс.

В теоретичната част бяха разгледани основните компоненти на Django Forms – обикновени форми и ModelForms, методите за автоматично и персонализирано валидиране, както и начините за вграждане на формите в шаблони и изгледи. Представени бяха и други алтернативни технологии, което даде възможност за сравнение и обосновка на избора на Django като основен фреймуърк за реализацията на проекта.

Практическата част демонстрира как събраните знания могат да бъдат приложени в реален казус – създаването на интерфейс за онлайн магазин. Реализирана бе функционалност за избор на стоки, добавяне в кошница и събиране на данни чрез динамична форма, интегрирана с база данни. Валидирането на входа бе осигурено както с вградени механизми, така и чрез персонализирани методи, което повиши надеждността на системата.

Извършената разработка потвърждава, че ученикът е усвоил умения за планиране, проектиране и реализация на уеб компоненти, които могат да бъдат приложени както в учебна, така и в реална работна среда. Освен това проектът показва способност за самостоятелно вземане на решения при избора на технологии и реализиране на функционалност.

В бъдеще проектът може да бъде разширен чрез добавяне на модули за потребителска регистрация, плащания, управление на поръчки и дори мобилна версия или REST API. По този начин дипломната работа не само изпълнява учебните цели, но поставя стабилна основа за надграждане и професионално развитие в областта на уеб програмирането.

# Литература:

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