

**Aldona Świrad, Patryk Wereszczak**

Aplikacje Internetowe

Calendarium – aplikacja wspomagająca organizację pracy

Opiekun projektu:

dr inż. Mirosław Mazurek  
mgr inż. Piotr Hadaj

Rzeszów, 2023

Spis treści

[1 Wstęp 4](#_Toc153878102)

[1.1 Opis modelu strony 4](#_Toc153878103)

[1.2 Cel i zastosowanie 4](#_Toc153878104)

[1.3 Opis koncepcji 5](#_Toc153878105)

[1.4 Opis logiki 6](#_Toc153878106)

[1.5 Opis strony – technologie 6](#_Toc153878107)

[1.6 Szkieletowanie 8](#_Toc153878108)

[1.6.1 Panel rejestracji 8](#_Toc153878109)

[1.6.2 Panel logowania 9](#_Toc153878110)

[1.6.3 Kalendarz 10](#_Toc153878111)

[1.6.4 Lista To Do 10](#_Toc153878112)

[1.7 Instalacja Django i innych podstawowych pakietów 10](#_Toc153878113)

[2 Kalendarz 12](#_Toc153878114)

[2.1 Model danych 12](#_Toc153878115)

[2.2 Widoki 13](#_Toc153878116)

[2.3 Templatki 14](#_Toc153878117)

[2.3.1 base.html 14](#_Toc153878118)

[2.3.2 index.html 18](#_Toc153878119)

[2.4 Wygląd kalendarza 19](#_Toc153878120)

[3 System logowania 19](#_Toc153878121)

[3.1 Ustawienia wstępne 19](#_Toc153878122)

[3.2 Ustawienia plików urls.py 19](#_Toc153878123)

[3.3 Views.py 21](#_Toc153878124)

[3.4 Forms 22](#_Toc153878125)

[3.5 Templates 24](#_Toc153878126)

[3.5.1 Index.html 24](#_Toc153878127)

[3.5.2 Login.html 24](#_Toc153878128)

[3.5.3 Signup.html 25](#_Toc153878129)

[3.6 Widok strony logowania i rejestracji 25](#_Toc153878130)

[3.7 Dodanie dodatkowych zabezpieczeń podczas logowania i rejestracji 26](#_Toc153878131)

[4 Lista ToDo 30](#_Toc153878132)

[4.1 Urls.py 31](#_Toc153878133)

[4.2 Models.py 32](#_Toc153878134)

[4.3 Views.py 33](#_Toc153878135)

[4.4 Templates 34](#_Toc153878136)

[4.4.1 tbase.html 34](#_Toc153878137)

[4.4.2 tindex.html 35](#_Toc153878138)

[4.4.3 Todo-list.html 36](#_Toc153878139)

[4.5 Wygląd listy todo 37](#_Toc153878140)

[5 Podsumowanie 37](#_Toc153878141)

[6 Literatura 39](#_Toc153878142)

# Wstęp

## Opis modelu strony

Projekt aplikacji internetowej skupia się na stworzeniu kompleksowej platformy wspomagającej organizację codziennych działań użytkowników. Oto rozwinięcie tematu projektu:

1. **Zarządzanie Kalendarzem:**
   * Dodawanie, edycja i usuwanie wydarzeń w kalendarzu.
   * Widok dzienny, tygodniowy, miesięczny dla lepszej organizacji czasu.
   * Powiadomienia o zbliżających się wydarzeniach.
2. **Lista Zadań Todo:**
   * Dodawanie nowych zadań do wykonania.
   * Oznaczanie zadań jako ukończone.
   * Priorytetyzacja zadań.
3. **Podgląd i Kontrolowana Modyfikacja:**
   * Podgląd postępów w realizacji zadań i wydarzeń.
   * Możliwość kontrolowanej modyfikacji, aktualizacji i usuwania.
4. **Elastyczność Dostępu:**
   * Dostęp do aplikacji z różnych urządzeń (komputer, tablet, smartfon).
   * Interfejs responsywny dla lepszej użyteczności.
5. **Bezpieczeństwo i Prywatność:**
   * Bezpieczne przechowywanie danych.
   * Opcje kontroli prywatności.

## Cel i zastosowanie

Stworzyliśmy stronę internetową, która wspiera naszą organizację działań poprzez zarządzanie kalendarzem i listą zadań todo. Nasza strona spełnia szereg celów i ma praktyczne zastosowania:

1. **Efektywne Zarządzanie Czasem:**
   * Pomaga nam w planowaniu i organizowaniu codziennych obowiązków.
   * Pozwala na optymalne wykorzystanie dostępnego czasu.
2. **Centralizacja Informacji:**
   * Dostarcza skonsolidowane miejsce do zarządzania kalendarzem i listą zadań.
   * Umożliwia łatwy dostęp do ważnych informacji.
3. **Zarządzanie Zadaniami:**
   * Daje nam możliwość kontrolowanej modyfikacji listy zadań zgodnie z bieżącymi potrzebami.
   * Umożliwia dodawanie, usuwanie i aktualizację zadań.
4. **Wsparcie w Planowaniu Wydarzeń:**
   * Ułatwia nam organizację spotkań, wydarzeń i ważnych dat.
   * Przypomina o nadchodzących ważnych terminach.
5. **Łatwy Dostęp z Dowolnego Miejsca:**
   * Daje nam możliwość korzystania ze strony z różnych urządzeń i lokalizacji.
   * Zapewnia dostęp do informacji nawet w podróży.
6. **Wsparcie dla Naszych Potrzeb:**
   * Jest dostosowana do różnych stylów naszej pracy i preferencji.
   * Pozwala na indywidualne dostosowanie strony do naszych potrzeb.
7. **Poprawa Produktywności:**
   * Zwiększa naszą efektywność poprzez lepsze zarządzanie czasem i zadaniami.
   * Minimalizuje chaos i ryzyko zapomnienia ważnych zadań.
8. **Wsparcie dla Planowania Długoterminowego:**
   * Pomaga nam w planowaniu działań na dłuższy okres.
   * Zapewnia spójność i ciągłość w naszym planowaniu.

W rezultacie, nasza strona internetowa stanowi kompleksowe narzędzie do organizacji naszego codziennego życia, zarządzania zadaniami i planowania przyszłych wydarzeń. Jej zastosowanie jest uniwersalne i dostosowane do naszych osobistych potrzeb.

## Opis koncepcji

Nasza aplikacja powstała w odpowiedzi na identyfikację konkretnych potrzeb biznesowych klienta oraz użytkowników końcowych. Kluczowe elementy koncepcji to:

1. **Zarządzanie Działaniami:**
   * *Potrzeby Biznesowe:* Efektywne zarządzanie codziennymi działaniami i obowiązkami.
   * *Potrzeby Użytkownika:* Prosta i intuicyjna platforma do planowania, organizowania oraz monitorowania postępów w realizacji zadań.
2. **Centralizacja Informacji:**
   * *Potrzeby Biznesowe:* Skonsolidowane miejsce do przechowywania informacji o wydarzeniach i zadaniach.
   * *Potrzeby Użytkownika:* Łatwy dostęp do ważnych danych z jednego miejsca.
3. **Podgląd i Monitorowanie Postępów:**
   * *Potrzeby Biznesowe:* Potrzeba monitorowania postępów prac w celu zoptymalizowania efektywności.
   * *Potrzeby Użytkownika:* Możliwość śledzenia postępów w realizacji zadań i planowanych wydarzeń.
4. **Zarządzanie Zadaniami:**
   * *Potrzeby Biznesowe:* Skuteczne zarządzanie zadaniami i projektami.
   * *Potrzeby Użytkownika:* Kontrolowane dodawanie, usuwanie i aktualizacja zadań z łatwością.
5. **Wsparcie w Planowaniu Wydarzeń:**
   * *Potrzeby Biznesowe:* Planowanie spotkań, wydarzeń i zadań z wyprzedzeniem.
   * *Potrzeby Użytkownika:* Wygodne i spersonalizowane narzędzie do organizacji planu dnia, tygodnia, czy miesiąca.
6. **Łatwy Dostęp z Dowolnego Miejsca:**
   * *Potrzeby Biznesowe:* Elastyczność dostępu do informacji z różnych lokalizacji.
   * *Potrzeby Użytkownika:* Dostępność aplikacji na różnych urządzeniach, umożliwiająca korzystanie z niej w podróży.
7. **Wsparcie dla Różnych Stylów Pracy:**
   * *Potrzeby Biznesowe:* Dostosowanie się do różnych modeli pracy i preferencji użytkowników.
   * *Potrzeby Użytkownika:* Elastyczność w dostosowywaniu aplikacji do własnych preferencji i potrzeb.

Koncepcja aplikacji została stworzona z myślą o zaspokojeniu konkretnych wymagań biznesowych klienta oraz dostarczeniu użytkownikowi końcowemu narzędzia, które nie tylko ułatwia organizację, ale także wpływa pozytywnie na efektywność i satysfakcję z codziennego zarządzania obowiązkami.

## Opis logiki

Aby spełnić zarówno wymagania biznesowe, jak i użytkowe, nasza aplikacja opiera się na spójnej logice, uwzględniającej interakcje użytkownika, zarządzanie danymi oraz dostarczanie intuicyjnych funkcji. Poniżej przedstawiamy kluczowe elementy logiki naszej aplikacji:

1. **Zarządzanie Zadaniami i Wydarzeniami:**
   * Użytkownik może dodawać, usuwać i aktualizować zadania oraz wydarzenia.
   * Każde zadanie lub wydarzenie ma przypisane informacje, takie jak data, opis, priorytet, itp.
2. **Centralne Miejsce Przechowywania Danych:**
   * Wszystkie dane, w tym zadania i wydarzenia, są przechowywane w centralnej bazie danych.
   * Dostęp do danych jest chroniony, zgodnie z zasadami bezpieczeństwa.
3. **Podgląd Postępów:**
   * Użytkownik ma dostęp do podglądu postępów realizacji zadań oraz planowanych wydarzeń.
   * Możliwość oznaczania zadań jako zakończone oraz monitorowania zbliżających się terminów.
4. **Planowanie i Kalendarz:**
   * Użytkownik może planować wydarzenia na różne okresy czasu (dzień, tydzień, miesiąc).
   * Kalendarz dostarcza czytelnej reprezentacji planu, umożliwiającej łatwe zarządzanie czasem.
5. **Elastyczność Dostępu:**
   * Aplikacja jest responsywna i dostosowuje się do różnych urządzeń, umożliwiając użytkownikowi korzystanie z niej na komputerze, tablecie czy smartfonie.
6. **Zabezpieczenia i Bezpieczeństwo:**
   * Bezpieczeństwo danych jest priorytetem; aplikacja stosuje odpowiednie zabezpieczenia, w tym uwierzytelnianie i autoryzację.
   * Dane przechowywane w bazie są regularnie backupowane.
7. **Intuicyjna Nawigacja:**
   * Interfejs użytkownika jest zaprojektowany z myślą o intuicyjnej nawigacji, umożliwiającej łatwe korzystanie nawet dla nowych użytkowników.

Nasza aplikacja, oparta na tej logice, ma na celu efektywne spełnienie zarówno oczekiwań biznesowych, jak i potrzeb użytkowników końcowych, zapewniając kompleksową i intuicyjną platformę do zarządzania codziennymi działaniami.

## Opis strony – technologie

Aplikacja została stworzona z wykorzystaniem różnych technologii, umożliwiając efektywne zarządzanie czasem i zadaniami. Poniżej opisujemy, jakie technologie zostały użyte i jakie potencjalne ograniczenia mogą wystąpić na etapie wdrożenia u klienta:

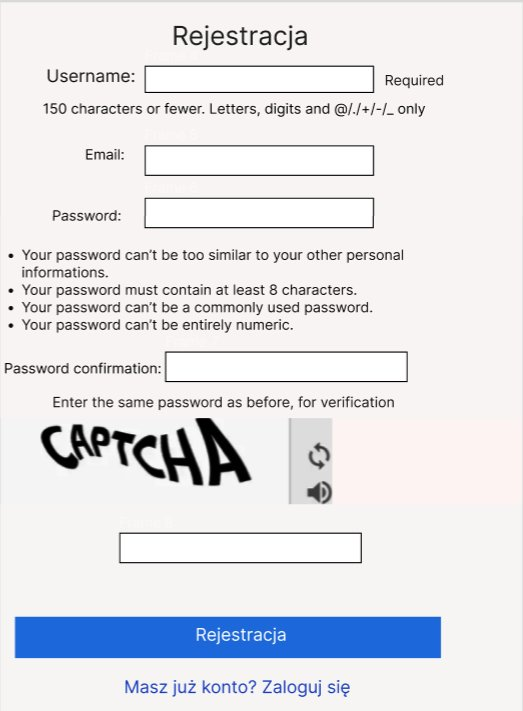
1. **VS Code (Visual Studio Code):**
   * **Opis Wykorzystania:** Edytor tekstu Visual Studio Code, stworzony przez Microsoft, był używany do kodowania aplikacji. Dzięki swojej lekkości, obszernej funkcjonalności, oraz bogatej ofercie rozszerzeń, umożliwia efektywną pracę programistyczną.
   * **Ograniczenia:** Brak, jest popularny i łatwo dostępny dla wielu systemów operacyjnych.
2. **HTML (Hypertext Markup Language):**
   * **Opis Wykorzystania:** HTML został wykorzystany do strukturyzowania zawartości strony internetowej. Zapewnia logiczną organizację treści, co jest kluczowe dla responsywnego i przejrzystego interfejsu użytkownika.
   * **Ograniczenia:** Jako język znaczników, HTML nie obsługuje logiki programistycznej; dla tego celu stosuje się języki skryptowe, takie jak Python.
3. **Python:**
   * **Opis Wykorzystania:** Python, jako łatwy w nauce i czytelny język programowania, został użyty jako główny język programowania w backendzie za pomocą frameworka Django.
   * **Ograniczenia:** Ograniczenia Pythona zwykle związane są z wydajnością w porównaniu do niektórych innych języków programowania, szczególnie w obszarach wymagających niskopoziomowej manipulacji pamięci.
4. **Bootstrap:**
   * **Opis Wykorzystania:** Bootstrap, framework front-endowy, został zastosowany do zapewnienia responsywnego i estetycznego designu strony. Ułatwia szybkie projektowanie interfejsu użytkownika.
   * **Ograniczenia:** Używanie Bootstrap może wprowadzić pewne ograniczenia w dostosowaniu designu do unikalnych wymagań projektu.
5. **Django:**
   * **Opis Wykorzystania:** Django, framework webowy dla Pythona, pozwala na szybkie i efektywne tworzenie aplikacji internetowych. Oferuje wiele gotowych rozwiązań, takich jak system kont użytkowników czy obsługa baz danych.
   * **Ograniczenia:** Wdrożenie wymaga zazwyczaj serwera obsługującego aplikacje oparte na Django. Niektóre funkcje mogą wymagać zaawansowanego dostosowania.
6. **Git/GitHub:**
   * **Opis Wykorzystania:** Git jako system kontroli wersji i GitHub jako platforma hostingowa są używane do śledzenia i zarządzania kodem źródłowym projektu.
   * **Ograniczenia:** Wprowadzenie systemu kontroli wersji może wymagać nauki przez zespół pracowników, ale długoterminowe korzyści są znaczne.
7. **Crispy:**
   * **Opis Wykorzystania:** Crispy to biblioteka dla Django, która ułatwia eleganckie formatowanie formularzy, co poprawia doświadczenia użytkownika.
   * **Ograniczenia:** Konieczność integracji z istniejącym systemem formularzy Django, ale korzyści obejmują poprawę estetyki i przejrzystości formularzy.
8. **htmx:**
   * **Opis Wykorzystania:** htmx to biblioteka JavaScript, która umożliwia dodawanie interakcji do stron internetowych bez potrzeby dużego kodu JavaScript. Ułatwia dynamiczną aktualizację treści bez przeładowywania całej strony.
   * **Ograniczenia:** Niektóre funkcje htmx mogą wymagać świadomej implementacji, a brak zrozumienia może prowadzić do błędów w interakcjach.

Wdrożenie aplikacji może napotkać pewne wyzwania, takie jak dostosowanie do środowiska klienta, konieczność konfiguracji serwera, czy integracja z systemami już istniejącymi. Jednak staranność w doborze technologii oraz świadome zarządzanie ograniczeniami może zminimalizować te potencjalne trudności.

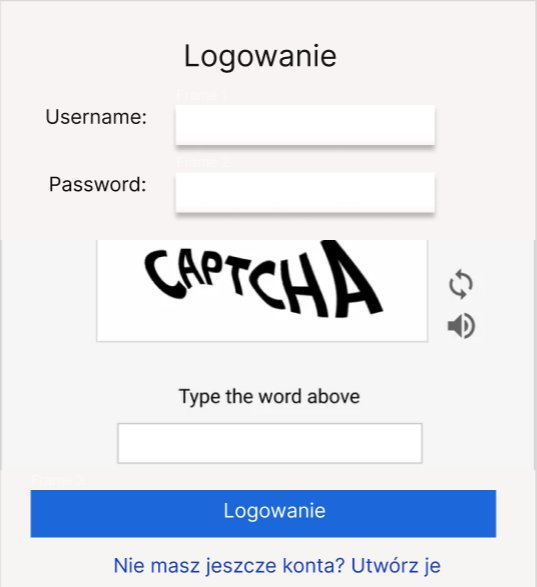
## Szkieletowanie

Przygotowaliśmy koncepcyjne wizualizację, jak chcielibyśmy, aby nasz projekt wyglądał. Do tworzenia wizualizacji korzystaliśmy z **www.figma.com**.

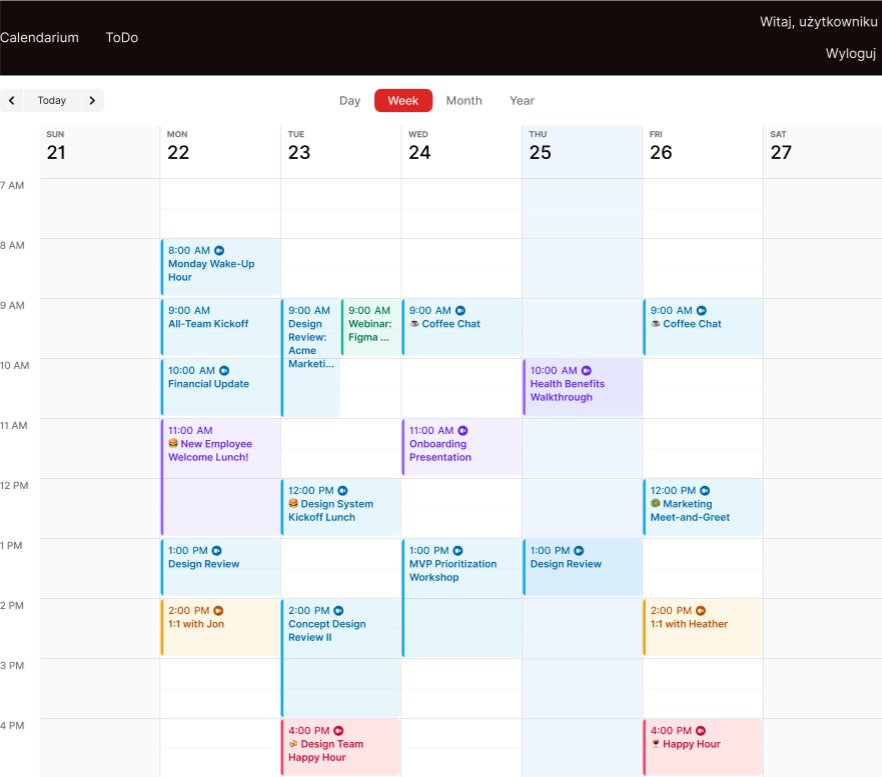
### Panel rejestracji



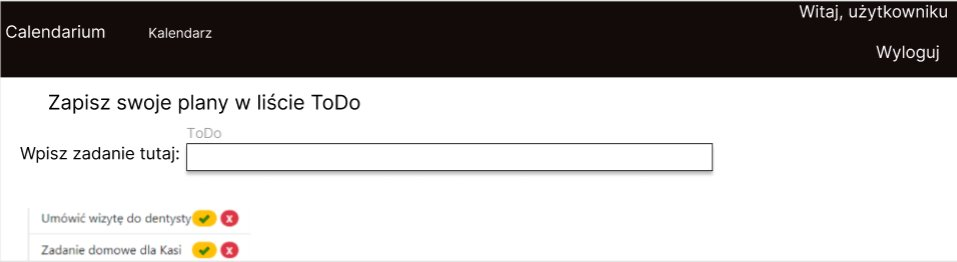
### Panel logowania



### Kalendarz



### Lista To Do



## Instalacja Django i innych podstawowych pakietów

Aby rozpocząć pracę z frameworkiem Django należy zainstalować odpowiednie pakiety. Poprzez polecenie:

|  |
| --- |
| python -m pip install Django |

instalujemy Django. Następnie, do stworzenia projektu korzystamy z polecenia:



W następnym kroku tworzymy aplikacje/komponenty/foldery, które będziemy wykorzystywać w naszej aplikacji, komendą:







Po tych działaniach nasza aplikacja ma taką strukturę:  
Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Struktura ta powinna pomóc nam w wygodny sposób korzystać z aplikacji. Dodatkowo, aby te komponenty mogły być widoczne dla projektu trzeba dodać je do wykazu zainstalowanych aplikacji w pliku setting.py.  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Aby móc rozpocząć działanie naszego projektu należy wykonać komendy.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

# Kalendarz

Głównym komponentem naszej aplikacji będzie kalendarz, który pomoże w uporządkowaniu planowanych wydarzeń.

Najpierw tworzymy model, czyli najważniejsze informacje jakie będą przechowywane, aby nasz kalendarz mógł funkcjonować.

## Model danych

Tworzymy model danych w Django dla wydarzeń. Poniżej przedstawiam krótkie wyjaśnienie każdego elementu:

1. Importowanie modułu models z Django:

Importuje moduł models z frameworka Django, który umożliwia definiowanie struktury bazy danych.

1. Tworzenie klasy Events dziedziczącej po models.Model:

Definiuje klasę Events, która reprezentuje model danych. Klasa ta dziedziczy po models.Model, co oznacza, że będzie używana do tworzenia tabeli w bazie danych.

1. Definicja pól modelu:

Określa różne pola, które będą przechowywane dla każdego rekordu w tabeli. W tym przypadku są to:

* id: Automatycznie inkrementowane pole identyfikujące, będące kluczem głównym modelu.
* name: Pole tekstu o maksymalnej długości 255 znaków, reprezentujące nazwę wydarzenia.
* start: Pole daty i czasu reprezentujące datę rozpoczęcia wydarzenia.
* end: Pole daty i czasu reprezentujące datę zakończenia wydarzenia.

1. Definiowanie klasy Meta:

Określa ustawienia metadanych modelu. W tym przypadku jest to db\_table, które ustala nazwę tabeli w bazie danych na "tblevents".

Ten model reprezentuje strukturę danych dla wydarzeń. Każde wydarzenie będzie przechowywane jako rekord w bazie danych, a pola modelu określają, jakie informacje są związane z każdym wydarzeniem. Model ten jest używany w aplikacji Django do przechowywania i pobierania danych o wydarzeniach.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

## Widoki

Tworzymy widoki Django w aplikacji, które obsługują żądania HTTP związane z wydarzeniami. Poniżej opisuję każdy widok:

1. Widok index:

* Pobiera wszystkie wydarzenia z bazy danych (Events.objects.all()).
* Przekazuje wydarzenia jako kontekst do szablonu HTML.
* Renderuje stronę główną (index.html) z wydarzeniami.

1. Widok all\_events:

* Pobiera wszystkie wydarzenia z bazy danych.
* Konwertuje dane o wydarzeniach na format JSON, który zostanie zwrócony jako odpowiedź HTTP.
* Używa JsonResponse do zwrócenia danych w formacie JSON.

1. Widok add\_event:

* Pobiera parametry start, end i title ządania GET.
* Tworzy nowy obiekt Events na podstawie przekazanych parametrów.
* Zapisuje nowe wydarzenie do bazy danych.
* Zwraca pusty obiekt JSON jako potwierdzenie.

1. Widok update:

* Pobiera parametry start, end, title i id z żądania GET.
* Pobiera istniejące wydarzenie z bazy danych na podstawie przekazanego id.
* Aktualizuje atrybuty wydarzenia zgodnie z nowymi danymi.
* Zapisuje zmodyfikowane wydarzenie do bazy danych.
* Zwraca pusty obiekt JSON jako potwierdzenie.

1. Widok remove:

* Pobiera parametr id z żądania GET.
* Pobiera istniejące wydarzenie z bazy danych na podstawie przekazanego id.
* Usuwa wydarzenie z bazy danych.
* Zwraca pusty obiekt JSON jako potwierdzenie.

Widoki te są przeznaczone do obsługi różnych operacji na wydarzeniach, takich jak wyświetlanie, dodawanie, aktualizowanie i usuwanie. Są używane w połączeniu z odpowiednimi adresami URL w pliku urls.py. Widoki korzystają z modelu Events zdefiniowanego w pliku models.py i wykorzystują szablony HTML do renderowania stron.

## Templatki

Do stworzenia templatek użyte były:

Bootstrap 5  
https://getbootstrap.com/docs/5.0/getting-started/introduction/  
https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css  
  
Jquery  
https://jquery.com/download/  
CDN : jsDelivr CDN  
https://www.jsdelivr.com/package/npm/jquery  
https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.min.js  
  
https://fullcalendar.io/  
CND : <https://cdnjs.com/libraries/fullcalendar/3.9.0>

W katalogu events utworzony został folder templates, a w nim dwa pliki index.html i base.html, które odpowiedzialne są na wygląd i funkcjonalność kalendarza.

### base.html

W pliku tym została utworzona pełna strona HTML, która korzysta z biblioteki Bootstrap 5 oraz FullCalendar.io do stworzenia interaktywnego kalendarza.

1. Bootstrap i FullCalendar:

* W sekcji <head> dołączane są odpowiednie pliki CSS i JavaScript z bibliotek Bootstrap 5 oraz FullCalendar.io za pomocą linków CDN. Są to pliki służące do stylizacji strony i obsługi interaktywnego kalendarza.

1. Struktura strony:

* W sekcji <body> znajduje się kontener Bootstrapa (<div class="container">) zawierający jeden wiersz (<div class="row">). W tym wierszu znajduje się nagłówek "Calendarium" (<h4>Calendarium</h4>) oraz sekcja, w której będzie renderowany kalendarz ({% block content %}{% endblock %}).

1. Skrypty JavaScript:

* Poniżej struktury strony znajduje się skrypt JavaScript, który inicjalizuje kalendarz za pomocą FullCalendar.io. Skrypt ten definiuje różne opcje i funkcje obsługujące interakcję użytkownika z kalendarzem.
* Określa nagłówek kalendarza, umożliwiając nawigację między miesiącami, tygodniami i dniem.
* Ustawia ścieżkę do endpointu /all\_events, z którego kalendarz pobiera wydarzenia.
* Obsługuje dodawanie nowego wydarzenia poprzez interaktywne zaznaczanie obszaru na kalendarzu i wprowadzanie tytułu.
* Obsługuje zmiany rozmiaru wydarzenia (eventResize), przesunięcia wydarzenia (eventDrop) oraz kliknięcie na wydarzenie (eventClick).
* W przypadku dodania, aktualizacji lub usunięcia wydarzenia, wysyła zapytanie AJAX do odpowiednich endpointów na serwerze, aby zaktualizować dane.

1. Wykorzystanie Jinja:
2. W sekcji {% block content %}{% endblock %} znajduje się blok, który może zostać nadpisany w szablonie potomnym (template) za pomocą Jinja, jeśli taki blok jest zdefiniowany. W tym przypadku, będzie to obszar, w którym kalendarz zostanie umieszczony.

Podsumowując, ten kod tworzy interaktywną stronę internetową zawierającą kalendarz, który umożliwia dodawanie, aktualizację, oraz usuwanie wydarzeń poprzez komunikację z serwerem przy użyciu technologii AJAX.

<!DOCTYPE html>

<!-- Bootstrap5 -->

<html>

<head>

    <title>Calendarium</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">

    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/fullcalendar/3.9.0/fullcalendar.css" />

    <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.24.0/moment.min.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/fullcalendar/3.9.0/fullcalendar.js"></script>

</head>

<body>

<div class="container">

 <div class="row">

    <h4>Calendarium</h4>

    {% block content %}{% endblock %}

 </div>

</div>

<script>

    // Fullcalendar.io

   $(document).ready(function () {

       var calendar = $('#calendar').fullCalendar({

        // locale: 'pl',

        monthNames: ['Styczeń', 'Luty', 'Marzec', 'Kwiecień', 'Maj', 'Czerwiec', 'Lipiec', 'Sierpień', 'Wrzesień', 'Październik', 'Listopad', 'Grudzień'],

        monthNamesShort: ['Sty', 'Lut', 'Mar', 'Kwi', 'Maj', 'Cze', 'Lip', 'Sie', 'Wrz', 'Paź', 'Lis', 'Gru'],

        dayNames: ['Poniedziałek', 'Wtorek', 'Środa', 'Czwartek', 'Piątek', 'Sobota', 'Niedziela'],

        dayNamesShort: ['Pn', 'Wt', 'Śr', 'Cz', 'Pt', 'Sb', 'Nd'],

           header: {

               left: 'prev,next today',

               center: 'title',

               right: 'month,agendaWeek,agendaDay'

           },

           buttonText: {

            today: 'Dziś',

            month: 'Miesiąc',

            week: 'Tydzień',

            day: 'Dzień',

            },

           events: '/events/all\_events',

           selectable: true,

           selectHelper: true,

           editable: true,

           eventLimit: true,

           select: function (start, end, allDay) {

               var title = prompt("Wpisz tytuł wydarzenia");

               if (title) {

                   var start = $.fullCalendar.formatDate(start, "Y-MM-DD HH:mm:ss");

                   var end = $.fullCalendar.formatDate(end, "Y-MM-DD HH:mm:ss");

                   $.ajax({

                       type: "GET",

                       url: '/events/add\_event',

                       data: {'title': title, 'start': start, 'end': end},

                       dataType: "json",

                       success: function (data) {

                           calendar.fullCalendar('refetchEvents');

                           alert("Dodano wydarzenie");

                       },

                       error: function (data) {

                           alert('Wystąpił problem, spróbuj ponownie później.');

                       }

                   });

               }

           },

           eventResize: function (event) {

               var start = $.fullCalendar.formatDate(event.start, "Y-MM-DD HH:mm:ss");

               var end = $.fullCalendar.formatDate(event.end, "Y-MM-DD HH:mm:ss");

               var title = event.title;

               var id = event.id;

               $.ajax({

                   type: "GET",

                   url: '/events/update',

                   data: {'title': title, 'start': start, 'end': end, 'id': id},

                   dataType: "json",

                   success: function (data) {

                       calendar.fullCalendar('refetchEvents');

                       alert('Wydarzenie zaktualizowane');

                   },

                   error: function (data) {

                       alert('Wystąpił problem, spróbuj ponownie później.');

                   }

               });

           },

           eventDrop: function (event) {

               var start = $.fullCalendar.formatDate(event.start, "Y-MM-DD HH:mm:ss");

               var end = $.fullCalendar.formatDate(event.end, "Y-MM-DD HH:mm:ss");

               var title = event.title;

               var id = event.id;

               $.ajax({

                   type: "GET",

                   url: '/events/update',

                   data: {'title': title, 'start': start, 'end': end, 'id': id},

                   dataType: "json",

                   success: function (data) {

                       calendar.fullCalendar('refetchEvents');

                       alert('Wydarzenie zaktualizowane');

                   },

                   error: function (data) {

                       alert('Wystąpił problem, spróbuj ponownie później.');

                   }

               });

           },

           eventClick: function (event) {

               if (confirm("Czy na pewno chcesz to usunąć?")) {

                   var id = event.id;

                   $.ajax({

                       type: "GET",

                       url: '/events/remove',

                       data: {'id': id},

                       dataType: "json",

                       success: function (data) {

                           calendar.fullCalendar('refetchEvents');

                           alert('Wydarzenie usunięte');

                       },

                       error: function (data) {

                           alert('Wystąpił problem, spróbuj ponownie później.');

                       }

                   });

               }

           },

       });

   });

</script>

</body>

</html>

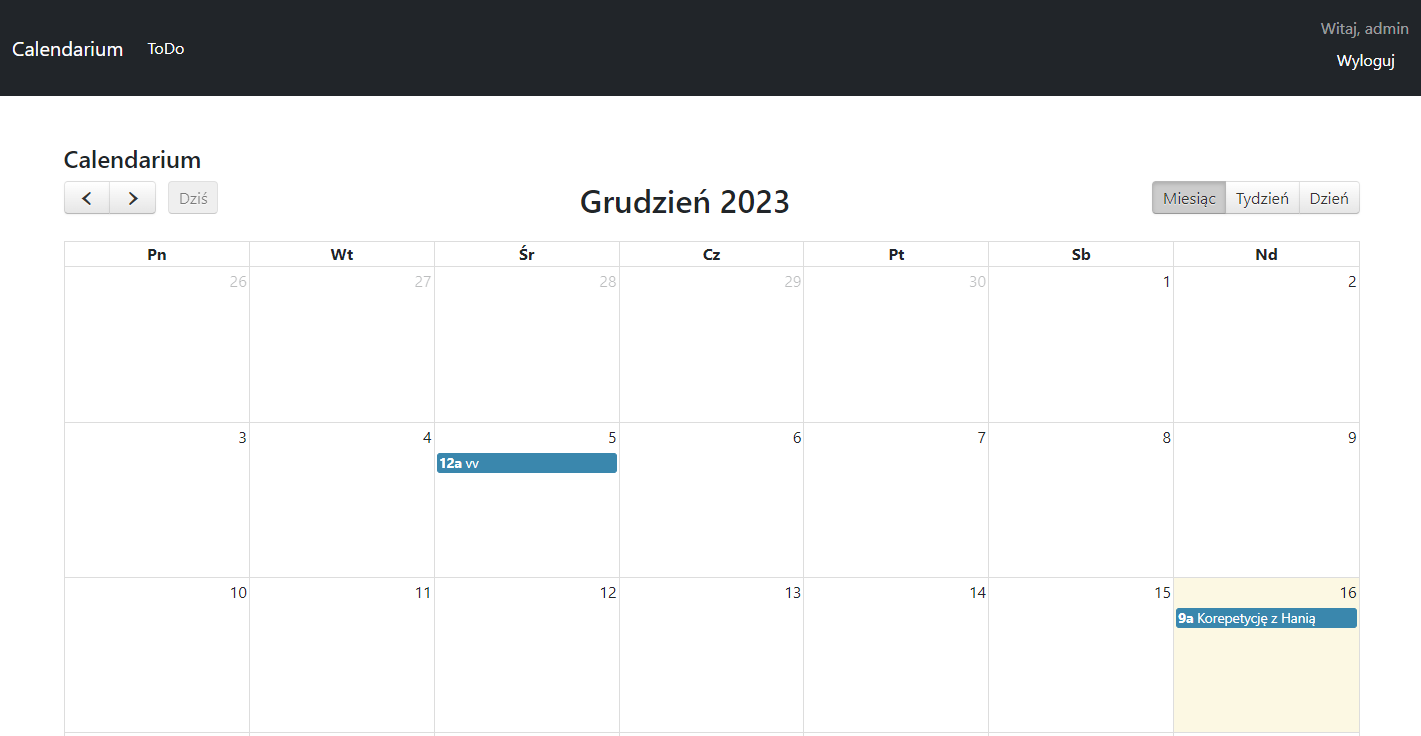
### index.html

1. {% extends "base.html" %}: W tej linii kodu używane jest polecenie extends, które wskazuje, że ten szablon rozszerza inny szablon o nazwie "base.html". To oznacza, że kod zawarty w tym szablonie zostanie dołączony do kodu zdefiniowanego w szablonie bazowym.
2. {% block title %} Calendarium (Create, Read, Update and Delete){% endblock title %}: Ta sekcja definiuje blok o nazwie "title". Blok ten będzie używany do dynamicznego wstawiania tytułu strony. W tym przypadku, tytuł strony to "Calendarium (Create, Read, Update and Delete)".
3. {% block content %} i {% endblock content %}: Pomiędzy tymi znacznikami znajduje się blok o nazwie "content". Jest to obszar, w którym definiowany jest główny content strony. W tym konkretnym przypadku, znajduje się tu <div class="col-md-12"> zawierający <div id='calendar'></div>, co sugeruje, że strona ma zawierać kalendarz.
4. {% extends "base.html" %}
6. {% block title %} Calendarium {% endblock title %}
8. {% block content %}
9. <div class="col-md-12">
10. <div id='calendar'></div>
11. </div>
12. {% endblock content %}

Cały ten kod stanowi szablon strony internetowej, który może być dynamicznie generowany i wstawiany w szablon bazowy ("base.html") poprzez mechanizm dziedziczenia zaimplementowany w systemie szablonów.

## Wygląd kalendarza

Aby móc lepiej poruszać się po stronie dodałam nawigację, posiadającą linki do innych podstron. Dodatkowo zmieniłam język na język użytkownika (polski).



# System logowania

Źródło: https://medium.com/@devsumitg/django-auth-user-signup-and-login-7b424dae7fab

## Ustawienia wstępne

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

## Ustawienia plików urls.py

1. calendarium/urls.py

Dodajemy odnośnik do pliku rols.py w folderze users.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

1. calendarium/users/urls.py

Ten kod definiuje zestaw wzorców adresów URL (URL patterns) dla Twojej aplikacji Django, w tym ścieżki do różnych widoków. Poniżej znajduje się opis każdej ścieżki w urlpatterns:

1. path('', views.index, name='home'):
   * Określa, że dla pustego ciągu znaków (czyli głównego adresu URL), ma być używany widok o nazwie index zdefiniowany w pliku views.py.
   * name='home': Nadaje nazwę tej ścieżce, co umożliwia odwoływanie się do niej w szablonach lub kodzie Pythona poprzez to oznaczenie.

2. path('login/', views.user\_login, name='login'):

* + Określa, że dla ścieżki 'login/' ma być używany widok user\_login zdefiniowany w pliku views.py.
  + name='login': Nadaje nazwę tej ścieżce.

2. path('signup/', views.user\_signup, name='signup'):

* + Określa, że dla ścieżki 'signup/' ma być używany widok user\_signup zdefiniowany w pliku views.py.
  + name='signup': Nadaje nazwę tej ścieżce.

3. path('logout/', views.user\_logout, name='logout'):

* + Określa, że dla ścieżki 'logout/' ma być używany widok user\_logout zdefiniowany w pliku views.py.
  + name='logout': Nadaje nazwę tej ścieżce.

Te ścieżki definiują, które widoki mają być używane dla określonych adresów URL. Znaczniki name są używane w celu identyfikacji ścieżek w kodzie aplikacji, co ułatwia ich odwoływanie się bez konieczności podawania całego adresu URL w kodzie. Na przykład, w szablonie można użyć {% url 'login' %} do wygenerowania URL dla ścieżki o nazwie 'login'.

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Czcionka

Opis wygenerowany automatycznie

## Views.py

Fragment kodu, który dostarczyłeś, reprezentuje plik views.py w Django i zawiera różne widoki do uwierzytelniania użytkowników oraz zarządzania kontami. Poniżej przedstawiam opis każdej funkcji:

1. index(request):

Ten widok reprezentuje stronę główną Twojej aplikacji. Renderuje szablon 'index.html' i zwraca go jako odpowiedź.

1. user\_signup(request):

Ten widok obsługuje stronę rejestracji. Sprawdza, czy metoda żądania to POST, co oznacza przesłanie formularza. Jeśli tak, sprawdza poprawność danych formularza za pomocą UserCreationForm. Jeżeli formularz jest poprawny, zapisuje użytkownika i przekierowuje go do strony logowania. Jeśli metoda żądania to GET, tworzy nową instancję UserCreationForm i renderuje szablon 'signup.html', przekazując formularz jako kontekst.

1. user\_login(request):

Ten widok obsługuje stronę logowania. Sprawdza, czy metoda żądania to POST, co oznacza przesłanie formularza. Sprawdza poprawność danych formularza za pomocą LoginForm. Jeżeli formularz jest poprawny, pobiera nazwę użytkownika i hasło z oczyszczonych danych. Następnie uwierzytelnia użytkownika za pomocą funkcji authenticate() i loguje go za pomocą login(). Jeśli uwierzytelnianie zakończy się sukcesem, przekierowuje go do strony głównej. Jeśli metoda żądania to GET, tworzy nową instancję LoginForm i renderuje szablon 'login.html', przekazując formularz jako kontekst.

1. user\_logout(request):

Ten widok obsługuje funkcjonalność wylogowywania. Wywołuje funkcję logout() dostarczoną przez Django, aby wylogować użytkownika, a następnie przekierowuje go do strony logowania.

Kod zawiera również instrukcje importu różnych modułów i formularzy Django (render, redirect, authenticate, login, logout, UserCreationForm i LoginForm). Importy te są niezbędne do poprawnego działania widoków.

Ogólnie rzecz biorąc, ten kod przedstawia podstawową implementację funkcji rejestracji, logowania i wylogowywania użytkowników przy użyciu Django, wraz z odpowiednimi formularzami. Możesz zintegrować te widoki w swoim projekcie Django, aby umożliwić uwierzytelnianie użytkowników i zarządzanie kontami.

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login, logout

from .forms import UserCreationForm, LoginForm

# Create your views here.

# Home page

def index(request):

    return render(request, 'index.html')

# signup page

def user\_signup(request):

    if request.method == 'POST':

        form = UserCreationForm(request.POST)

        if form.is\_valid():

            form.save()

            return redirect('login')

    else:

        form = UserCreationForm()

    return render(request, 'signup.html', {'form': form})

# login page

def user\_login(request):

    if request.method == 'POST':

        form = LoginForm(request.POST)

        if form.is\_valid():

            username = form.cleaned\_data['username']

            password = form.cleaned\_data['password']

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

            if user:

                login(request, user)

                return redirect('home')

    else:

        form = LoginForm()

    return render(request, 'login.html', {'form': form})

# logout page

def user\_logout(request):

    logout(request)

    return redirect('login')

## Forms

Tworzymy plik forms.py w katalogu users.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

class SignupForm(UserCreationForm):

    class Meta:

        model = User

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

class LoginForm(forms.Form):

    username = forms.CharField()

    password = forms.CharField(widget=forms.PasswordInput)

Fragment kodu, przedstawia moduł formularzy w Django, który definiuje dwie formularze: SignupForm i LoginForm. Te formularze są wykorzystywane do funkcji rejestracji i logowania użytkownika w aplikacji Django.

1. SignupForm:

Dziedziczy po UserCreationForm, wbudowanym formularzu Django przeznaczonym specjalnie do rejestracji użytkownika.

UserCreationForm dostarcza pola dla nazwy użytkownika (username), hasła (password1), oraz potwierdzenia hasła (password2).

Klasa Meta określa model do użycia, którym jest domyślny model użytkownika Django, importowany z django.contrib.auth.models.

Atrybut fields wymienia pola, które powinny być uwzględnione w formularzu, a mianowicie 'username', 'password1' i 'password2'.

1. LoginForm:

Standardowy formularz Django używany do logowania użytkownika.

Nie dziedziczy bezpośrednio po żadnej konkretnej klasie formularza Django.

Definiuje dwa pola, 'username' i 'password', za pomocą metody forms.CharField().

Pole 'password' jest renderowane jako pole wprowadzania hasła dzięki argumentowi widget=forms.PasswordInput.

Te formularze mogą być wykorzystane w widokach Django do obsługi procesów rejestracji i uwierzytelniania użytkownika. Zapewniają one wygodny sposób generowania formularzy HTML z odpowiednimi polami i walidacją.

## Templates

Tworzymy ścieżkę:

**calendarium/users/templates/users/dowolny\_plik.html**

Do odpowiedniej obsługi tworzymy pliki html:

Obraz zawierający tekst, Czcionka, zrzut ekranu, numer

Opis wygenerowany automatycznie

### Index.html

{% if request.user.is\_authenticated %}

  <p>{{ request.user.username }}</p>

  <a href="{% url 'logout' %}">Logout</a>

{% else %}

  <a href="{% url 'login' %}">Login</a>

  <a href="{% url 'signup' %}">Signup</a>

{% endif %}

<h1>Welcome!</h1>

Podstawowy interfejs strony startowej:

Obraz zawierający tekst, zrzut ekranu, Czcionka, biały

Opis wygenerowany automatycznie

### Login.html

<h1>Login</h1>

<form method="POST">

    {% csrf\_token %}

    {{ form.as\_p }}

    <button type="submit">Login</button>

    <a href="{% url 'signup' %}">Dont have Account Create</a>

</form>

Podstawowy interfejs logowania:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

### Signup.html

<h1>Signup</h1>

<form method="POST">

    {% csrf\_token %}

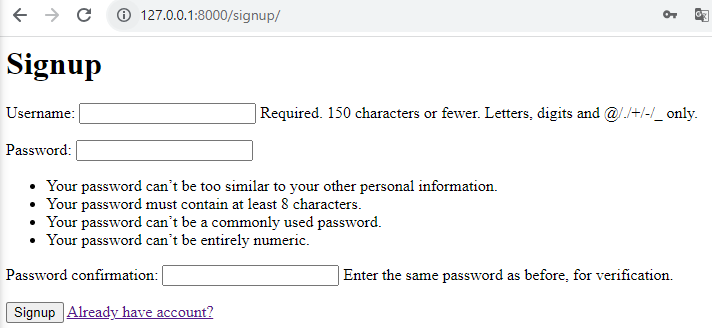
    {{ form.as\_p }}

    <button type="submit">Signup</button>

    <a href="{% url 'login' %}">Already have account?</a>

</form>

Podstawowy interfejs rejestracji:



## Widok strony logowania i rejestracji

Aby móc lepiej poruszać się po stronie dodałam nawigację, posiadającą linki do innych podstron. Dodatkowo zmieniłam język na język użytkownika (polski).

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, design

Opis wygenerowany automatycznie

## Dodanie dodatkowych zabezpieczeń podczas logowania i rejestracji

Jedną z form zabezpieczeń może być popularna captcha, utrudniająca np.: tworzenie nowych kont botom itp. Zaimplementowaliśmy takie rozwiązanie w naszej aplikacji.

Instajacja **django-simple-captcha:** **Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie**

Dodanie do settings.py:

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, wyświetlacz

Opis wygenerowany automatycznie**

Zmieniliśmy możliwość odczytywania słów na rozwiązywanie prostych równań matematycznych oraz ustawiliśmy czas na rozwiązanie na 2 minuty.

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

Calendarium/urls.py

**Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie**

Koniecznie trzeba dodać:

path('captcha/', include('captcha.urls')),

gdyż stąd renderuje obrazki.

Zmiany w forms.py

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

from captcha.fields import CaptchaField

class SignupForm(UserCreationForm):

    captcha = CaptchaField()

    email = forms.EmailField(required=True, max\_length=50)

    # password1 = forms.CharField(

    #     label = "Password",

    #     widget=forms.PasswordInput,

    #     required=True,

    #     max\_length=100,

    #     min\_length= 8,

    # )

    # password2 = forms.CharField(

    #     label = "Password ",

    #     widget=forms.PasswordInput,

    #     required=True,

    #     max\_length=100,

    #     min\_length= 8,

    # )

    class Meta:

        model = User

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

class LoginForm(forms.Form):

    username = forms.CharField()

    email = forms.CharField()

    password = forms.CharField(widget=forms.PasswordInput)

    captcha = CaptchaField()

Ostateczny widok logowania i rejestracji:

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

# Lista ToDo

Korzystanie z listy "to-do" (ang. to-do list) ma wiele korzyści, zarówno w życiu codziennym, jak i w obszarze zarządzania projektem czy pracy zawodowej. Oto niektóre z głównych korzyści z korzystania z listy "to-do":

1. **Organizacja i Planowanie:**
   * Pomaga zorganizować zadania i planować ich wykonanie.
   * Umożliwia śledzenie postępu w wykonywaniu zadań.
2. **Priorytetyzacja:**
   * Pozwala na określenie priorytetów wśród różnych zadań.
   * Ułatwia koncentrację na najważniejszych i najbardziej pilnych zadaniach.
3. **Zapominanie:**
   * Zapobiega zapominaniu o ważnych zadaniach i terminach.
   * Pomaga w skupieniu się na tym, co jest aktualnie istotne.
4. **Redukcja Stresu:**
   * Pomaga w redukcji stresu poprzez uporządkowanie obowiązków.
   * Pozwala uniknąć uczucia przytłoczenia nadmiarem zadań.
5. **Efektywność i Produktywność:**
   * Wspiera efektywne zarządzanie czasem i zadaniami.
   * Zachęca do skupienia uwagi na jednym zadaniu na raz, co może zwiększyć produktywność.
6. **Motywacja:**
   * Dostarcza uczucia satysfakcji po zaznaczeniu wykonanego zadania.
   * Motywuje do kontynuowania pracy nad pozostałymi zadaniami.
7. **Łatwe Monitorowanie Postępu:**
   * Umożliwia łatwe monitorowanie postępu w wykonywaniu zadań.
   * Pomaga zidentyfikować obszary, w których osiągamy sukcesy, oraz te, które wymagają dodatkowej uwagi.
8. **Współpraca i Komunikacja:**
   * Wspiera współpracę w zespołach poprzez jasne określenie obowiązków.
   * Ułatwia komunikację między członkami zespołu odnośnie przypisanych zadań.
9. **Rutyna i Nawyk:**
   * Pomaga w budowaniu rutyny i nawyku codziennego sprawdzania zadań.
   * Staje się częścią systemu zarządzania czasem.
10. **Śledzenie Celów:**
    * Umożliwia śledzenie postępu w realizacji długoterminowych celów.
    * Pozwala na podjęcie konkretnych kroków w kierunku osiągnięcia celów.

W sumie, lista "to-do" jest narzędziem, które pomaga w skutecznym zarządzaniu czasem, zadaniami i priorytetami, co przyczynia się do osiągania celów osobistnych i zawodowych.

## Urls.py

calendarium/urls.py

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

Plik urls.py w Django służy do określania wzorców URL, które przekierowują żądania do odpowiednich funkcji widoku.

Na przykład, linia kodu **path('', views.index, name='home')** definiuje prostą funkcję widoku (opartą na funkcji), która renderuje stronę główną oraz dane listy zadań do zrobienia.

Dodatkowo, wprowadza osobną listę htmxpatterns do określenia dodatkowych wzorców URL specyficznie dla funkcji obsługujących htmx.

Na przykład:

* **'create\_todo/'**: Przypisuje do funkcji widoku create\_todo w module views i ma nazwę 'create\_todo'.
* **'delete\_todo/<int:pk>/'**: Przypisuje do funkcji widoku delete\_todo, która oczekuje na parametr całkowitoliczbowy pk w adresie URL. Ten wzorzec pozwala na usunięcie określonego elementu z listy zadań.
* **'mark\_todo/<int:pk>/'**: Przypisuje do funkcji widoku mark\_todo, która oczekuje na parametr całkowitoliczbowy pk w adresie URL. Ten wzorzec pozwala na oznaczenie określonego elementu z listy zadań jako ukończonego.

**urlpatterns += htmxpatterns** dodaje listę htmxpatterns do istniejącej listy urlpatterns, co efektywnie je łączy.

from django.urls import path

from . import views

urlpatterns = [

    path('', views.index, name='home'),

]

htmxpatterns = [

    path('create\_todo/', views.create\_todo, name='create\_todo'),

    path('delete\_todo/<int:pk>/', views.delete\_todo, name='delete\_todo'),

    path('mark\_todo/<int:pk>/', views.mark\_todo, name='mark\_todo'),

]

urlpatterns += htmxpatterns

## Models.py

Instrukcja **from django.db import models** służy do importowania modułu models z warstwy abstrakcji bazy danych w ramach frameworka Django.

W tym przypadku klasa Todo jest zdefiniowana jako podklasa models.Model. Oznacza to, że Todo to model Django, reprezentujący tabelę w bazie danych.

Wewnątrz klasy Todo zdefiniowane są następujące pola:

* title: To jest CharField o maksymalnej długości 100 znaków. Reprezentuje tytuł lub nazwę elementu todo.
* completed: To jest BooleanField z domyślną wartością False. Reprezentuje status ukończenia elementu todo.
* created\_at: To jest DateTimeField z auto\_now\_add=True. Reprezentuje znacznik czasowy, kiedy element todo został utworzony. Atrybut auto\_now\_add sprawia, że pole automatycznie ustawiane jest na bieżącą datę i godzinę podczas tworzenia nowego elementu todo.

Dodatkowo, wewnątrz klasy Todo zdefiniowana jest metoda **str**. Ta metoda zwraca ciąg znaków reprezentujący obiekt Todo, co jest przydatne do wyświetlania znaczących informacji podczas pracy z instancjami modelu Todo.

from django.db import models

# 👇 1. Add the following line's

class Todo(models.Model):

    title = models.CharField(max\_length=100)

    completed = models.BooleanField(default=False)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return self.title

## Views.py

Prosta funkcja widoku oparta na funkcji, która renderuje szablon strony głównej oraz dane listy zadań do zrobienia.

from django.shortcuts import render

from .models import Todo

# Rendering home page

def index(request):

    todos = Todo.objects.all().order\_by('-id')

    return render(request, 'index.html', {'todos': todos})

1. **Tworzenie nowego zadania (create\_todo):** Ta funkcja obsługuje żądanie POST i pobiera tytuł z danych POST żądania. Następnie tworzy nowy obiekt Todo z podanym tytułem, używając metody Todo.objects.create(). Metoda todo.save() zapisuje nowy obiekt Todo do bazy danych. W końcu pobiera wszystkie obiekty Todo z bazy danych, uporządkowane według ich identyfikatorów malejąco, i renderuje szablon 'todo-list.html' z zaktualizowaną listą zadań do zrobienia.
2. **Oznaczanie zadania jako ukończone (mark\_todo):** Ta funkcja przyjmuje identyfikator główny (pk) jako parametr i pobiera odpowiadający mu obiekt Todo za pomocą Todo.objects.get(). Następnie ustawia pole completed obiektu Todo na True i zapisuje zmiany w bazie danych. Następnie pobiera wszystkie obiekty Todo z bazy danych, uporządkowane według ich identyfikatorów malejąco, i renderuje szablon 'todo-list.html' z zaktualizowaną listą zadań do zrobienia.
3. **Usuwanie zadania (delete\_todo):** Ta funkcja również przyjmuje identyfikator główny (pk) jako parametr i pobiera odpowiadający mu obiekt Todo za pomocą Todo.objects.get(). Następnie usuwa obiekt Todo z bazy danych za pomocą metody delete(). Podobnie jak w poprzednich przypadkach, pobiera wszystkie obiekty Todo z bazy danych, uporządkowane według ich identyfikatorów malejąco, i renderuje szablon 'todo-list.html' z zaktualizowaną listą zadań do zrobienia.

W każdej funkcji, po modyfikacji obiektów Todo lub usunięciu jednego, renderowany jest szablon 'todo-list.html', aby wyświetlić zaktualizowaną listę zadań do zrobienia.

[Uwaga: Upewnij się, że powiązane wzorce URL są poprawnie zdefiniowane i przypisane do tych funkcji w konfiguracji URL aplikacji Django.]

from django.shortcuts import render

from .models import Todo

# Rendering home page

def index(request):

    todos = Todo.objects.all().order\_by('-id')

    return render(request, 'index.html', {'todos': todos})

# Creating a new Todo

def create\_todo(request):

    title = request.POST.get('title')

    todo = Todo.objects.create(title=title)

    todo.save()

    todos = Todo.objects.all().order\_by('-id')

    return render(request, 'todo-list.html', {'todos': todos})

# Marking completed True

def mark\_todo(request, pk):

    todo = Todo.objects.get(pk=pk)

    todo.completed = True

    todo.save()

    todos = Todo.objects.all().order\_by('-id')

    return render(request, 'todo-list.html', {'todos': todos})

# Deleting a Todo

def delete\_todo(request, pk):

    todo = Todo.objects.get(pk=pk)

    todo.delete()

    todos = Todo.objects.all().order\_by('-id')

    return render(request, 'todo-list.html', {'todos': todos})

## Templates

Obraz zawierający zrzut ekranu, Czcionka, tekst, numer

Opis wygenerowany automatycznie

### tbase.html

{% load static %}

<!doctype html>

<html lang="pl">

  <head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>{% block title %}{% endblock %}</title>

    <!-- Bootstrap 5 css -->

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUUVM" crossorigin="anonymous">

    <!-- Custom styles -->

    <link rel="stylesheet" href="{% static 'css/styles.css' %}">

  </head>

  <body>

        <div class="container mt-5">

            {% block content %}

            {% endblock %}

        </div>

      <!-- Bootstrap 5 js -->

      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js" integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy3Te4Bkz" crossorigin="anonymous"></script>

      <!-- HTMX -->

      <script src="https://unpkg.com/htmx.org@1.9.2" integrity="sha384-L6OqL9pRWyyFU3+/bjdSri+iIphTN/bvYyM37tICVyOJkWZLpP2vGn6VUEXgzg6h" crossorigin="anonymous"></script>

      {% block js\_script %}

      {% endblock %}

  </body>

</html>

### tindex.html

To jest element formularza, który używa atrybutów htmx. Atrybut hx-post określa URL dla przesłania formularza, a atrybut hx-target wskazuje element na stronie, który powinien zostać zaktualizowany w odpowiedzi. Tag {% csrf\_token %} dodaje token CSRF do formularza w celach zabezpieczenia.

Element div todoList: Ten element <div> o identyfikatorze "todoList" jest elementem docelowym, który zostanie zaktualizowany w odpowiedzi na przesłanie formularza. Zawiera on również inny szablon o nazwie 'todo-list.html', używając tagu {% include %}. Szablon 'todo-list.html' prawdopodobnie zawiera HTML reprezentujący listę zadań do zrobienia.

Blok js\_script: Ten blok definiuje skrypt JavaScript dodający token CSRF do nagłówków żądań htmx. Zapewnia to, że ochrona CSRF jest stosowana podczas wykonywania żądań htmx.

Podsumowując, ten szablon Django rozszerza szablon podstawowy i definiuje blok zawartości, który obejmuje formularz do tworzenia nowych elementów Todo. Wykorzystuje htmx do obsługi przesyłania formularza asynchronicznie i aktualizacji sekcji „todoList” na stronie w odpowiedzi.

{% extends 'base.html' %}

{% block title %} Home {% endblock %}

{% block content %}

<h3 class="my-5">Welcome to Your Todo List</h3>

<form hx-post="{% url 'create\_todo' %}" hx-target="#todoList" class="mx-auto">

    {% csrf\_token %}

    <div class="mb-3 row align-items-center">

        <label for="todoText" class="col-auto col-form-label">Enter your todo here: </label>

        <div class="col-6">

            <input type="text" name="title" class="form-control" id="todoText" required>

        </div>

        <div class="col-auto">

            <button class="btn btn-success">Add</button>

        </div>

    </div>

</form>

<div id="todoList">

    {% include 'todo-list.html' %}

</div>

{% endblock %}

{% block js\_script %}

<script>

    document.body.addEventListener('htmx:configRequest', (event) => {

        event.detail.headers['X-CSRFToken'] = '{{ csrf\_token }}';

    })

</script>

{% endblock %}

### Todo-list.html

To jest element formularza, który używa atrybutów htmx. Atrybut hx-post określa URL dla przesłania formularza, a atrybut hx-target wskazuje element na stronie, który powinien zostać zaktualizowany w odpowiedzi. Tag {% csrf\_token %} dodaje token CSRF do formularza w celach zabezpieczenia.

{% if todos %}

<div class="card" style="width: 18rem;">

    <ul class="list-group list-group-flush">

        {% for todo in todos %}

        <li class="list-group-item d-flex justify-content-between align-items-center">

            <div class="{% if todo.completed %} text-success text-decoration-line-through {% endif %}">

                {{ todo.title }}

            </div>

            <div>

                <span class="action badge rounded-pill text-bg-warning" hx-get="{% url 'mark\_todo' todo.pk %}"

                  hx-target="#todoList" style="cursor: pointer;">✔</span>

                <span class="action badge rounded-pill text-bg-danger" hx-delete="{% url 'delete\_todo' todo.pk %}"

                  hx-target="#todoList" hx-confirm="Are you sure you wish to delete?" style="cursor: pointer;">X</span>

            </div>

        </li>

        {% endfor %}

    </ul>

</div>

{% else %}

<h5>Currently, you don't have any todos.</h5>

{% endif %}

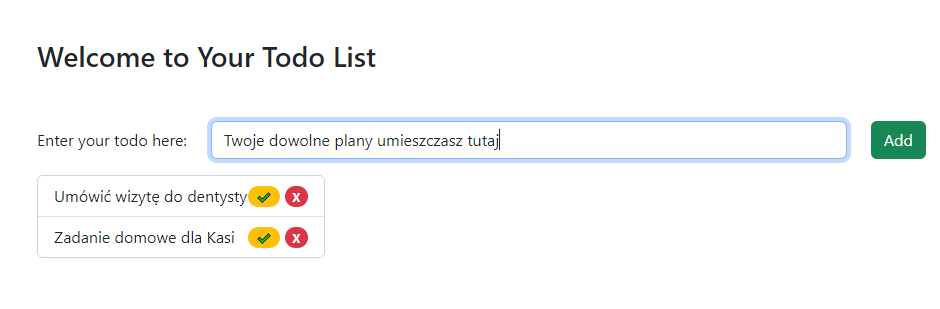
Element div todoList: Ten element <div> o identyfikatorze "todoList" jest elementem docelowym, który zostanie zaktualizowany w odpowiedzi na przesłanie formularza. Zawiera on również inny szablon o nazwie 'todo-list.html', używając tagu {% include %}. Szablon 'todo-list.html' prawdopodobnie zawiera HTML reprezentujący listę zadań do zrobienia.

Blok js\_script: Ten blok definiuje skrypt JavaScript dodający token CSRF do nagłówków żądań htmx. Zapewnia to, że ochrona CSRF jest stosowana podczas wykonywania żądań htmx.

Podsumowując, ten szablon Django rozszerza szablon podstawowy i definiuje blok zawartości, który obejmuje formularz do tworzenia nowych elementów Todo. Wykorzystuje htmx do obsługi przesyłania formularza asynchronicznie i aktualizacji sekcji „todoList” na stronie w odpowiedzi.

## Wygląd listy todo

Tak prezentuje się podstawowy wygląd tej strony:



Dodamy nawigację u góry (navbar) i zmienimy język na język użytkownika (polski).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

# Podsumowanie

Projekt aplikacji internetowej skupia się na kompleksowej platformie wspomagającej codzienne działania użytkowników. Zapewnia zarządzanie kalendarzem, listą zadań todo, kontrolowaną modyfikację, elastyczność dostępu, powiadomienia, personalizację interfejsu, oraz dba o bezpieczeństwo i prywatność.

Aplikacja efektywnie wspiera organizację czasu poprzez planowanie zadań i wydarzeń. Jej uniwersalne zastosowanie obejmuje centralizację informacji, podgląd postępów, zarządzanie zadaniami, wsparcie w planowaniu wydarzeń, łatwy dostęp z różnych urządzeń, wsparcie dla różnych stylów pracy, poprawę produktywności i wsparcie dla planowania długoterminowego.

Aplikacja została zaprojektowana, aby zaspokoić konkretne potrzeby biznesowe i użytkowników końcowych. Kluczowe elementy to zarządzanie działaniami, centralizacja informacji, podgląd postępów, zarządzanie zadaniami, wsparcie w planowaniu wydarzeń, łatwy dostęp z dowolnego miejsca, wsparcie dla różnych stylów pracy.

Aplikacja opiera się na spójnej logice, obejmującej zarządzanie zadaniami i wydarzeniami, centralne przechowywanie danych, podgląd postępów, planowanie i kalendarz, elastyczność dostępu, indywidualne dostosowanie, powiadomienia i przypomnienia, zabezpieczenia i bezpieczeństwo, intuicyjną nawigację oraz potencjalnie wsparcie dla zespołów.

Stworzenie aplikacji było możliwe dzięki zastosowaniu różnorodnych technologii, takich jak VS Code, HTML, Python, Bootstrap, Django, Git/GitHub, Crispy, oraz htmx. Każda z tych technologii wniosła specyficzne korzyści, a staranne zarządzanie ograniczeniami pozwala na efektywne wdrożenie aplikacji.

Tworząc projekt aplikacji internetowej do zarządzania kalendarzem i listą zadań todo, zdobyłem różnorodne umiejętności i doświadczenia. Przede wszystkim, praca nad tym projektem pozwoliła mi na rozwinięcie umiejętności programowania w Pythonie. Wykorzystanie tego języka było kluczowe do efektywnego tworzenia logiki backendu, a także do korzystania z frameworka Django.

Praca z frameworkiem Django była kolejnym aspektem projektu, który pozwolił mi na naukę, jak efektywnie tworzyć aplikacje internetowe. Zdobycie doświadczenia w obszarze tworzenia modeli, obsługi bazy danych za pomocą Django ORM oraz implementacji logiki biznesowej było kluczowe dla sukcesu projektu.

Projektowanie responsywnego interfejsu użytkownika przy użyciu Bootstrapa było również częścią mojej pracy. Wykorzystanie tego frameworka front-endowego pozwoliło mi na nabycie umiejętności projektowania atrakcyjnego i funkcjonalnego interfejsu, który działa sprawnie na różnych urządzeniach.

Korzystanie z systemu kontroli wersji Git i platformy GitHub było integralną częścią projektu. Umożliwiło mi to skuteczne śledzenie zmian w kodzie, współpracę zespołową oraz zarządzanie wersjami projektu.

Implementacja zabezpieczeń, takich jak uwierzytelnianie i autoryzacja, była ważnym aspektem projektu, który pozwolił mi na rozwinięcie umiejętności dbania o bezpieczeństwo danych użytkowników.

Wykorzystanie htmx do obsługi asynchronicznych żądań i dynamicznej aktualizacji treści strony pozwoliło mi na rozwinięcie umiejętności pracy z technologią umożliwiającą interaktywne elementy strony.

Praktyka korzystania z Visual Studio Code jako edytora tekstu była kluczowa dla efektywnej pracy programistycznej. Pozwoliła mi na wydajne kodowanie i debugowanie aplikacji.

Praca z różnymi technologiami, takimi jak Python, HTML, Bootstrap, Django, Git/GitHub, Crispy, i htmx, dostarczyła mi umiejętności organizacji kodu i zasobów projektu oraz rozwijania kreatywnych rozwiązań dla interfejsu użytkownika.

Analiza wymagań biznesowych i użytkowych, koordynacja zadań pomiędzy różnymi modułami aplikacji, a także utrzymanie spójności projektu to kolejne aspekty, które przyczyniły się do mojego wszechstronnego rozwoju.

Aplikacja stanowi kompleksowe narzędzie do organizacji codziennego życia, zarządzania zadaniami i planowania przyszłych wydarzeń. Jej uniwersalne zastosowanie, spójna koncepcja i logiczne podejście do technologii sprawiają, że spełnia ona oczekiwania zarówno biznesu, jak i użytkowników końcowych, poprawiając efektywność i satysfakcję z organizacji codziennych obowiązków.

# Literatura

https://medium.com/@devsumitg/django-auth-user-signup-and-login-7b424dae7fab

https://tutorial101.blogspot.com/2022/05/python-django-ajax-fullcalender-crud.html

<https://medium.com/@devsumitg/simple-todo-list-app-in-django-framework-htmx-bootstrap-5-8b68cbd47a1a>

<https://getbootstrap.com/docs/5.3/getting-started/introduction/>

<https://api.jquery.com/category/ajax/>

<https://api.jquery.com/>

https://docs.djangoproject.com/en/5.0/

<https://django-crispy-forms.readthedocs.io/en/latest/>

<https://htmx.org/docs/>

<https://git-scm.com/doc>

https://docs.github.com/en

<https://code.visualstudio.com/docs>

<https://developer.mozilla.org/en-US/docs/Web/HTML>

https://fullcalendar.io/