Jakis przykadowy HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Cool Colorful Page</title>
  <style>
    body {
      background-color: #f0f8ff; /* light blue background */
      font-family: Arial, sans-serif; /* nicer text */
    }

    h1 {
      color: #ff4500; /* orange title */
      text-align: center;
    }

    table {
      width: 50%;
      margin: 20px auto;
      border-collapse: collapse;
      box-shadow: 0 0 10px rgba(0,0,0,0.2);
    }

    td {
      border: 2px solid #4CAF50; /* green border */
      padding: 15px;
      text-align: center;
      background-color: #e0ffe0; /* light green cells */
    }

    .highlight {
      background-color: yellow; /* special cell */
      color: red;
      font-weight: bold;
    }

    button {
      color: purple;
      background-color: pink;
      font-family: Tahoma;
    }
    td:hover {
      background-color: #add8e6; /* light blue on hover */
    }
  </style>
</head>

<body>
  <ul>
    <li>Test <a href="."">Test</a> Test</li>
    <li>Test <a href="."">Test drugi</a> Test</li>
    <li>
      <p>A tu nie ma linkow</p>
      <table>
        <tr>
          <td>1</td>
          <td>2</td>
        </tr>
        <tr>
          <td>3</td>
          <td>4</td>
          <td class="highlight">5</td>
        </tr>
      </table>
    </li>
  </ul>
  <button onclick="alert('You clicked me!')">Click me!</button>

  <p>
    Nowy paragraf
  </p>
  <div style="width: 200px; height: 100px; background-color: lightcoral; border-radius: 10px; text-align: center;">
    I am a colorful block!
  </div>
</body>
</html>
```

urls

```
from django.urls import path
from . import views
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('', views.gameboard_list, name='gameboard_list'),
    path('gameboard/create/', views.gameboard_create, name='gameboard_create'),
    path('gameboard/<int:board_id>/dot/add/', views.dot_add, name='dot_add'),
    path('register/', views.register, name='register'),
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('gameboard/<int:gameboard_id>/delete/', views.gameboard_delete, name='gameboard_delete'),
    path('gameboard/<int:gameboard_id>/save/', views.gameboard_save, name='gameboard_save'),
    path('gameboard/<int:board_id>/save_path/', views.save_user_path, name='save_user_path'),
    path('gameboard/<int:board_id>/user_paths/', views.user_paths, name='user_paths'),
    path('gameboard/<int:board_id>/delete_path/', views.delete_path, name='delete_path'),
    path('sse/notifications/',views.sse_notifications,name='sse_notifications')
]
```

forms

```
from django import forms
from .models import GameBoard

class GameBoardForm(forms.ModelForm):
    rows = forms.IntegerField(min_value=2, max_value=20, label="Liczba wierszy")
    cols = forms.IntegerField(min_value=2, max_value=20, label="Liczba kolumn")
    class Meta:
        model = GameBoard
        fields = ['name', 'rows', 'cols']
```

Models: projekt

```
from django.db import models
from django.contrib.auth.models import User
from django.urls import reverse


class GameBoard(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name="Uzytkownik")
    name = models.CharField(max_length=100, verbose_name="Nazwa")
    rows = models.PositiveIntegerField(verbose_name="Liczba wierszy")
    cols = models.PositiveIntegerField(verbose_name="Liczba kolumn")

    class Meta:
        verbose_name = "Plansza gry"
        verbose_name_plural = "Plansze gry"

    def __str__(self):
        return f"{self.name} ({self.user.username})"

class Dot(models.Model):
        board = models.ForeignKey(GameBoard, on_delete=models.CASCADE, related_name='dots',
verbose_name="Plansza")
    row = models.PositiveIntegerField(verbose_name="Wiersz")
    col = models.PositiveIntegerField(verbose_name="Kolumna")
    color = models.CharField(max_length=7, verbose_name="Kolor (HEX)")

    class Meta:
        verbose_name = "Kropka"
        verbose_name_plural = "Kropki"

    def __str__(self):
        return f"Kropka ({self.row}, {self.col}, {self.color}) na {self.board.name}"

class UserPath(models.Model):
    board = models.ForeignKey(GameBoard, on_delete=models.CASCADE, related_name='paths')
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    color = models.CharField(max_length=7, verbose_name="Kolor (HEX)")
    path = models.JSONField()  # lista punktow [{row: x, col: y}, ...]

    class Meta:
        verbose_name = "Sciezka uzytkownika"
        verbose_name_plural = "Sciezki uzytkownika"
        unique_together = ('board', 'user', 'color')  # jeden uzytkownik, jeden kolor, jedna plansza = jedna sciezka

    def __str__(self):
        return f"Sciezka {self.user.username} {self.color} na {self.board.name}"
```

serializers

```
# plik: editor/serializers.py
from rest_framework import serializers
from .models import Route, RoutePoint

class RoutePointSerializer(serializers.ModelSerializer):
    class Meta:
        model = RoutePoint
        fields = ['id', 'x', 'y', 'order']

class RouteSerializer(serializers.ModelSerializer):
    points = RoutePointSerializer(many=True, read_only=True)

    class Meta:
        model = Route
        fields = ['id', 'name', 'background', 'points']
```

logged_out / login / register

```
{% extends "editor/base.html" %}
{% block content %}
<p>Wylogowano.</p>
<a href="{% url 'login' %}">Zaloguj sie ponownie</a>
{% endblock %}
####
{% extends "editor/base.html" %}

{% block content %}
    {% if not user.is_authenticated %}
        <h2>Logowanie</h2>
        <form method="post" action="{% url 'login' %}">
            {% csrf_token %}
            {{ form.as_p }}
            <button type="submit">Zaloguj</button>
            <input type="hidden" name="next" value="{{ next }}">
        </form>
    {% endif %}
{% endblock %}
####
{% extends "editor/base.html" %}

{% block content %}
<h2>Rejestracja</h2>
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Zarejestruj</button>
</form>
<p>Masz juz konto? <a href="{% url 'login' %}">Zaloguj sie</a></p>
{% endblock %}
```

views

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from django.contrib.auth import login as auth_login
from django.contrib.auth.forms import UserCreationForm
from django.contrib import messages
from .models import GameBoard, Dot, UserPath
from django.core.paginator import Paginator
from django.contrib import messages
from .forms import GameBoardForm

import json

from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from django.views.decorators.http import require_POST

import time
from django.http import StreamingHttpResponse

import queue
import threading

client_queues = []
client_queues_lock = threading.Lock()

def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            auth_login(request, user)
            return redirect('route_list')
    else:
        form = UserCreationForm()
    return render(request, 'registration/register.html', {'form': form})

@login_required
def gameboard_list(request):
    boards = GameBoard.objects.all().order_by('-id')
    paginator = Paginator(boards, 10)
    page = request.GET.get('page')
    gameboards = paginator.get_page(page)
    return render(request, 'editor/gameboard_list.html', {'gameboards': gameboards})
```

views 2

```
@login_required
def gameboard_create(request):
    if request.method == 'POST':
        form = GameBoardForm(request.POST)
        if form.is_valid():
            board = form.save(commit=False)
            board.user = request.user
            board.save()
            return redirect('gameboard_list')
    else:
        form = GameBoardForm()
    return render(request, 'editor/gameboard_form.html', {'form': form})

@login_required
def dot_add(request, board_id):
    board = get_object_or_404(GameBoard, id=board_id, user=request.user)
    if request.method == 'POST':
        row = request.POST.get('row')
        col = request.POST.get('col')
        color = request.POST.get('color')
        if row is not None and col is not None and color:
            Dot.objects.create(
                board=board,
                row=int(row),
                col=int(col),
                color=color
            )
            messages.success(request, "Kropka zostaa dodana.")
        else:
            messages.error(request, "Wszystkie pola sa wymagane.")
        return redirect('gameboard_edit', board_id=board.id)
    return render(request, 'editor/dot_form.html', {'board': board})


@login_required
def gameboard_detail(request, gameboard_id):
    # Tu usunaem user=request.user
    board = get_object_or_404(GameBoard, id=gameboard_id)
    dots = list(board.dots.values('row', 'col', 'color'))  # jesli relacja: board.dots.all()
    context = {
        'board': board,
        'dots_json': json.dumps(dots),  # import json na gorze pliku!
    }
    return render(request, 'editor/gameboard_detail.html', context)

@login_required
def gameboard_delete(request, gameboard_id):
    board = get_object_or_404(GameBoard, id=gameboard_id, user=request.user)
    board.delete()
    messages.success(request, "Plansza zostaa usunieta.")
    return redirect('gameboard_list')

@login_required
def gameboard_save(request, gameboard_id):
    if request.method == 'POST':
        board = get_object_or_404(GameBoard, id=gameboard_id, user=request.user)
        try:
            data = json.loads(request.body)
            dots = data.get('dots', [])
            rows = data.get('rows')
            cols = data.get('cols')
            if rows is not None and cols is not None:
                board.rows = rows
                board.cols = cols
                board.save()
                # Optionally: remove dots out of bounds in backend
                board.dots.filter(row__gte=rows).delete()
                board.dots.filter(col__gte=cols).delete()
            # Remove old dots
            board.dots.all().delete()
            # Add new dots
            for dot in dots:
                Dot.objects.create(
                    board=board,
                    row=dot['row'],
                    col=dot['col'],
                    color=dot['color']
                )
            return JsonResponse({'status': 'ok'})
        except Exception as e:
            return JsonResponse({'status': 'error', 'message': str(e)}, status=400)
    return JsonResponse({'status': 'error', 'message': 'Invalid method'}, status=405)

@login_required
@require_POST
def save_user_path(request, board_id):
    board = get_object_or_404(GameBoard, id=board_id)
    data = json.loads(request.body)
    color = data.get('color')
    path = data.get('path', [])
    # Validate path: must connect exactly two dots of the same color
    if not color or not path or len(path) < 2:
        return JsonResponse({'status': 'error', 'message': 'Nieprawidowa sciezka.'}, status=400)

    # Find all dots of this color
    dots = list(board.dots.filter(color=color).values('row', 'col'))
    endpoints = [p for p in [path[0], path[-1]] if p in dots]
    print("SAVE USER: " + str(color) + " " + str(len(endpoints)))
    if len(endpoints) != 2 or endpoints[0] == endpoints[1]:
        return JsonResponse({'status': 'error', 'message': 'Sciezka musi aczyc dwie rozne kropki tego samego koloru.'}, status=400)

    # Save or update UserPath
    UserPath.objects.update_or_create(
        board=board,
        user=request.user,
        color=color,
        defaults={'path': path}
    )
    return JsonResponse({'status': 'ok'})
def user_paths(request, board_id):
    # Filter by user and board
    paths = UserPath.objects.filter(user=request.user, board_id=board_id)
    data = {
        "paths": [
            {
                "color": up.color,
                "path": up.path,  # path should be a list of {row, col}
            }
            for up in paths
        ]
    }
    return JsonResponse(data)
```

views 3

```
@login_required
@require_POST
def delete_path(request, board_id):
    board = get_object_or_404(GameBoard, id=board_id)
    data = json.loads(request.body)
    color = data.get('color')
    if not color:
        return JsonResponse({'status': 'error', 'message': 'Brak koloru.'}, status=400)
    UserPath.objects.filter(board=board, user=request.user, color=color).delete()
    return JsonResponse({'status': 'ok'})


def sse_notifications(request):
    q = queue.Queue()
    with client_queues_lock:
        client_queues.append(q)

    def event_stream():
        try:
            while True:
                try:
                    event = q.get(timeout=5)
                    yield f"event: {event['event']}\n"
                    yield f"data: {json.dumps(event['data'])}\n\n"
                except queue.Empty:
                    yield ":\n\n"  # keep-alive
        finally:
            with client_queues_lock:
                client_queues.remove(q)

    response = StreamingHttpResponse(event_stream(), content_type='text/event-stream')
    response['Cache-Control'] = 'no-cache'
    return response

def broadcast_event(event):
    with client_queues_lock:
        for q in list(client_queues):
            q.put(event)


from django.db.models.signals import post_save
from django.dispatch import receiver

@receiver(post_save, sender=GameBoard)
def board_created(sender, instance, created, **kwargs):
    if created:
        broadcast_event({
            "event": "newBoard",
            "data": {
                "board_id": instance.id,
                "board_name": instance.name,
                "creator_username": instance.user.username,
            }
        })

@receiver(post_save, sender=UserPath)
def path_created(sender, instance, created, **kwargs):
    if created:
        broadcast_event({
            "event": "newPath",
            "data": {
                "path_id": instance.id,
                "board_id": instance.board.id,
                "board_name": instance.board.name,
                "user_username": instance.user.username,
            }
        })
```

gameboard_detail 1:

```
interface Dot {
    row: number;
    col: number;
    color: string;
}
interface PathPoint { row: number; col: number; }
interface UserPath {
    color : string;
    path : PathPoint[];
}
console.log("Script loaded"); // Add this at the top

// These should be set in your HTML template
declare const boardId: number;
declare const csrfToken: string;
declare const initialDots: Dot[];
declare const isOwner: boolean;
let activeColor: string | null = null;

let dots: Dot[] = initialDots ? [...initialDots] : [];
const gridElem = document.getElementById('board-grid') as HTMLElement;

let currentPath: PathPoint[] = [];
let pathColor: string | null = null;

let userPaths: UserPath[] = [];


const resizeRowsInput = document.getElementById('resize-rows') as HTMLInputElement;
const resizeColsInput = document.getElementById('resize-cols') as HTMLInputElement;
const resizeBtn = document.getElementById('resize-board-btn') as HTMLButtonElement;



// Helper: render the grid and all dots
function renderGrid() {
    const rows = Number(gridElem.dataset.rows);
    const cols = Number(gridElem.dataset.cols);
    gridElem.innerHTML = '';
    gridElem.style.display = 'grid';
    gridElem.style.gridTemplateRows = `repeat(${rows}, 1fr)`;
    gridElem.style.gridTemplateColumns = `repeat(${cols}, 1fr)`;

    const borderSize = 2; // px, per cell (1px each side)
    const gapSize = 2;    // px, per gap

    // Remove old paths
    userPaths = userPaths.filter(up => {
        if (!up.path.length) return false;
        const endpoints = [up.path[0], up.path[up.path.length - 1]];
        const endpointsExist = endpoints.every(ep =>
            dots.some(d => d.row === ep.row && d.col === ep.col && d.color === up.color)
        );
        if (!endpointsExist) {
            // Remove from backend
            deleteUserPath(up);
            return false;
        }
        return true;
    });
    // Calculate total space taken by borders and gaps
    const totalGapWidth = gapSize * (cols - 1);
    const totalGapHeight = gapSize * (rows - 1);
    const totalBorderWidth = borderSize * cols;
    const totalBorderHeight = borderSize * rows;

    const maxGridWidth = Math.min(window.innerWidth * 0.7, 800);
    const maxGridHeight = Math.min(window.innerHeight * 0.6, 600);

    // Subtract borders and gaps from available space
    const availableWidth = maxGridWidth - totalGapWidth - totalBorderWidth;
    const availableHeight = maxGridHeight - totalGapHeight - totalBorderHeight;

    // Calculate the largest possible square cell size that fits the grid
    const cellSize = Math.floor(Math.min(
        availableWidth / cols,
        availableHeight / rows
    ));

    // Set the grid's width and height so all cells are square and fit
    gridElem.style.width = `${cellSize * cols + totalGapWidth + totalBorderWidth}px`;
    gridElem.style.height = `${cellSize * rows + totalGapHeight + totalBorderHeight}px`;

    gridElem.style.background = '#fff';
    gridElem.style.border = 'none';
    gridElem.style.gap = `${gapSize}px`;

    for (let r = 0; r < rows; r++) {
        for (let c = 0; c < cols; c++) {
            const cell = document.createElement('div');
            cell.className = 'grid-cell';
            cell.style.background = '#f8f8f8';
            cell.style.border = '1px solid #444';
            cell.style.display = 'flex';
            cell.style.alignItems = 'center';
            cell.style.justifyContent = 'center';
            cell.style.width = `${cellSize}px`;
            cell.style.height = `${cellSize}px`;
            cell.dataset.row = r.toString();
            cell.dataset.col = c.toString();

            // Render dot if present
            const dot = dots.find(d => d.row === r && d.col === c);
            if (dot) {
                const dotElem = document.createElement('div');
                dotElem.style.width = '60%';
                dotElem.style.height = '60%';
                dotElem.style.borderRadius = '50%';
                dotElem.style.background = dot.color;
                cell.appendChild(dotElem);
```

gameboard_dettail 2

```
                // Remove dot on left-click
                if (isOwner) {
                    cell.addEventListener('click', () => {
                        dots = dots.filter(d => !(d.row === r && d.col === c));
                        renderGrid();
                    });
                }
                else {
                    // User: remove own path if clicking endpoint
                    cell.addEventListener('click', () => {
                        // Find user's path for this color
                        const myPath = userPaths.find(up => up.color === dot.color);
                        if (myPath && myPath.path.length > 1) {
                            const first = myPath.path[0];
                            const last = myPath.path[myPath.path.length - 1];
                            if (
                                (first.row === r && first.col === c) ||
                                (last.row === r && last.col === c)
                            ) {
                                deleteUserPath(myPath);
                                // Optionally update UI immediately:
                                userPaths = userPaths.filter(up => up !== myPath);
                                renderGrid();
                            }
                        }
                    });
                }
                // Start  / end path logic
                cell.addEventListener('contextmenu', (e) => {
                    e.preventDefault();
                    if (activeColor && dot.color === activeColor) {
                        // If not drawing a path, start a new one
                        if (!pathColor || currentPath.length === 0) {
                            currentPath = [{ row: r, col: c }];
                            pathColor = activeColor;
                            renderGrid();
                        } else if (
                            pathColor === dot.color &&
                            currentPath.length > 0 &&
                            isAdjacent(currentPath[currentPath.length - 1], { row: r, col: c }) &&
                            !currentPath.some(p => p.row === r && p.col === c)
                        ) {
                            // If finishing a path on the second dot of the same color
                            currentPath.push({ row: r, col: c });
                            saveUserPath();
                            currentPath = [];
                            pathColor = null;
                            renderGrid();
                        }
                    }
                });
            } else {
                if (isOwner) {
                    cell.addEventListener('click', () => {
                        if (
                            activeColor &&
                            dots.filter(d => d.color === activeColor).length < 2 &&
                            !dots.find(d => d.row === r && d.col === c)
                        ) {
                            dots.push({ row: r, col: c, color: activeColor });
                            renderGrid();
                        }
                    });
                }

                // Right-click to extend path
                cell.addEventListener('contextmenu', (e) => {
                    e.preventDefault();
                    if (
                        pathColor &&
                        activeColor === pathColor &&
                        currentPath.length > 0
                    ) {
                        const last = currentPath[currentPath.length - 1];
                        if (
                            isAdjacent(last, { row: r, col: c }) &&
                            !currentPath.some(p => p.row === r && p.col === c)
                        ) {
                            currentPath.push({ row: r, col: c });
                            renderGrid();
                        }
                    }
                });
```

gameboard_detail 3:

```
            userPaths.forEach(up => {
                up.path.forEach((p: PathPoint) => {
                    if (p.row === r && p.col === c) {
                        cell.style.background = up.color.length === 7 ? up.color + "30" : up.color;
                    }
                });
            });
            // Visual feedback for path (semi-transparent)
            if (currentPath.some(p => p.row === r && p.col === c) && pathColor) {
                cell.style.background = pathColor.length === 7 ? pathColor + "80" : pathColor;
            }
        }

        gridElem.appendChild(cell);
    }
  }
}

// Color palette logic (assumes .color-btn elements exist)
document.querySelectorAll('.color-btn').forEach(btn => {
    btn.addEventListener('click', () => {
        activeColor = (btn as HTMLElement).dataset.color!;
        console.log("Active color set to", activeColor)
        document.querySelectorAll('.color-btn').forEach(b => b.classList.remove('active'));
        btn.classList.add('active');
    });
});

function getBoardState() {
    return {
        rows: Number(gridElem.dataset.rows),
        cols: Number(gridElem.dataset.cols),
        dots: dots
    };
}
function saveBoard() {
    const boardState = getBoardState();
    fetch(`/gameboard/${boardId}/save/`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': csrfToken,
        },
        body: JSON.stringify(boardState),
    })
    .then(res => res.json())
    .then(data => {
        if (data.status === 'ok') {
            alert('Plansza zapisana!');
        } else {
            alert('Bad zapisu: ' + data.message);
        }
    });
}

function isAdjacent(a: PathPoint, b: PathPoint) {
    return (Math.abs(a.row - b.row) + Math.abs(a.col - b.col)) === 1;
}

// Save path to backend
function saveUserPath() {
    console.log("Saving path:", currentPath, "color:", pathColor);
    fetch(`/gameboard/${boardId}/save_path/`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': csrfToken,
        },
        body: JSON.stringify({ color: pathColor, path: currentPath }),
    })
    .then(res => res.json())
    .then(data => {
        if (data.status === 'ok') {
            alert('Sciezka zapisana!');
            // Clear current path and color
            currentPath = [];
            pathColor = null;
            // Fetch and render updated user paths
            fetchUserPaths();
        } else {
            alert('Bad: ' + data.message);
        }
    });
}
function fetchUserPaths() {
    fetch(`/gameboard/${boardId}/user_paths/`)
        .then(res => res.json())
        .then(data => {
            userPaths = data.paths || [];
            renderGrid();
        });
}
```

gameboard_detail 4

```
function deleteUserPath(path: UserPath) {
    fetch(`/gameboard/${boardId}/delete_path/`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': csrfToken,
        },
        body: JSON.stringify({ color: path.color }),
    })
    .then(res => res.json())
    .then(data => {
        if (data.status !== 'ok') {
            console.error('Failed to delete path:', data.message);
        }
    });
}
if (resizeRowsInput && resizeColsInput && resizeBtn && isOwner) {
    resizeBtn.addEventListener('click', () => {
        const newRows = Math.max(2, Math.min(20, Number(resizeRowsInput.value)));
        const newCols = Math.max(2, Math.min(20, Number(resizeColsInput.value)));
        // Always use current values from DOM, not global rows/cols
        if (
            newRows !== Number(gridElem.dataset.rows) ||
            newCols !== Number(gridElem.dataset.cols)
        ) {
            gridElem.dataset.rows = newRows.toString();
            gridElem.dataset.cols = newCols.toString();
            renderGrid();
            saveBoard();
        }
    });
}
// Initial render
renderGrid();
fetchUserPaths();
```

NOWE views

```
from django.shortcuts import render
import requests
from bs4 import BeautifulSoup
from django.http import HttpResponse
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib.auth import login, logout
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required

def register_view(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('/')
    else:
        form = UserCreationForm()
    return render(request, 'egz/register.html', {'form': form})

def logout_view(request):
    logout(request)
    return redirect('login_view')

def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data = request.POST)
        if form.is_valid():
            user = form.get_user()
        login(request, user)
        return redirect('/')
    else:
        form = AuthenticationForm()
    return render(request, 'egz/login.html', {'form': form})

@login_required
def index(request):
    return render(request, "egz/index.html")


@login_required
def scraper(request):
    url = "http://127.0.0.1:8000/"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    links = soup.find_all('a')
    return HttpResponse(f"Liczba linkow na stronie: {len(links)}")

import time
import random
from django.http import StreamingHttpResponse

MESSAGES = ["Wiadomosc A", "Info B", "Alert C"]

def sse_random_info(request):
    def event_stream():
        while True:
            msg = random.choice(MESSAGES)
            yield f"data: {msg}\n\n"
            time.sleep(random.uniform(3, 5))
    response = StreamingHttpResponse(event_stream(), content_type='text/event-stream')
    response['Cache-Control'] = 'no-cache'
    return response
```

Reczne

```
LOGIN_URL="/login" w settings
LOGIN_REDIRECT_URL = '/'
python3 manage.py runserver
python manage.py createsuperuser


// main.scss
$primary-color: #3498db;
$secondary-color: #2ecc71;
$font-family: 'Arial, sans-serif';

body {
  font-family: $font-family;
  background-color: $primary-color;
  header {
    color: $secondary-color;
  }
  footer {
    color: $secondary-color;
  }
}

// package.json
"scripts": {
  "scss": "sass static/scss/main.scss static/css/style.css"
}

<!-- index.html -->
<div id="sse-box" style="position:fixed; top:0; right:0;"></div>
<script>
  const evtSource = new EventSource('/sse-random-info/');
  evtSource.onmessage = e => {
    document.getElementById('sse-box').textContent = e.data;
  };
</script>
SCSS/TS build: npm run scss, npm run ts

SSE must use StreamingHttpResponse.

For BeautifulSoup: pip install beautifulsoup4 requests

Authentication uses Djangos built-in forms.

LOGIN_URL redirects unauthorized users.

TypeScript compiles to JS via tsc.

Struktura

"scripts": {
"test": "echo \"Error: no test specified\" && exit 1",
"build-css": "sass egz/static/egz/main.scss egz/static/egz/style.css",
"build-ts": "tsc egz/typescript/script.ts --outFile egz/static/egz/script.js",
"build-ts2": "tsc egz/typescript/notification.ts --outFile egz/static/egz/notification.js"
},
```

REczne 2

```
{% extends "base.html" %}

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

{% block content %}
    <h2>This is the homepage!</h2>
    <p>Welcome!</p>
{% endblock %}


///
document.addEventListener('DOMContentLoaded', function () {
    var links = document.querySelectorAll('a');
    links.forEach(function (link) {
        link.addEventListener('mouseenter', function () {
            console.log('Dugosc tekstu w linku:', link.textContent.length);
        });
    });
});
```