USOS
Architektura projektu

By Krakowskie Szakale

Co to jest architektura?

- Fundament projektu, na którym budowane są kolejne moduły i funkcje. Dzięki niej można zrozumieć, jak poszczególne elementy współdziałają w celu realizacji założonych wymagań biznesowych.

- Definiuje najważniejsze właściwości systemu w jego środowisku

- Jest to również rezultat zastosowania procesów projektowania architektonicznego

Jaka wybrać odpowiednią architekturę

- Jakie są kluczowe cele biznesowe (np. szybkość wdrożenia, skalowalność, niezawodność)?
- Jakie są najważniejsze funkcjonalności które musimy dostarczyć?
- Jaka architektura rozwiązuje najwięcej problemów?
- Jaka jest skala projektu?
- Jakie posiadamy zasoby?
- Jakich narzędzi i technologii używamy?
- Jak modularny powinien być projekt?
- Jakie doświadczenie posiada zespół

Rozpatrzone opcje

- Mikroserwisy
- P2P
- trójwarstwowa (UI/API/DB)
- Klient-serwer

Rozpatrzone opcje: Mikroserwisy

System podzielony na niezależne, małe serwisy, które komunikują się ze sobą przez API.

Zalety:

- Możliwość skalowania tylko tych mikroserwisów, które tego wymagają
- Każdy serwis może być rozwijany i utrzymywany oddzielnie
- Każdy serwis może być napisany w innej technologii
- Niezależność działania serwisów

Wady:

- Konieczność osobnej konfiguracji każdego serwisu
- Duża złożoność systemu
- Wysokie koszty utrzymania
- Problemy ze spójnością danych

Rozpatrzone opcje: P2P

Model, w którym każdy węzeł w sieci może pełnić zarówno rolę klienta, jak i serwera.

Zalety:

- Eliminacja pojedynczego punktu awarii
- Łatwe dodawanie nowych węzłów bez wpływu na działanie systemu
- Brak potrzeby utrzymywania centralnego serwera

Wady:

- Problemy z synchronizacją
- Problemy z wersjonowaniem
- Trudne zarządzanie węzłami
- Brak jednego spójnego źródła informacji o systemie (brak centralizacji)

Rozpatrzone opcje: trójwarstwowa (UI/API/DB)

System podzielony na trzy warstwy: prezentacji (frontend), logiki biznesowej (backend) i danych (baza danych).

Zalety:

- Modularność
- Podział odpowiedzialności
- Skalowalność pozioma

Wady:

- Konieczność konfiguracji i obsługi zewnętrznej bazy danych
- Bardziej skomplikowana infrastruktura
- Dodatkowe opóźnienie od komunikacji z bazą danych

Rozpatrzone opcje: Klient - serwer

Model, w którym klient wysyła żądania do serwera, a serwer przetwarza je i zwraca odpowiedzi.

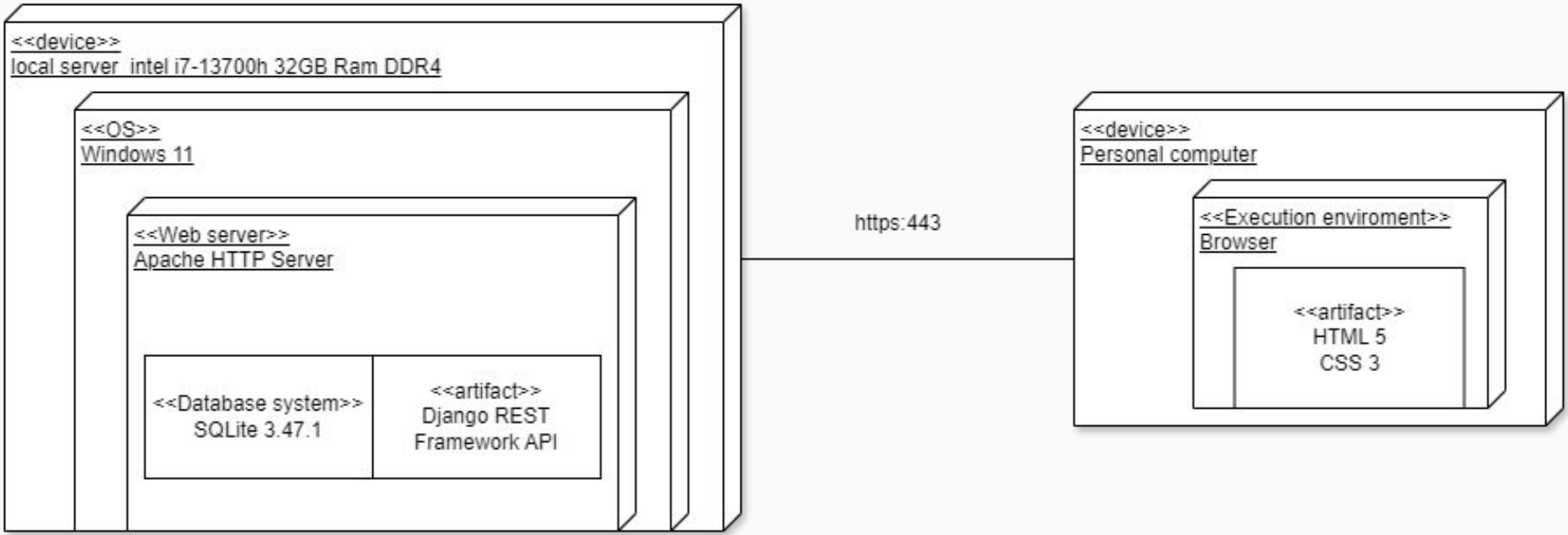
Zalety:

- Łatwy do zrozumienia i wdrożenia
- Dane i logika są przechowywane na jednym serwerze (centralizacja)
- Wymagana minimalna konfiguracja
- Spójność wersji
- Łatwe skalowanie pionowe
- Tani w utrzymaniu

Wady:

- Serwer może stać się wąskim gardłem przy dużym ruchu
- Awaria serwera unieruchamia cały system

Deployment diagram



Przykłady łączenia się klienta z serwerem

```
async getUsersGradesFromSubject(userId: number, subjectId: number): Promise<string[]> {
  const url = `http://localhost:8000/grades/${userId}/${subjectId}`;
  try {
    const response = await lastValueFrom(this.http.get<getGradeResponse[]>(url, { withCredentials: true }));
    return response.map(element => element.value);

  } catch (error) {
    console.error('Błąd w ładowaniu ocen:', error);
    return [];
  }
}
```

Przykłady łączenia się klienta z serwerem

```
async getAllUsersSubjects(userId: number): Promise<Map<number, { id: number, name: string }[]>> {
  const url = `${this.userUrl}/${userId}/student/groups`;
  const userGroups = await this.getAllUserGroups(userId);
  const subjects = new Map<number, { id: number, name: string }[]>();

  for (const groupId of userGroups) {
    try {
      const response = await lastValueFrom(
        this.http.get<getResponseSubject[]>(`${url}/${groupId}/subjects`,
          { withCredentials: true }));
      const tempSubjects = response.map((element: { id: any; subject_name: any; }) => ({
        id: element.id,
        name: element.subject_name
      }));
      subjects.set(groupId, tempSubjects);

    } catch (error) {
      console.error(`Błęd ładowania danych groupId ${groupId}:`, error);
    }
  }

  return subjects;
}
```

Przykład widoku Django odpowiadającego klientowi na zapytania

```python
class GradeListCreateView(APIView):
    permission_classes = [IsAuthenticated]
    serializer_class = GradeSerializer

    def get(self, request, user_id, subject_id):
        student = get_object_or_404(Student, user_id=user_id)
        subject = get_object_or_404(SchoolSubject, id=subject_id)

        grades = Grade.objects.filter(
            student=student, grade_column__school_subject=subject)
        serializer = GradeSerializer(grades, many=True)
        return Response(serializer.data)

    def post(self, request, user_id, subject_id):
        student = get_object_or_404(Student, user_id=user_id)

        data = request.data.copy()
        data['student'] = student.user_id
        serializer = GradeSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Przykład widoku Django odpowiadającego klientowi na zapytania

```python
class StudentSubjectsView(APIView):
    permission_classes = [IsAuthenticated]
    serializer_class = SchoolSubjectSerializer

    def get(self, request, user_id, group_id):
        student = get_object_or_404(Student, user_id=user_id)
        group = get_object_or_404(StudentGroup, id=group_id, students=student)
        subjects = group.schoolsubject_set.all()
        serializer = SchoolSubjectSerializer(subjects, many=True)
        return Response(serializer.data)
```

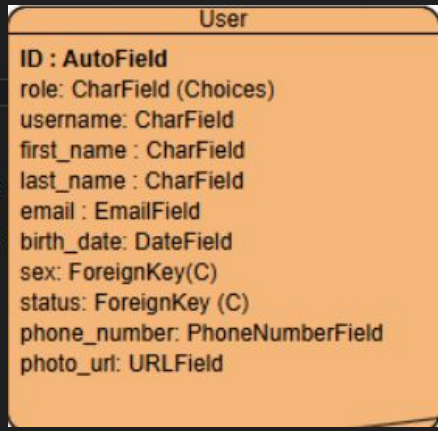
Przykład endpointów

- Grades API: /grades
  - GET - Get grades of currently authorized user
  - /{user_id}/{subject_id}/
    - GET - List student grades in given subject
    - POST - Create new grade for student in given subject
  - /{grade_id}/
    - GET - Get the grade data
    - PUT - Update the grade
  - /columns/{subject_id}
    - GET - Get grade columns for subject
    - POST - Add grade column for subject
    - DELETE - Delete grade column for subject
    - /{column_id}/
      - GET - Get grades for given column
      - PUT - Update column info
      - DELETE - Delete column

Przykład odpowiedzi serwera

```python
class User(AbstractUser):
    """ Username is user's unique identifier (inherited from AbstractUser) """
    ROLE_CHOICES = [
        ('student', 'Student'),
        ('parent', 'Parent'),
        ('teacher', 'Teacher'),
    ]
    role = models.CharField(max_length=10, choices=ROLE_CHOICES, editable=False
    first_name = models.CharField(max_length=255, default="name_example")
    last_name = models.CharField(max_length=255, default="Lname_example")
    email = models.EmailField(unique=True, default="example@example.com")
    birth_date = models.DateField(default=datetime.date(2010, 1, 1))
    sex = models.CharField(max_length=15, choices=[(
        "M", "Male"), ("F", "Female")], default="M")
    status = models.CharField(max_length=31, choices=[(
        "A", "Active"), ("U", "Inactive")], default="A")
    phone_number = models.CharField(max_length=15, blank=True, null=True)
    photo_url = models.URLField(blank=True, null=True)

    objects = UserManager()

    def __str__(self):
        return f"{self.username}: {self.first_name} {self.last_name}"
```

```
GET /user/

HTTP 200 OK
Allow: GET, PUT, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 8,
    "username": "student4",
    "first_name": "student4",
    "last_name": "student4",
    "email": "student4@student4.pl",
    "status": "A",
    "birth_date": "2000-11-11",
    "sex": "M",
    "phone_number": "123123123",
    "photo_url": "",
    "role": "student"
}
```



Przykład odpowiedzi serwera

```python
class Grade(models.Model):
    value = models.ForeignKey(
        CategoryGradeValue, on_delete=models.SET_NULL, null=True)
    timestamp = models.DateField(auto_now_add=True)
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    grade_column = models.ForeignKey("GradeColumn", on_delete=models.CASCADE)
    count_to_avg = models.BooleanField(default=True)

    def __str__(self):
        return f"{self.value} for {self.student}"


class GradeColumn(models.Model):
    title = models.CharField(max_length=255)
    weight = models.IntegerField(default=1)
    description = models.TextField(blank=True, default="")
    school_subject = models.ForeignKey(SchoolSubject, on_delete=models.CASCADE)

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


```
"results": [
    {
        "id": 5,
        "value": "GRADE5",
        "timestamp": "2024-12-04",
        "student": 8,
        "grade_column": 1,
        "count_to_avg": true
    },
    {
        "id": 6,
        "value": "GRADE4",
        "timestamp": "2024-12-04",
        "student": 8,
        "grade_column": 1,
        "count_to_avg": true
    },
    {
        "id": 7,
        "value": "GRADEABS",
        "timestamp": "2024-12-04",
        "student": 8,
        "grade_column": 1,
        "count_to_avg": true
    }
]
```


Dziękujemy za uwagę
Kryspin Kucha
Jakub Konecki
Illia Kuziv