React + Spring Boot Deployment

React'i Vercel'e, gerçek bir Spring Boot REST API'sini Render'a (Docker) deploy etmek, ortam değişkenleriyle backend adresini yönetmek, ve production CORS yapılandırması -- basit örneklerle.

Orta 14 dk

Son güncelleme: 6 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/react-spring-boot-deployment

---

React + Spring Boot Deployment

Build & Deployment dersinde bir React uygulamasını Vercel'e deploy ettik -- ama o uygulamanın hiç backend'i yoktu. Bu ders, React'in GERÇEK bir Spring Boot backend'ine bağlı olduğu, iki AYRI platforma deploy edilmiş bir uygulamayı anlatıyor.

Neden İki Ayrı Platform?

Vercel, statik siteler ve kısa ömürlü sunucusuz (serverless) fonksiyonlar için tasarlanmıştır -- Node.js/Python gibi runtime'ları destekler, ama Spring Boot'un ihtiyaç duyduğu SÜREKLİ ÇALIŞAN bir Java sunucu sürecini (embedded Tomcat) barındıramaz. Bu yüzden gerçek dünyada yaygın bir desen: React'i Vercel'e, Spring Boot backend'ini ise sürekli çalışan sunuculara uygun bir platforma (bu derste **Render**) AYRI AYRI deploy etmek.

Backend Adresini Ortam Değişkeninden Okumak

Build & Deployment dersindeki desenin aynısı, bu kez backend'in adresi için:

```
// The same pattern as in the Build & Deployment lesson -- except this time
// the value is the REAL address of a deployed Spring Boot API. Locally
// (against the backend running on localhost:8080) and in production
// (against the address Render provides), the SAME code works with a
// different .env / Vercel environment variable value.
const API_BASE_URL = import.meta.env.VITE_API_BASE_URL ?? "http://localhost:8080";

function ApiAddress() {
  return (
    <p>
      Data from <code>{API_BASE_URL}</code>
    </p>
  );
}

export default ApiAddress;
```

Yerelde `VITE_API_BASE_URL`, `localhost:8080`'de çalışan Spring Boot'u gösterir; production'da (Vercel'in ortam değişkenlerinde) Render'ın verdiği GERÇEK adresi gösterir. Kodun kendisi hiç değişmez.

Deploy Edilmiş Backend'den Veri Çekmek

API ve Veri Çekme kategorisindeki `useEffect` + `fetch` +loading/error deseni, artık gerçek bir deploy'a karşı çalışıyor:

```
import { useEffect, useState } from "react";

// The useEffect+fetch+loading/error pattern from the API & Data Fetching
// lesson -- except on the other end there's no longer json-server, but a
// REAL Spring Boot application deployed to Render.
const API_BASE_URL = import.meta.env.VITE_API_BASE_URL ?? "http://localhost:8080";

function CourseList() {
  const [courses, setCourses] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    async function loadCourses() {
      try {
        const response = await fetch(`${API_BASE_URL}/api/courses`);
        if (!response.ok) {
          throw new Error(`Request failed: ${response.status}`);
        }
        setCourses(await response.json());
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    }

    loadCourses();
  }, []);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Could not reach the backend: {error}</p>;

  return (
    <ul>
      {courses.map((course) => (
        <li key={course.id}>{course.name}</li>
      ))}
    </ul>
  );
}

export default CourseList;
```

Bu component'in davranışı, karşıdaki `json-server`'dan gerçek bir Spring Boot uygulamasına geçmiş olmasından hiç etkilenmiyor -- `fetch`'in gördüğü tek şey bir URL ve bir JSON yanıtı.

CORS: Deploy Edilmiş Frontend'e İzin Vermek

Advanced Spring MVC dersinde `addCorsMappings` ile GENEL bir CORS yapılandırması yazmıştık. Gerçek bir deploy'da bu, sabit kodlanmış bir domain yerine bir ortam değişkeninden okunur:

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

// This is what GlobalCorsConfigExample (addCorsMappings) from the Advanced
// Spring MVC lesson looks like when used in a real deployment. allowedOrigin
// is read from a CORS_ALLOWED_ORIGIN environment variable via
// application.properties -- without changing the code at all, it's enough
// to fill in this variable in the Render Dashboard with the real address
// Vercel provides.
@Configuration
class DeploymentCorsConfigExample implements WebMvcConfigurer {

    @Value("${app.cors.allowed-origin}")
    private String allowedOrigin;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
                .allowedOrigins(allowedOrigin, "http://localhost:5173")
                .allowedMethods("GET");
    }
}
```

`allowedOrigin`, `application.properties` üzerinden `CORS_ALLOWED_ORIGIN` ortam değişkeninden geliyor -- Render Dashboard'ında bu değişkeni Vercel'in verdiği GERÇEK adresle doldurmak yeterli, kod hiç değişmiyor. Origin tam olarak eşleşmezse (`https://` dahil, sonda `/` OLMADAN), tarayıcı isteği CORS hatasıyla engeller.

Deploy Sırası: Tavuk-Yumurta Problemi

React'in backend'in adresini, backend'in de React'in adresini bilmesi gerektiği için, sıra önemli:

1. Önce **backend'i Render'a deploy et** -- sana bir URL verir.
2. Sonra **React'i Vercel'e deploy et**, `VITE_API_BASE_URL`'i 1. adımdaki URL ile doldur -- sana BAŞKA bir URL verir.
3. Render'a geri dön, `CORS_ALLOWED_ORIGIN`'i 2. adımdaki URL ile doldur. Bu, backend'i otomatik olarak yeniden başlatır.

Bu sıra atlanırsa (ör. CORS değişkeni hiç doldurulmazsa), frontend backend'e ulaşır ama tarayıcı yanıtı JavaScript'e vermeyi REDDEDER -- Network sekmesinde isteğin başarılı (200) göründüğü ama konsolda bir CORS hatası olduğu bir durumla karşılaşırsın.

Özet ve Terimler Sözlüğü

React ve Spring Boot, farklı platformlara (Vercel + Render) AYRI AYRI deploy edilir -- her ikisi de birbirinin adresini bir ortam değişkeni üzerinden bilir. Backend'deki CORS yapılandırması,

sabit kodlanmış bir domain yerine bir ortam değişkeninden okunarak, kod değişmeden hangi frontend'e izin verileceği ayarlanabilir. Deploy sırası önemlidir: backend önce, sonra frontend, sonra backend'in CORS ayarı güncellenir.

Terimler Sözlüğü

CORS (Cross-Origin Resource Sharing) — Bir tarayıcının, farklı bir origin'deki (domain) bir sunucudan gelen yanıtı JavaScript'e vermesine izin veren mekanizma.

Origin — Şema + host + port üçlüsü (ör. `https://example.vercel.app`).

Pratik Proje

Bu derste öğrendiğimiz kavramları kullanan, GERÇEKTEN deploy edilmiş bir proje var:

- Backend (Render): **fullstack-deployment-backend-zst1.onrender.com**
- Frontend (Vercel): **React + Spring Boot Deployment Demo**
- Kaynak kod: **fullstack-deployment (React)** ve **backend/fullstack-deployment (Spring Boot)**

Canlı sayfayı açtığında, `GET /api/courses`'tan gelen "Java", "React", "Spring Boot" listesini görürsün -- CORS doğru yapılandırıldığı için tarayıcı, farklı bir domain'deki (`onrender.com`) bu yanıtı React koduna vermeyi kabul ediyor.

```
git clone https://github.com/cdurgun/react-course-projects.git
cd react-course-projects

# Backend (Java 21 + Maven gerekiyor)
cd backend/fullstack-deployment
mvn spring-boot:run

# Ayrı bir terminalde: frontend
cd ../../
npm install
cp projects/fullstack-deployment/.env.example projects/fullstack-deployment/.env
cd projects/fullstack-deployment
npm run dev
```