Software Design Document

Ahmet Cemil Tarık Keskinkılıç

July 23, 2025

Contents

1 Yazar Listesi

- Ahmet Cemil Tarık Keskinkılıç 2-8
- Gültekin Erişik 9

2 Yüksek Seviyeli Mimari Açıklama

Öğrenci Kart Sistemi, modern bir istemci-sunucu (client-server) mimarisi üzerine inşa edilmiştir. Bu mimari yaklaşım, uygulamanın farklı katmanlarını ayırarak geliştirme, bakım ve ölçeklenebilirlik süreçlerini kolaylaştırmaktadır. Sistem, temel olarak üç ana katmandan oluşmaktadır: İstemci Katmanı (Client Layer), Uygulama Katmanı (Application Layer - Backend) ve Veritabanı Katmanı (Database Layer).

- **İstemci Katmanı:** Bu katman, son kullanıcıların sistemle etkileşim kurduğu arayüzleri temsil eder. Web tarayıcıları veya mobil uygulamalar aracılığıyla erişilebilir. Kullanıcı arayüzü (UI) ve kullanıcı deneyimi (UX) bu katmanda sağlanır. İstemci, HTTP/HTTPS protokolleri üzerinden Uygulama Katmanı'ndaki API'lere istek gönderir ve yanıtları işleyerek kullanıcıya sunar.
- **Uygulama Katmanı (Backend):** Sistemin iş mantığını ve veri işleme süreçlerini barındıran merkezi katmandır. İstemci katmanından gelen istekleri alır, gerekli iş mantığını uygular ve veritabanı katmanıyla etkileşime girer. Bu katman, Node.js ve Express.js kullanılarak geliştirilmiştir, bu da hızlı ve ölçeklenebilir bir sunucu tarafı uygulama oluşturulmasına olanak tanır. API (Uygulama Programlama Arayüzü) uç noktaları bu katmanda tanımlanır ve istemcilerin veri alışverişi yapmasını sağlar.
- **Veritabanı Katmanı:** Uygulama tarafından depolanan ve yönetilen tüm verilerin bulunduğu katmandır. Öğrenci bilgileri, kart bilgileri, dersler, kafeterya menüleri gibi tüm yapısal veriler bu katmanda saklanır. Sistem, ilişkisel bir veritabanı olan MySQL kullanmaktadır. Uygulama katmanı, veritabanı ile güvenli ve verimli bir şekilde iletişim kurarak veri okuma, yazma, güncelleme ve silme işlemlerini gerçekleştirir.

Bu katmanlı mimari, her katmanın kendi sorumluluklarına sahip olmasını sağlayarak sistemin daha düzenli, yönetilebilir ve hata ayıklanabilir olmasını sağlar. Ayrıca, her katman bağımsız olarak geliştirilip test edilebilir, bu da geliştirme sürecini hızlandırır ve olası bağımlılık sorunlarını azaltır.

3 Veri Modeli

Öğrenci Kart Sistemi, öğrenci, kart, ders ve diğer ilgili bilgileri yönetmek için bir veritabanı kullanır. Temel veri modelleri ve ilişkileri aşağıda açıklanmıştır.

3.1 Kullanıcı (Users) Tablosu

Alan Adı	Veri Tipi	Kısıtlamalar	Açıklama
user_id	INT	PRIMARY KEY, AUTO_INCREMENT	Kullanıcının benzersiz kimliği
username	VARCHAR(255)	UNIQUE, NOT NULL	Kullanıcı adı
password	VARCHAR(255)	NOT NULL	Şifrenin hashlenmiş hali
email	VARCHAR(255)	UNIQUE, NOT NULL	Kullanıcının e-posta adresi
role	VARCHAR(50)	NOT NULL	Kullanıcı rolü (örn: student, admin)
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Kayıt tarihi

3.2 Öğrenci Kartı (Student_Cards) Tablosu

Alan Adı	Veri Tipi	Kısıtlamalar	Açıklama
card_id	INT	PRIMARY KEY, AUTO_INCREMENT	Kartın benzersiz kimliği
user_id	INT	FOREIGN KEY (Users)	Kartın ait olduğu kullanıcının kimliği
card_number	VARCHAR(255)	UNIQUE, NOT NULL	Kart numarası
balance	DECIMAL(10, 2)	DEFAULT 0.00	Kart bakiyesi
status	VARCHAR(50)	DEFAULT 'active'	Kart durumu (örn: active, inactive, lost)
issue_date	DATE	NOT NULL	Kartın verildiği tarih
expiry_date	DATE	NOT NULL	Kartın son kullanma tarihi

3.3 Dersler (Courses) Tablosu

Alan Adı	Veri Tipi	Kısıtlamalar	Açıklama
course_id	INT	PRIMARY KEY, AUTO_INCREMENT	Dersin benzersiz kimliği

course_name	VARCHAR(255)	NOT NULL	Ders adı
course_code	VARCHAR(50)	UNIQUE, NOT NULL	Ders kodu
credits	INT	NOT NULL	Dersin kredisi
capacity	INT	NOT NULL	Dersin maksimum öğrenci kapasitesi

3.4 Kayıtlar (Enrollments) Tablosu

Alan Adı	Veri Tipi	Kısıtlamalar	Açıklama
enrollment_id	INT	PRIMARY KEY, AUTO_INCREMENT	Kaydın benzersiz kimliği
user_id	INT	FOREIGN KEY (Users)	Kayıt olan öğrencinin kimliği
course_id	INT	FOREIGN KEY (Courses)	Kayıt olunan dersin kimliği
enrollment_date	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Kayıt tarihi

3.5 Kafeterya Menüsü (Cafeteria_Menu) Tablosu

Alan Adı	Veri Tipi	Kısıtlamalar	Açıklama
item_id	INT	PRIMARY KEY, AUTO_INCREMENT	Menü öğesinin benzersiz kimliği
item_name	VARCHAR(255)	NOT NULL	Menü öğesi adı
description	TEXT		Menü öğesi açıklaması
price	DECIMAL(10, 2)	NOT NULL	Menü öğesi fiyatı
category	VARCHAR(100)		Menü öğesi kategorisi (örn: main, dessert, drink)

3.6 Sınıflar (Classrooms) Tablosu

Alan Adı	Veri Tipi	Kısıtlamalar	Açıklama
classroom_id	INT	PRIMARY KEY, AUTO_INCREMENT	Sınıfın benzersiz kimliği

classroom_name	VARCHAR(255)	NOT NULL	Sınıf adı
capacity	INT	NOT NULL	Sınıf kapasitesi

3.7 Arkadaşlar (Friends) Tablosu

Alan Adı	Veri Tipi	Kısıtlamalar	Açıklama
friendship_id	INT	PRIMARY KEY, AUTO_INCREMENT	Arkadaşlık ilişkisinin benzersiz kimliği
user_id1	INT	FOREIGN KEY (Users)	Birinci kullanıcının kimliği
user_id2	INT	FOREIGN KEY (Users)	İkinci kullanıcının kimliği
status	VARCHAR(50)	DEFAULT 'pending'	Arkadaşlık durumu (örn: pending, accepted, rejected)

4 Veri Doğrulama Kuralları

Sistemde veri bütünlüğünü ve güvenliğini sağlamak için çeşitli doğrulama kuralları uygulanacaktır:

- **Genel Doğrulama Kuralları:**
  - Tüm gerekli alanlar boş bırakılamaz (NOT NULL).
  - Benzersiz olması gereken alanlar (örn: `username`, `email`, `card_number`, `course_code`) için benzersizlik kontrolü yapılacaktır.
  - String formatındaki veriler için maksimum uzunluk kontrolü yapılacaktır.
- **Öğrenci Kartı (Student_Cards) Doğrulama Kuralları:**
  - `balance`: Bakiye değeri negatif olamaz. Minimum bakiye 0.00 olmalıdır. Maksimum bakiye sistemin belirlediği bir üst sınırı (örn: 1000.00 TL) aşamaz. Formatı ondalıklı sayı (float) olmalıdır.
  - `card_number`: Belirli bir formatı takip etmelidir (örn: 16 haneli sayısal bir değer). Uzunluk ve karakter tipi kontrolü yapılacaktır.
  - `issue_date` ve `expiry_date`: Geçerli tarih formatında olmalı ve `expiry_date` `issue_date`'den sonra olmalıdır.
  - `status`: Belirli değerlerle sınırlı olmalıdır (örn: 'active', 'inactive', 'lost').
- **Kullanıcı (Users) Doğrulama Kuralları:**

- `username`: Minimum 3, maksimum 50 karakter uzunluğunda olmalı ve özel karakter içermemelidir.
- `password`: Minimum 8 karakter uzunluğunda olmalı, en az bir büyük harf, bir küçük harf, bir rakam ve bir özel karakter içermelidir.
- `email`: Geçerli bir e-posta formatında olmalıdır.
- `role`: Belirli değerlerle sınırlı olmalıdır (örn: 'student', 'admin').

- **Dersler (Courses) Doğrulama Kuralları:**
  - `credits`: Pozitif bir tam sayı olmalıdır.
  - `capacity`: Pozitif bir tam sayı olmalı ve 0'dan büyük olmalıdır.

- **Kafeterya Menüsü (Cafeteria_Menu) Doğrulama Kuralları:**
  - `price`: Pozitif bir ondalıklı sayı olmalıdır.
  - `category`: Belirli değerlerle sınırlı olabilir (örn: 'main', 'dessert', 'drink').

Bu doğrulama kuralları, hem frontend hem de backend katmanlarında uygulanarak veri tutarlılığı ve güvenliği sağlanacaktır.

5 ER Diyagramı

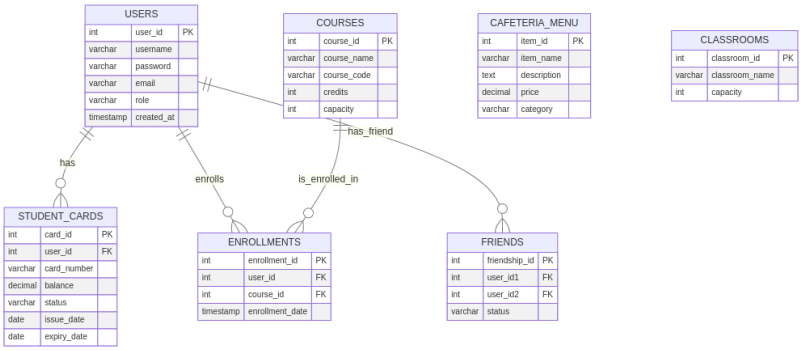

Figure 1: ER Diyagramı

6 UML Diyagramları

6.1 Sınıf Diyagramı (Class Diagram)

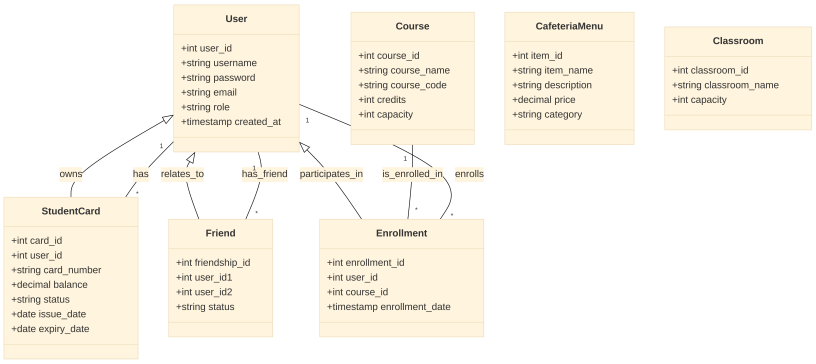

Figure 2: Sınıf Diyagramı

7 Pattern Açıklamaları

Öğrenci Kart Sistemi, yazılımın esnekliğini, yeniden kullanılabilirliğini ve sürdürülebilirliğini artırmak amacıyla çeşitli tasarım desenlerini (design patterns) kullanmayı hedeflemektedir. Bu desenler, yaygın yazılım tasarım problemlerine kanıtlanmış çözümler sunar ve sistemin daha sağlam bir yapıya sahip olmasına yardımcı olur.

7.1 Singleton Pattern

Açıklama: Singleton deseni, bir sınıfın yalnızca tek bir örneğinin (instance) olmasını ve bu örneğe global bir erişim noktası sağlamasını garanti eder. Bu desen, veritabanı bağlantıları, konfigürasyon yöneticileri veya loglama servisleri gibi kaynakların tek bir noktadan yönetilmesi gerektiği durumlarda kullanışlıdır.

Uygulama: Öğrenci Kart Sistemi bağlamında, veritabanı bağlantı havuzu (database connection pool) için Singleton deseni uygulanabilir. Bu, uygulamanın veritabanına her ihtiyaç duyduğunda yeni bir bağlantı oluşturmak yerine, mevcut tek bir bağlantı havuzunu kullanmasını sağlar. Bu yaklaşım, kaynak tüketimini optimize eder ve veritabanı bağlantılarının tutarlı bir şekilde yönetilmesini garanti eder.

7.2 Factory Method Pattern

Açıklama: Factory Method deseni, bir nesne oluşturma sorumluluğunu alt sınıflara bırakarak, bir sınıfın hangi tür nesneleri oluşturacağını alt sınıfların

belirlemesine olanak tanır. Bu desen, sistemin farklı türde nesneler (örneğin, farklı kart tipleri veya kullanıcı rolleri) oluşturması gerektiğinde ve bu nesnelerin oluşturulma mantığının merkezi bir yerden yönetilmesi istendiğinde faydalıdır.

Uygulama: Öğrenci Kart Sistemi içinde, farklı öğrenci kartı tipleri (örneğin, normal öğrenci kartı, misafir öğrenci kartı, personel kartı) olabilir. Her kart tipinin kendine özgü özellikleri veya doğrulama kuralları bulunabilir. Factory Method deseni kullanılarak, `StudentCardFactory` gibi bir fabrika sınıfı oluşturulabilir. Bu fabrika, gelen isteğe göre doğru kart tipini oluşturur ve döndürür. Bu sayede, yeni bir kart tipi eklendiğinde mevcut kodda büyük değişiklikler yapmaya gerek kalmaz, sadece fabrika sınıfına yeni bir oluşturma metodu eklenmesi yeterli olur.

7.3 Observer Pattern

Açıklama: Observer deseni, bir nesne (subject) durumunda bir değişiklik olduğunda, ona bağımlı olan tüm nesnelerin (observers) otomatik olarak bilgilendirilmesini sağlar. Bu desen, olay tabanlı sistemlerde veya bir durum değişikliğinin birden fazla bileşeni etkilemesi gerektiği durumlarda kullanılır.

Uygulama: Öğrenci Kart Sistemi içinde, bir öğrencinin kart bakiyesi değiştiğinde (örneğin, kafeteryada harcama yaptığında veya bakiye yüklendiğinde) bu durumun ilgili diğer sistemlere (örneğin, bildirim servisi, loglama servisi) bildirilmesi gerekebilir. Observer deseni kullanılarak, `StudentCard` nesnesi bir subject olarak tanımlanabilir ve `NotificationService` veya `LoggingService` gibi servisler observer olarak kaydedilebilir. Kart bakiyesi değiştiğinde, subject tüm observerları bilgilendirir ve ilgili servisler kendi işlemlerini (bildirim gönderme, log kaydı tutma) gerçekleştirir. Bu, sistemin gevşek bağlı (loosely coupled) olmasını sağlar ve modüller arası bağımlılığı azaltır.

7.4 Iterator Pattern

Açıklama: Iterator deseni, bir koleksiyonun öğeleri üzerinde ardışık ve soyut bir şekilde gezinmeyi sağlar, dışarıdan erişim mantığını gizler.

Uygulama: Günlük menü servisinde, backend'de `CafeteriaMenuIterator` ile tüm `CafeteriaMenuItem` nesneleri üzerinde gezinilip JSON'a dönüştürülür. Böylece client kodu menü yapısının iç detaylarına bağlı kalmadan aynı arabirimi kullanır.

7.5 Strategy Pattern

Açıklama: Strategy deseni, farklı algoritmaları çalışma zamanında seçilebilir kılar; her biri ortak bir arabirimi uygular.

Uygulama: Boş sınıf listeleme özelliğinde, `RoomFilterStrategy` arabirimi üzerinden "Bina Göre Filtreleme" ve "Kapasiteye Göre Filtreleme" stratejileri uygulanır. Yeni filtreler eklenmesi gerektiğinde mevcut kodu değiştirmeden sadece yeni bir strateji sınıfı yazmak yeterlidir.

8 Veri Akışı

Öğrenci Kart Sistemi içindeki veri akışı, istemci katmanından başlayarak backend ve veritabanı katmanlarına doğru ilerler ve yanıtlar ters yönde hareket eder. Bu akış, sistemin farklı bileşenleri arasında bilginin nasıl iletildiğini ve işlendiğini gösterir.

8.1 Kullanıcı Girişi Akışı

1. **İstemci:** Kullanıcı, web arayüzü veya mobil uygulama üzerinden kullanıcı adı ve şifresini girer.
2. **İstemci -> Backend (API):** İstemci, bu kimlik bilgilerini bir HTTP POST isteği olarak `/api/users/login` gibi bir API uç noktasına gönderir.
3. **Backend (Kullanıcı Yönetimi Modülü):** Backend, gelen isteği alır. Kullanıcı adı ve şifreyi doğrulamak için `Users` tablosunu sorgular. Şifre, `bcrypt` kullanılarak hashlenmiş şifre ile karşılaştırılır.
4. **Backend -> Veritabanı:** Backend, kullanıcı bilgilerini ve hashlenmiş şifreyi veritabanından çeker.
5. **Veritabanı -> Backend:** Veritabanı, sorgu sonucunu (kullanıcı bilgileri) backend'e döndürür.
6. **Backend:** Doğrulama başarılı olursa, backend bir JSON Web Token (JWT) oluşturur ve bu token'ı yanıt olarak hazırlar. Başarısız olursa, uygun bir hata mesajı oluşturulur.
7. **Backend (API) -> İstemci:** Backend, JWT'yi veya hata mesajını HTTP yanıtı olarak istemciye gönderir.
8. **İstemci:** İstemci, JWT'yi alır ve sonraki kimliği doğrulanmış istekler için saklar (örn: `localStorage` veya `sessionStorage`). Kullanıcı arayüzü, başarılı giriş durumuna göre güncellenir.

8.2 Öğrenci Kartı Bakiyesi Sorgulama Akışı

1. **İstemci:** Kullanıcı, öğrenci kartı bakiyesini görüntülemek için arayüzdeki ilgili butona tıklar veya sayfayı açar.
2. **İstemci -> Backend (API):** İstemci, kullanıcının kimlik doğrulama token'ını (JWT) içeren bir HTTP GET isteği olarak `/api/student_cards/balance` gibi bir API uç noktasına gönderir.
3. **Backend (Öğrenci Kartı Yönetimi Modülü):** Backend, gelen isteği alır. JWT'yi doğrular ve kullanıcının kimliğini çıkarır. Bu kimlikle `Student_Cards` tablosunu sorgulayarak kullanıcının kart bakiyesini bulur.

4. **Backend -> Veritabanı:** Backend, `user_id`'ye göre `Student_Cards` tablosundan bakiye bilgisini çeker.

5. **Veritabanı -> Backend:** Veritabanı, sorgu sonucunu (bakiye değeri) backend'e döndürür.

6. **Backend:** Backend, bakiye bilgisini JSON formatında bir yanıt olarak hazırlar.

7. **Backend (API) -> İstemci:** Backend, bakiye bilgisini HTTP yanıtı olarak istemciye gönderir.

8. **İstemci:** İstemci, bakiye bilgisini alır ve kullanıcı arayüzünde görüntüler.

8.3 Ders Kaydı Akışı

1. **İstemci:** Öğrenci, mevcut dersleri listeler ve kaydolmak istediği dersi seçer.

2. **İstemci -> Backend (API):** İstemci, seçilen dersin `course_id`'sini ve kullanıcının kimlik doğrulama token'ını içeren bir HTTP POST isteği olarak `/api/enrollments/add` gibi bir API uç noktasına gönderir.

3. **Backend (Ders Yönetimi ve Kayıt Yönetimi Modülleri):** Backend, gelen isteği alır. JWT'yi doğrular ve öğrencinin kimliğini çıkarır. Dersin kapasitesini ve öğrencinin daha önce bu derse kayıtlı olup olmadığını kontrol etmek için `Courses` ve `Enrollments` tablolarını sorgular.

4. **Backend -> Veritabanı:** Backend, ders ve kayıt bilgilerini veritabanından çeker ve günceller.

5. **Veritabanı -> Backend:** Veritabanı, sorgu ve güncelleme sonuçlarını backend'e döndürür.

6. **Backend:** Tüm kontroller başarılı olursa, backend `Enrollments` tablosuna yeni bir kayıt ekler ve dersin kapasitesini günceller. Başarısız olursa, uygun bir hata mesajı oluşturulur.

7. **Backend (API) -> İstemci:** Backend, işlemin başarı durumunu veya hata mesajını HTTP yanıtı olarak istemciye gönderir.

8. **İstemci:** İstemci, yanıtı alır ve kullanıcıya ders kaydının başarılı olup olmadığını bildirir. Gerekirse ders listesini günceller.

8.4 Bakiye Yükleme Akışı

1. **İstemci:** Kullanıcı, kart bakiyesine TL yüklemek için arayüzde "Bakiye Yükle" formunu doldurur.

2. **İstemci → Backend (API):** HTTP POST isteği, JSON gövdesi `{"user_id":..., "amount":...}` ile `/api/student_cards/updateBalance` uç noktasına gönderilir.

3. **Backend (StudentCard Modülü):** `updateBalance` metodu tetiklenir. Factory Method deseni kullanılarak `BalanceOperationFactory` üzerinden bir `TopUpOperation` nesnesi üretilir.

4. **Veritabanı:** Mevcut bakiyeye belirtilen miktar eklenir.

5. **Backend → İstemci:** Güncellenmiş bakiye ve onay mesajı JSON olarak döner.

6. **İstemci:** Yanıtı alır, UI'da yeni bakiyeyi görüntüler.

8.5 Yemek Menüsü Görüntüleme Akışı

1. **İstemci:** "Günlük Menü" sekmesine tıklar.

2. **İstemci → Backend (API):** HTTP GET isteği `/api/cafeteria_menu` uç noktasına gönderilir.

3. **Backend (Cafeteria_Menu Modülü):** Tüm menü öğeleri alınır ve Iterator deseni kullanılarak `CafeteriaMenuIterator` ile üzerinde dönülür.

4. **Backend → İstemci:** Menü öğeleri JSON dizisi olarak gönderilir.

5. **İstemci:** Gelen diziyi liste halinde render eder.

8.6 Boş Sınıf Listeleme Akışı

1. **İstemci:** "Sınıflar" ekranında tarih ve saat seçimini yapar.

2. **İstemci → Backend (API):** HTTP GET isteği, parametreler `?date=YYYY-MM-DD&time=HH:mm` ile `/api/classrooms/available` uç noktasına gider.

3. **Backend (Classrooms Modülü):** Strategy deseni kullanılarak seçilen `RoomFilterStrategy` (`ByBuildingStrategy` veya `ByCapacityStrategy`) uygulanır ve boş sınıflar tespit edilir.

4. **Backend → İstemci:** Filtrelenmiş oda listesi JSON olarak döner.

5. **İstemci:** Boş sınıfları tablo veya kart görünümünde gösterir.

Bu veri akışı senaryoları, sistemin farklı katmanlarının nasıl etkileşimde bulunduğunu ve verilerin sistem içinde nasıl hareket ettiğini göstermektedir. API'ler, bu etkileşimlerin temelini oluşturur ve sistemin modülerliğini ve esnekliğini destekler.

9 Test Dökümanı

Yazar: Gültekin Erişik

9.1 Test Ortamının Kurulumu (setupTests.js)

React uygulamalarında bileşenlerin doğru çalışıp çalışmadığını kontrol etmek amacıyla `jest-dom` kütüphanesi kullanılır.

Bu kütüphane, DOM üzerinde özel eşleştiricilerle test yapmayı kolaylaştırır.

```
import '@testing-library/jest-dom';
```

Açıklama:

- `@testing-library/jest-dom` import edilerek Jest ortamına özel matcher'lar eklenir.
- Bu matcher'lar sayesinde daha okunabilir ve anlamlı testler yazmak mümkündür.
- Örnek: `expect(element).toBeInTheDocument()` gibi ifadeler kullanılabilir.

9.2 Birim Test: App Bileşeni (App.test.js)

Bu dosyada, uygulamanın ana bileşeni olan `App`'in doğru şekilde render edilip edilmediğini test eden bir örnek yer almaktadır.

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

Açıklama:

- `render(<App />)` komutu ile `App` bileşeni test ortamında render edilir.
- `screen.getByText(/learn react/i)` ile DOM'da ''learn react''metni aratılır.
- `expect(...).toBeInTheDocument()` ile bu metni içeren bir öğenin DOM'da bulunup bulunmadığı doğrulanır.

9.3 Genel Değerlendirme

Bu iki dosya, React Testing Library ve Jest kullanılarak test yazmanın temel adımlarını göstermektedir:

- `setupTests.js`: Jest ortamına özel matcher'ların yüklendiği yerdir.

- `App.test.js`: Bileşen render işleminin başarılı olup olmadığını kontrol eder.

Bu yapıyla birlikte daha karmaşık test senaryoları da yazılabilir, örneğin kullanıcı etkileşimleri, form doğrulamaları veya bileşen yaşam döngüsü testleri.