

Quality Assurance (QA) Plan

Proje Adı: TripGuide

Takım: TrueGuiders

Katkıda Bulunanlar

- İrem Keser
- Efe Selim Sürekli
- Zeynep Ay
- Elif Sema Küçük
- Saadet Cansu Baktıroğlu

1. Quality Assurance Strategy

1.1 Overview

Bu QA Planı, TripGuide projesinde yazılım kalitesini güvence altına almak için izlenecek süreçleri tanımlar.

Amaçlarımız:

- Hataları mümkün olduğunca erken aşamada yakalamak,
- Rota oluşturma ve gösterme akışının doğru ve tutarlı çalışmasını sağlamak,
- Kullanıcıların şehir ve gün seçimi yaparken rahat ve sorunsuz bir deneyim yaşamasını garanti etmek,
- Final demoda gösterilecek 5 use case'in hatasız ve stabil çalışmasını sağlamak.

1.2 Testing Methodologies

1. Unit Testler (Birim Testleri)

- Servis katmanındaki tek bir fonksiyonun doğru çalışıp çalışmadığını test eder.
- Örnekler:
 - Gezi puanlama fonksiyonunun 1–5 dışındaki değerleri reddetmesi.
- Bu testler geliştiriciler tarafından yazılır ve çalıştırılır.

2. Entegrasyon Testleri

- Farklı katmanların birlikte doğru çalışıp çalışmadığını kontrol eder.
- Örnekler:
 - Yazılan test senaryolarının birlikte doğru çalışıp çalışmadığını test eder.

3. Sistem Testleri (End-to-End)

- Kullanıcının gerçek senaryolarını uçtan uca test eder.

- Örnek akış:
 - Kullanıcı ana sayfada şehir ve gün sayısını seçer.
 - “Rota Oluştur” butonuna basar.
 - Öneri listesi ve zaman çizelgesi görüntülenir.
 - Kullanıcı planı puanlar.

4. Usability Test (Kullanılabilirlik Testleri)

- Arayüzün kullanıcı açısından ne kadar anlaşılır olduğunu ölçer.
- Sorgulanan noktalar:
 - Şehir ve gün seçim alanları yeterince açık mı?
 - “Rota Oluştur” butonu kolay fark ediliyor mu?
 - Hata mesajları (örneğin şehir seçmeden form gönderme) kullanıcı dostu mu?

1.3 Automated vs. Manual Testing

1. Otomatik Testler (Automated Testing)

- **Unit Testler (JUnit):**
 - Öneri motoru, puanlama fonksiyonu gibi kritik metodlar otomatik test edilir.
 - Her `mvn test` çalıştırıldığında bu testler de çalışır.
- **Entegrasyon Testleri (Spring Boot Test):**
 - Örneğin `TripControllerIntegrationTest` ile, API çağrısının DB’ye doğru yazıp yazmadığı test edilir.
- **UI Otomasyon Testleri (Selenium):**
 - Gerçek tarayıcıda şu akış otomatik denenir:
 - Ana sayfayı aç.
 - Bir şehir seç.
 - Gün sayısını seç.
 - “Rota Oluştur” butonuna tıkla.
 - Sonuç ekranında rota başlığının görüldüğünü doğrula.
- **CI/CD Pipeline Entegrasyonu :**
 - Repository’e push edilen her değişiklikte unit ve entegrasyon testleri otomatik koşturulabilir.
 - Testler başarısızsa build başarısız olur ve hatalı kodun ana branch’e merge edilmesi engellenir.

2. Manuel Testler (Manual Testing)

- **Usability Testleri:**
 - Gerçek kullanıcı (veya ekip üyeleri) arayüzü kullanarak senaryoları dener.
 - Sorulan sorular:
 - Formu doldurmak kolay mı?

- Hangi noktada kafa karışıklığı oldu?

- **Arayüz Kontrolleri (UI Testing):**

- Farklı ekran genişliklerinde (laptop, tablet) sayfanın bozulup bozulmadığı kontrol edilir.

- **Özel Senaryo Testleri:**

- Normal akış dışında hataya sebep olabilecek uç senaryolar test edilir:
 - Şehir seçmeden “Rota Oluştur”a basmak.
 - Veritabanı boşken rota oluşturma isteği atmak.

2. Quality Factors & Metrics

| Kalite Faktörü | Açıklama | Ölçüm Metriği |
|-----------------|-----------------------------------|----------------------------|
| Performance | Sistem yanıt süresi hızlı olmalı | Ortalama yanıt süresi (ms) |
| Security | Sistemin güvenli olması | Tespit edilen açık sayısı |
| Usability | Kullanıcı dostu arayüz | Memnuniyet puanı (1–5) |
| Maintainability | Kodun kolay düzenlenebilir olması | Cyclomatic Complexity |

3. Test Plan

3.1 Test Cases

Aşağıda 5 temel test senaryosu verilmiştir:

| Test Adı | Amaç | Giriş | Beklenen Çıktı |
|----------------------|---|--------------------|----------------------------|
| Şehir Seçimi Testi | Formdaki şehir seçimi doğru işliyor mu? | city = “Antalya” | Şehir başarıyla seçilir |
| Öneri Üretme Testi | Algoritma doğru filtreleme yapıyor mu? | Kategori: Müze | Müze kategorisi listelenir |
| Rota Oluşturma Testi | Rota planı doğru oluşturuluyor mu? | city=İzmir, days=3 | 3 günlük timeline döner |

| | | | |
|---------------------|---|-------------|---------------------------|
| Plan Kaydetme Testi | Oluşturulan plan DB'ye kaydediliyor mu? | Trip object | DB'de yeni kayıt oluşur |
| Gezi Puanlama Testi | Kullanıcı puanlama yapabiliyor mu? | score=5 | Puan başarıyla kaydedilir |

3.2 Bug Tracking

Araç: GitHub Issues

1. Hata Bildirimi

- Bir hata tespit edildiğinde geliştirici, test mühendisi veya kullanıcı tarafından yeni bir Issue açılır.
- Issue başlığı kısa, açıklama ise ayrıntılı olmalıdır. Açıklamada şunlar bulunur:
 - Hatanın nasıl ortaya çıktığı (adım adım)
 - Beklenen sonuç vs. gerçekleşen sonuç
 - Hata tekrar üretiliyor mu? (Yeniden üretim adımları)
 - Ekran görüntüsü veya hata mesajı varsa eklenir
- Bu aşama, hatanın resmî olarak kayıt altına alınmasını sağlar.

2. Hata Seviyesinin Belirlenmesi

Her hata etkisine göre sınıflandırılır:

- **Önemli:**
 - Uygulamanın ana fonksiyonlarını durduran hatalar.
 - Örneğin rota oluşturma fonksiyonunun tamamen çalışmaması.
- **Yüksek:**
 - Önemli fonksiyonların hatalı çalışmasına sebep olur fakat sistem tamamen çökmez.
 - Örneğin filtreleme çalışıyor ama yanlış mekanları listeliyor.
- **Orta:**
 - Ana fonksiyonlarda soruna yol açmayan fakat kullanıcı deneyimini olumsuz etkileyen hatalar.
 - Örneğin timeline bazı ekranlarda kayıyor.
- **Düşük:**
 - Görsel hatalar veya kullanım kolaylığını az etkileyen sorunlar.
 - Örneğin ikon hizasının yanlış olması.

Hata seviyesi belirlendikten sonra Issue üzerine uygun label eklenir.

3. Sorunların Çözümü ve Geliştirici Atamaları

- Issue açıldıktan sonra proje yöneticisi (PM) veya ilgili geliştirici, hatayı çözmesi gereken kişiyi belirler. İlgili kısım ile ilgili sorun o kısım için görevlendirilen kişi sorumlu olarak atanır.

Sorumlu atandıktan sonra:

- Geliştirici gerekli kod düzeltmelerini yapar,
- Değişiklikleri açıklayan bir commit mesajı ekler,
- Gerekirse Pull Request açarak çözümü paylaşır.

4. Düzeltme Sonrası Test (Retest)

- Hata düzeltildikten sonra QA sorumlusu hatayı yeniden test eder.
- Testte:
 - Aynı adımlar tekrar uygulanır.
 - Hatanın artık görünmediği doğrulanır.
 - Yeni bir yan etki oluşturmadığı kontrol edilir.
- Eğer hata tamamen çözülmediyse Issue yeniden açılır veya yorum eklenir.

5. Kapatma

1. Retest başarılıysa Issue kapatılır.
2. Kapatılan Issue, bu hatanın artık TripGuide sisteminde çözülmüş olduğunu gösterir.
3. Eğer ileride hata tekrar görülürse Issue yeniden açılır veya yeni bir Issue oluşturulur.