ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ
MÜHENDİSLİK-MİMARLIK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

YAZILIM TESTİ

PyCodeFlow Testi

Yusuf Kenan AKSAN **152120181012**

Yunus AKYOL **152120181045**

Efekan SARGIN **152120181049**

Hüseyin Can ERGÜN **152120181068**

Muhammed Talha ŞAHİN **152120181071**

Yazılım Testi Proje Raporu
Dr. Uğur YAYAN Hotkey / PyCodeFlow Projesi ve İnnova Bilişim

Katkılar ve Bölümler;

Muhammed Talha ŞAHİN: Arayüz otomasyonu testleri ve sonuçları kısmı doldurulduu.

Yunus AKYOL: Sistem ve entegrasyon testleri ve sonuçları kısmı doldurulduu.

Hüseyin Can ERGÜN: Beyaz kutu test teknikleri kısmı doldurulduu.

Efekan SARGIN: Birim testleri ve sonuçları kısmı doldurulduu.

Yusuf Kenan AKSAN: Kara kutu ve tecrübeye dayalı sistem test sonuçları kısmı doldurulduu.

Revizyon Tarihçesi

Versiyon	Tarih	Yazar	Yorum
v.1.0	**03.11.2022**	**Yusuf Kenan Aksan Yunus Akyol Efekan Sargın Hüseyin Can Ergün M. Talha Şahin**	**Kullanım Senaryosu, değerlendirme senaryoları ve kriterleri eklendi.**
v.2.0	**24.11.2022**	**Yusuf Kenan Aksan Yunus Akyol Efekan Sargın Hüseyin Can Ergün M. Talha Şahin**	**Test Aktiviteleri Faaliyetleri başlığı altında test planlama ve hedeflenen statik testler ve gözden geçirme planı eklendi.**
v.3.0	**15.12.2022**	**Yusuf Kenan Aksan Yunus Akyol Efekan Sargın Hüseyin Can Ergün M. Talha Şahin**	**Yapılan statik testler ve gözden geçirmeler eklendi. Ele alınacak olan birim, entegrasyon ve sistem testlerinin raporlanması gerçekleştirildi. Uygulanacak test teknikleri kısmı eklendi**
v.4.0	**05.01.2023**	**Yusuf Kenan Aksan Yunus Akyol Efekan Sargın Hüseyin Can Ergün M. Talha Şahin**	**Birim, entegrasyon, arayüz ve sistem testleri gerçekleştirilip raporlamaları yapıldı. Test uygulamaları ve sonuçları raporlandı.**

İÇİNDEKİLER

1. GİRİŞ VE TANITIM

İnnova Bilişim Çözümleri A.Ş, farklı teknolojilerde bilgi birikimine sahip 1400 kişinin üzerindeki profesyonel kadrosu ile Türkiye'nin önde gelen bilişim çözümleri firmalarından birisidir. 1999'dan bugüne telekomünikasyon, finans, üretim, kamu ve hizmet sektörleri başta olmak üzere her sektördeki kuruluşlara platform bağımsız çözümler sunan İnnova, uluslararası standartlarda ürettiği çözümleri şimdiye kadar 4 kıtada 37 ülkeye ihraç etmeyi başarmıştır. 2007 yılından bu yana Türk Telekom iştiraki olan İnnova Bilişim Çözümleri A.Ş, İstanbul ve Ankara ana ofisleri ile beraber Türkiye'nin çeşitli bölgelerine yayılmış toplamda 14 ofis üzerinden faaliyetlerine devam etmektedir.

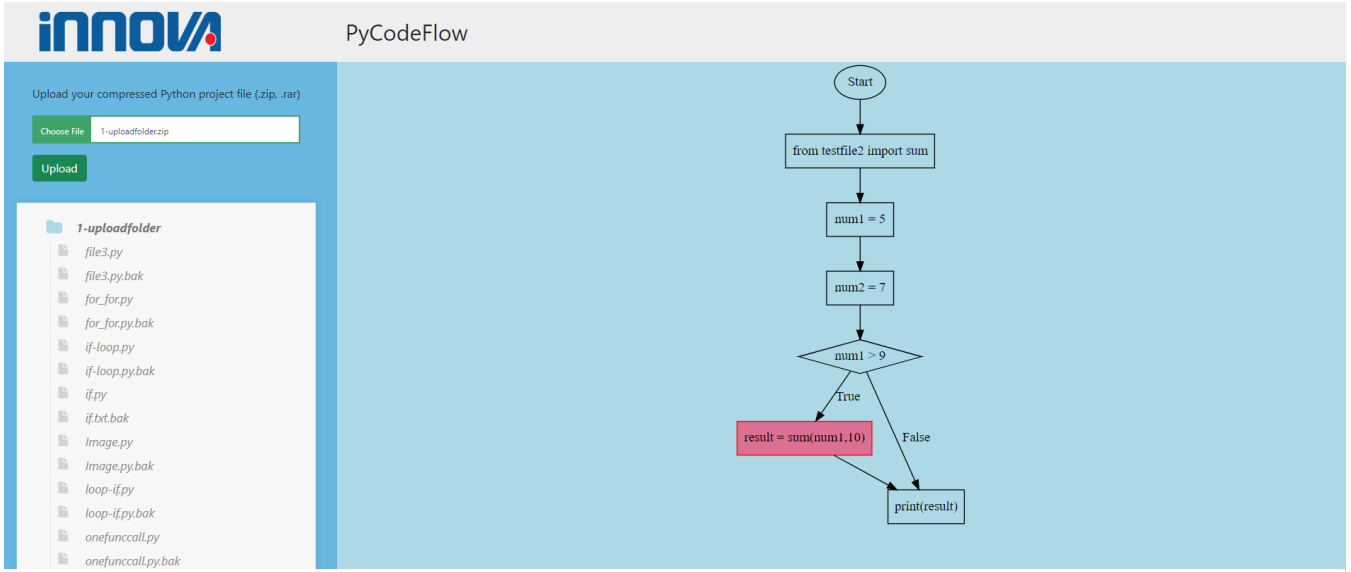
İnnova, başta danışmanlık hizmetleri, altyapı, kurumsal çözümler ve yazılım geliştirme olmak üzere bulut hizmetleri, sistem yazılımı ve bunun gibi birçok alanda geniş kapsamlı faaliyet göstermektedir. Şimdiye kadar sunduğu hizmetler dahilinde kamu ve özel sektör tarafından talep edilen sayısız ürünün geliştirilmesi sağlanmıştır. Bunlara örnek olarak kamu alanında T.C. Sağlık Bakanlığı için yapılan Hayat Eve Sığar (HES) projesi, özel sektör için ise Burj Khalifa içerisinde bulunan kiosk projesi verilebilir. Ayrıca, Ar-Ge alanında da uluslararası kapsamda avrupa birliği tarafından fonlanan birçok proje dahilinde çalışmalar yapılmakta, çözümler üretilmektedir. Bunun yanında, İnnova'nın dünya genelinde başta Apple, Microsoft, Huawei olmak üzere birçok dünyaca ünlü firma ile iş ortaklığı bulunmaktadır.

Yazılım Testi dersi kapsamında yapılacak olan proje İnnova Bilişim şirketinin bir aracı olan PyCodeFlow aracının yazılım testini gerçekleştirilip, değerlendirmelerini sunmaktır. Bu kapsamda İnnova Bilişim şirketi bünyesinde geliştirilen PyCodeFlow, Python projelerinin akış diyagramlarını çıkaran bir web uygulamasıdır.

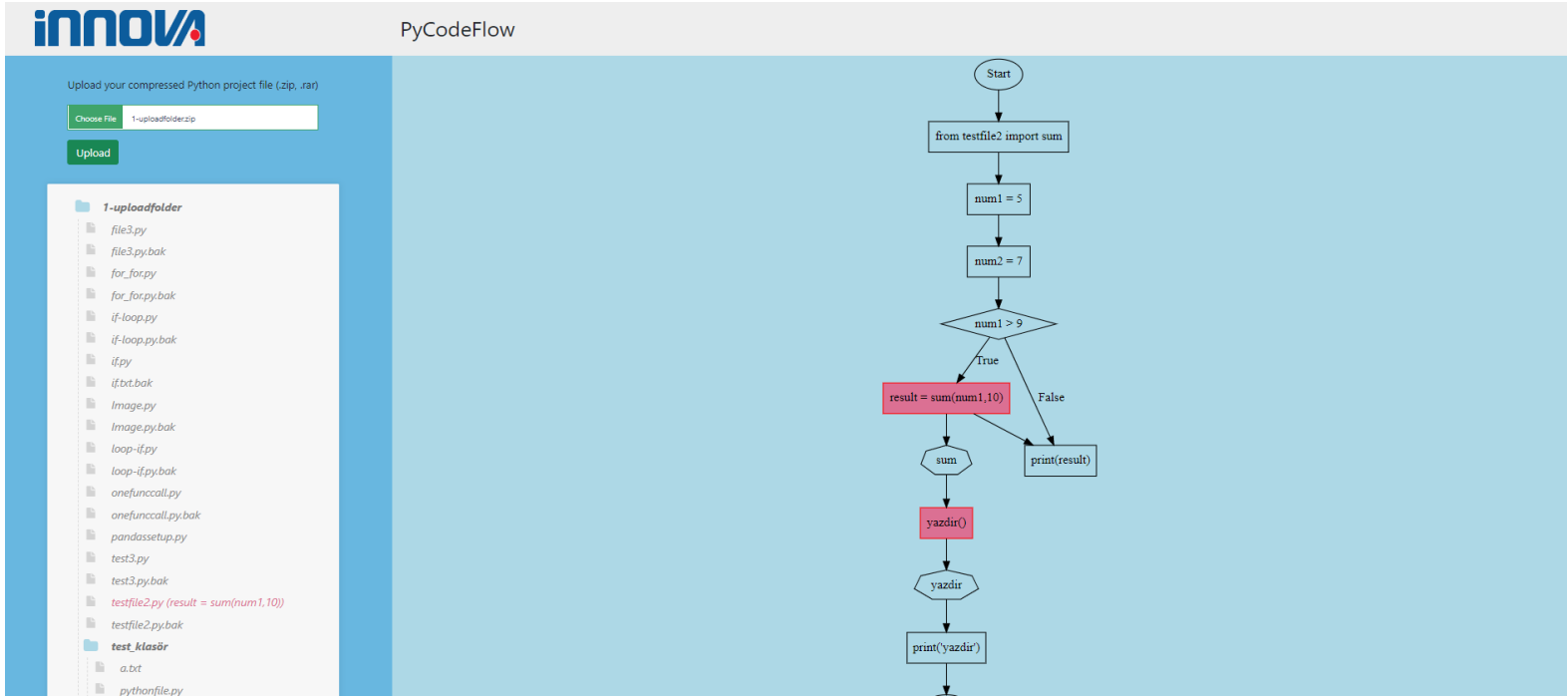
2. KULLANIM SENARYOSU

2.1. Kullanım Senaryosu Tanıtımı

Python kullanılarak yazılan kaynak kodların incelenerek anlaşılması zor bir süreçtir. Bu süreci kolaylaştırmak için kaynak kodların kod akış diyagramlarını çıkartmak amaçlanmıştır. Kullanıcı Python projesini web arayüzü aracılığı ile yükler. Proje klasör yapısı web arayüzünde görülebilir. Bu sayede kullanıcı akış diyagramını çıkarmak istediği Python kaynak kodunu seçerek akışın çıkarılmasını sağlar. Akış diyagramı web arayüzünde kullanıcıya interaktif bir şekilde sunulur. Projenin en önemli kullanımı ise, aynı proje içerisindeki başka bir kaynak kodun içe aktarıldığı senaryodur. Bu durumda ana akış diyagramı dallanarak içe aktarılmış fonksiyon akışı da ana akışa eklenmektedir. Bu sayede proje akışı da görülebilmektedir.



Şekil 1 Arayüz Örnek Görüntüsü



Şekil 2 Arayüz Örnek Görüntüsü

2.2. Gereksinimler

Tablo 1 Gereksinim Tablosu

Gereksinim ID	Değerlendirme senaryosu	Description
A.1	PyCodeFlow_Software_1	Kodun akışının bir ağaç olarak belirlenmesi
A.2	PyCodeFlow_Software_1	Kod akış ağacının SVG olarak görselleştirilmesi
A.3	PyCodeFlow_Software_1	Aynı proje içerisinde bir fonksiyon çağrıldığında akışın ana akışa eklenmesi
B.1	PyCodeFlow_Software_1	Kullanıcının Python proje dosyasını sisteme yükleyebilmesi
F.1	PyCodeFlow_Software_2	Sistemin web arayüzü olacak şekilde gösterilmesi
F.2.	PyCodeFlow_Software_2	Web arayüzündeki elementlerin gösterilmesi
F.3	PyCodeFlow_Software_2	Web arayüzündeki elementlerin interaktif olması
F.4	PyCodeFlow_Software_3	Ağaçtan tıklanan .py dosyasının diyagramının canvas üzerinde gösterilmesi.
D.1	PyCodeFlow_Software_4	Kaynak kod akış diyagramının SVG formatında veri tabanında saklanması
D.2	PyCodeFlow_Software_4	Kaynak kod akış diyagramı SVG dosyasının veri tabanından çekilmesi

2.3. Değerlendirme Senaryoları

Tablo 2 Değerlendirme Senaryoları Tablosu

Hedef	No	ID	İsim	Açıklama	Risk
Yazılım	PyCodeFlow_Software_1	1	Akış diyagramı algoritmasının çalışması	Akış diyagramını çıkaran algoritmanın doğru bir şekilde çalışması	Algoritmanın çalışması kod akışının çıkarılmasını sağlar. Akışın doğru çıkmadığı durumda kullanıcıya yanlış sonuç verilir.
Yazılım	PyCodeFlow_Software_2	2	Arayüzün interaktifliği	Uygulama arayüzünün interaktif olması	Arayüz interaktif elementlerinin doğru çalışması ile akışı çıkartılacak proje uygulamaya yüklenir. Ayrıca akış üzerinde interaktif bir şekilde akışdan ekstra bilgiler elde edilir. Bu durumun sağlanamaması ile akış diyagramı oluşturulamaz.
Yazılım	PyCodeFlow_Software_3	3	Frontend - Backend İletişimi	Uygulama arayüzündeki elementler ile backend fonksiyonlarının çalıştırılması	Arayüzdeki elementler ile akış diyagramı çıkartılacak olan kodlar backend'e gönderilir. Oluşturulan akış SVG formatında frontend'e gönderilir. Bu durumun sağlanamaması ile ya akış diyagramı oluşturulamaz ya da oluşturulan akış kullanıcıya sunulamaz.
Yazılım	PyCodeFlow_Software_4	4	Veritabanı Fonksiyonelliği	SVG formatındaki akış diyagramının veri tabanında saklanması ve kullanılması	Veri tabanında SVG formatındaki akışın saklanması ile kullanıcının işlem tekrarının önüne geçilmektedir. Bunun sağlanamaması ile uygulamanın verimliliği düşmektedir.

2.4. Test Durumları / Senaryoları

Tablo 3 Test Durumları/Senaryoları Tablosu

Senaryo	**Ön koşullar**	**Girdi koşulları/adımları**	**Beklenen sonuçlar**
1 (Algoritma)	Algoritma yazılımının tamamlanmış olması	Algoritma girdi-çıktılarının kontrol edilmesi	Algoritmanın doğru/tutarlı sonuçlar vermesi
2 (Arayüz)	Web uygulamasının ayağa kaldırılması	Elemanların sayfada bulunduğu, etkileşime geçilebilir olduğu ve fonksiyonel olduklarının test edilmesi	Arayüz elemanlarının çalışır durumda olduğunun görülmesi
3 (Front-Back İletişim)	Frontend ve backend yazılımların tamamlanmış olması	Bağlantı noktalarının çalıştırılması	Frontend ve backendin sorunsuz çalıştığının görülmesi
4 (Veritabanı)	Web uygulamasının çalışır durumda olması	Veritabanı girdi-çıktı işlemleri koşulup veritabanına etkileri izlenir	Veritabanı içerisinde bulunan verilerin doğru bir şekilde uygulamaya yansıtılabilmesi

2.5. Değerlendirme Kriterleri

I. **Algoritma:** Kod kapsamının ve birim testlerinin yapılarak algoritmanın düzgün çalışabileceğinin doğrulanması.
II. **Arayüz:** Web uygulaması üzerinde bulunan elementlerin fonksiyonelliğinin çalışabilir olması.
III. **Front-Back İletişim:** Frontend ve backend'in yazılımlarının birbiri ile bağlantısının sağlanıp sorunsuz olarak çalışabileceğinin test edilmesi.
IV. **Veritabanı:** Web uygulamasından alınan verilerin veritabanına doğru bir şekilde işlenip daha sonra kullanıcı, istediği zaman uygulamaya doğru bir şekilde yansıtabilmesi.

3. Test Aktiviteleri Faaliyetleri

3.1. Test Planlama

3.1.1. Hedeflenen Test Seviyeleri ve Çeşitleri

3.1.1.1. Test Seviyeleri

Çalışmada uygulanacak test seviyeleri; birim testleri, entegrasyon testleri, arayüz testleri ve sistem testlerini kapsamaktadır. Kabul testleri, müşteri ve kullanıcı katılımını yoğunluklu olarak gerektireceği için bu proje kapsamının dışında kalacaktır.

3.1.1.1.1. Birim Testleri

Projenin kaynak kodundaki kilit sorumluluklara sahip olan birimlerin testleri yapılacaktır. Birim testleri uygulanacak fonksiyonlar temel olarak **Visit_Flow** sınıfının özellikleri olacaktır. Aynı zamanda kod içerisindeki diğer önemli görülen birim fonksiyonlar test edilecektir.

Birim olarak test edilecek fonksiyonlar aşağıda listelenmiştir.

Tablo 4 Birim Fonksiyonlarını İçeren Tablo

def __init__( )	Visit_Flow.py
def name( )	Visit_Flow.py
def import_stmt( )	Visit_Flow.py
def funcdef( )	Visit_Flow.py
def classdef( )	Visit_Flow.py
def stmt_pop( )	Visit_Flow.py
def loop_pop( )	Visit_Flow.py
def find_loop_node( )	Visit_Flow.py
def stmts( )	Visit_Flow.py
def loop_stmt( )	Visit_Flow.py

def suite( )	Visit_Flow.py
def elif_( )	Visit_Flow.py
def if_stmt( )	Visit_Flow.py
def check_subgraph( )	Visit_Flow.py
def create_svg( )	main.py
def make_tree( )	main.py
def allowed_file( )	main.py
def upload_file( )	main.py
def show_svg( )	main.py

3.1.1.1.2. Arayüz Testleri

Arayüz testleri, web sayfasının fonksiyonelliğini test etmek amacı ile yapılacaktır. Test süreci selenium araç setleri ile yürütülecektir. Arayüz testi, web sayfasının bütün fonksiyonalitesini denetleyebilecektir.

Arayüz testi otomasyonu, regresyon setine uygun senaryolar içerecektir.

3.1.1.1.3. Entegrasyon Testleri

Entegrasyon testi gerçekleştireceğimiz modül bağlantıları; temel sınıf olan Visit_Flow sınıfı ile main.py dosyası içerisindeki fonksiyonların bağlantısını ve web bağlantılarını kapsayacaktır.

Visit_Flow sınıfına ait; yapıcı fonksiyon, visit_topdown ve export_svg fonksiyonları main.py ile birlikte kullanıldığı için bağlantı noktası olarak bu noktalar alınacaktır.

3.1.1.1.4. Sistem Testi

Yapılacak olan sistem test; uygulamanın sağlaması beklenen fonksiyonel ve fonksiyonel olmayan özelliklerin, otomatik ve manuel yöntemler ile uçtan uca test edilip raporlanmasını içerecektir. Yazılımın akışı ve web sistemlerinin düzgün çalışması hakkında raporlama yapılacaktır.

Hazırlanacak olan regresyon seti baz alınarak yazılacak olan arayüz otomasyonunun koşumu ile birlikte gereksinimlerin manuel olarak uçtan uca bir şekilde testi yapılacak ve raporlanacaktır.

3.1.1.2. Test Çeşitleri

3.1.1.2.1. Fonksiyonel Testler

Sistemin fonksiyonel olarak sağlaması gereken özelliklerin testleri yapılacaktır. Fonksiyonel test adımlarından **3.1.1.1. Test Seviyeleri** başlığı altında bahsedilen yöntemler ile testler gerçekleştirilecektir.

3.1.1.2.2. Fonksiyonel Olmayan Testler

Projenin fonksiyonel olmayan testleri, yazılımın çalışma performansından ve bu yazılımın kullanılabilirliğinden oluşmaktadır. Bu aşamaların testleri gerçekleştirilecektir.

3.1.1.2.3. Onaylama ve Regresyon Testleri

Test edilecek uygulama için bir regresyon seti oluşturulacaktır. Kapsayıcı bir şekilde oluşturulacak olan regresyon seti, yazılımsal değişikliklerden sonra sıkıntılı bir geliştirme yapılmamış olduğunun sağlamasını yapacaktır. Aynı zamanda regresyon seti içerisindeki ilgili maddeler ile onaylama testleri de yapılabilecek durumda olacaktır.

3.1.2. Hedeflenen Statik Testler ve Gözden Geçirme Planı

3.1.2.1 Hedeflenen Statik Testler

Projenin testi aşamasında uygulanacak olan statik testler hataların maliyet bakımından daha düşük olumsuzluklara sebep olması amacını taşımaktadır. Uygulamaların ve kodların çalıştırılmadan test edilebilmesi sayesinde ürünün canlıya çıkmadan önce minimum risk ihtimaline sahip olması istenmektedir. Projenin statik analizi aşamasında sırasıyla yazılım gereksinimleri dokümanının *Word* uygulaması kullanılarak dil bilgisi ve yazım kuralları açısından test edilmesi ve *Python, Javascript, HTML, CSS* programlama dillerinden oluşan kaynak kod üzerinde *sonarcloud, pylint* gibi statik kod analizi gerçekleştirme araçları kullanılarak statik analiz yapılması planlanmıştır.

3.1.2.2 Hedeflenen Gözden Geçirme Planı

Yazılım veya diğer ürün değerlerinin manuel olarak incelenmesine gözden geçirme denmektedir. Gözden geçirme sayesinde proje ürünleri içerisindeki hatalar erken aşamada saptanır. Bu projede ilk olarak proje kaynak kodlarına teknik gözden geçirilmesi uygulanacaktır. Teknik gözden geçirme esnasında ekip, yazılım geliştiricisi ile birlikte çalışacaktır. Bunun sonucunda potansiyel hata kayıtları ve gözden geçirme raporları oluşturulacaktır. Ayrıca web sayfasının istenilen tasarımda ve yapıda olup olmadığı gayri resmi gözden geçirilecektir. Son olarak yazılım gereksinim dokümanı resmi olarak gözden geçirilecektir. Dokümana teknik gözden geçirme yöntemi uygulanacaktır. Yöntem sonucunda potansiyel hata kayıtları ve gözden geçirme raporu oluşturulması beklenmemektedir.

4. Statik Testler ve Gözden Geçirme Faaliyetleri

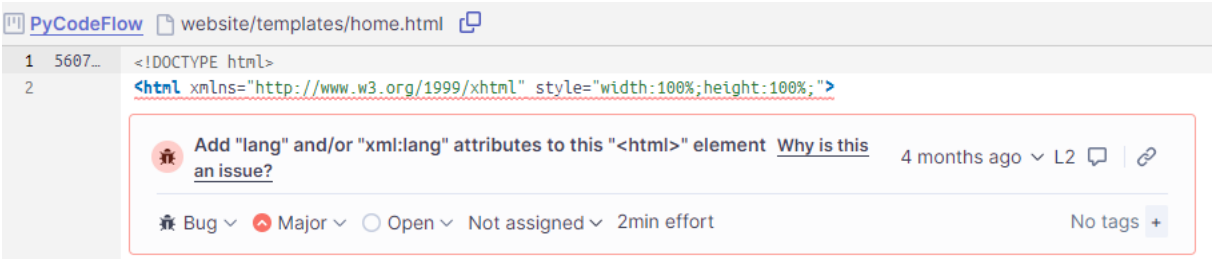
4.1. Statik Testler

Yazılım gereksinimleri dokümanı üzerinde Word uygulaması kullanılarak dil bilgisi ve yazım kuralları açısından statik analiz yapılmıştır. Bu analiz sonucunda doküman içerisinde karma dil kullanımından kaynaklı bazı dil bilgisi ve birkaç adet de noktalama işareti hatası tespit edilmiştir. Dokümanın genel olarak tablolardan oluşması ve kısa metinler içermesi dolayısıyla az sayıda hata bulunmuş, fakat dokümanın erişilebilirlik kısıtlarından dolayı bu şekilde rapor edilmiştir.

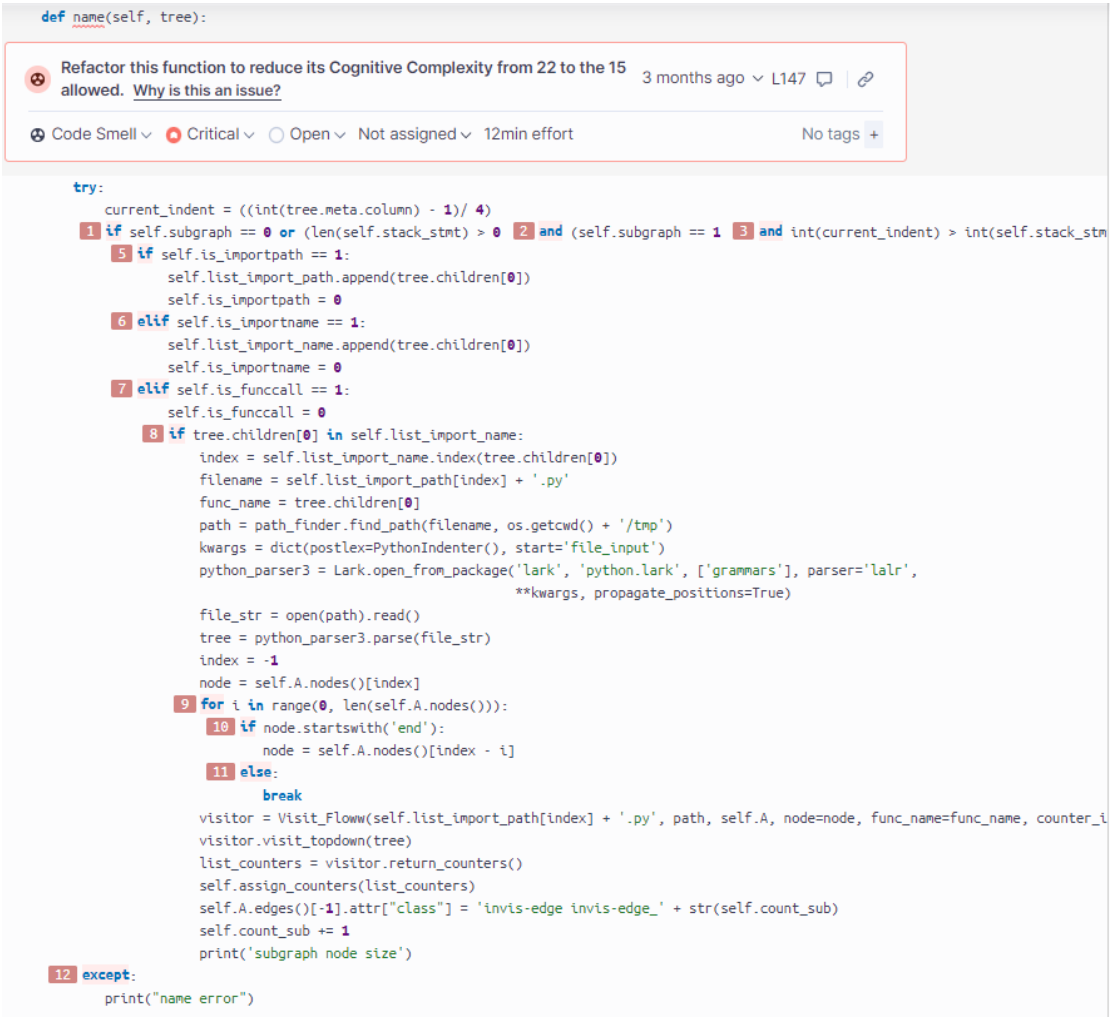
Proje kaynak kodunun, planlandığı üzere sonarcloud ve pylint yazılım testi araçları kullanılarak statik test analizi yapılmıştır. Bu analizlerin sonucunda araçlar tarafından güvenilirlik, kod karmaşıklığı, güvenlik gibi kategorilerde bazı problem oluşturabilecek hatalar ve uyarılar tespit edilmiştir. Karşılaşılan hata ve uyarılar kategorileştirilmiş ve aşağıdaki gibi raporlanmıştır. Ayrıca oluşturulan bu rapor yazılım geliştiricisi ile de paylaşılmıştır.

Tablo 5 Sonarcloud Aracı Sonuçlar Tablosu

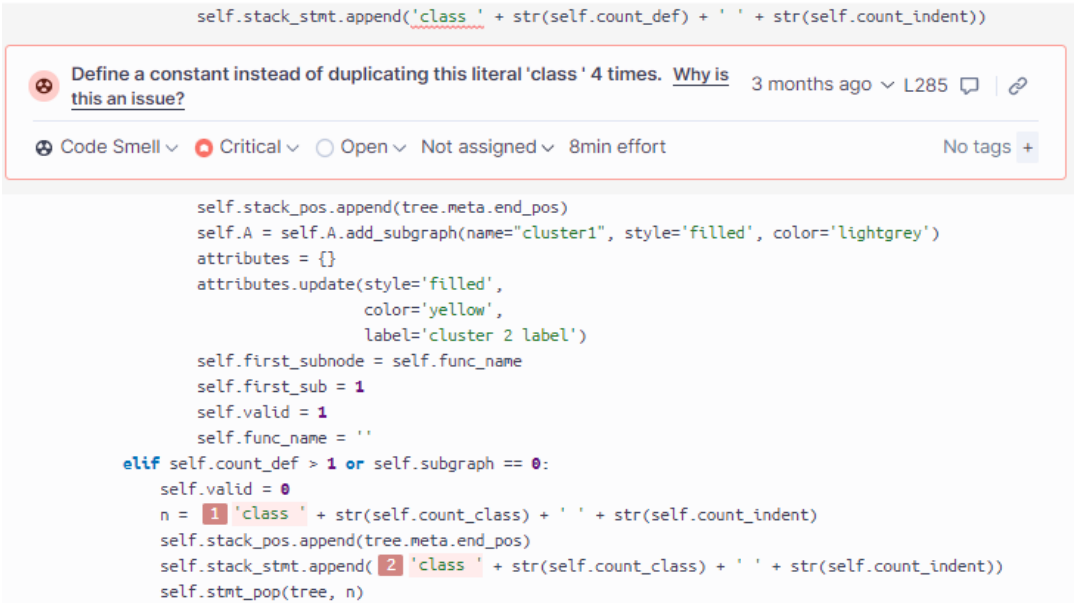
Sonarcloud			
Güvenilirlik	Önemli	Bug - html tagi içerisine xml:lang özelliğinin tanımlanmaması	Şekil_R_1
Sürdürülebilirlik	Kritik	İç içe if koşullarının karmaşıklık yaratması	Şekil_M_1
Sürdürülebilirlik	Kritik	Değişmezlerin tekrarlanması, tekrarlamak yerine değişken tanımlayarak kullanma	Şekil_M_2
Sürdürülebilirlik	Önemli	If kontrolünün kapatılmadan yeni bir if kontrolünün açılması	Şekil_M_3
Sürdürülebilirlik	Önemli	Gereksiz parantez kullanımı	Şekil_M_4
Sürdürülebilirlik	Önemli	Gereksiz yorum satırlarının bulunması	Şekil_M_5
Sürdürülebilirlik	Normal	Kullanılmayan yerel değişken tanımlanması	Şekil_M_6
Güvenlik	Önemli	Bir URL ile onu çalıştıran fonksiyon arasındaki ilişki tanımlanırken yetkili HTTP yöntemlerinin açıkça tanımlanmaması	Şekil_S_1



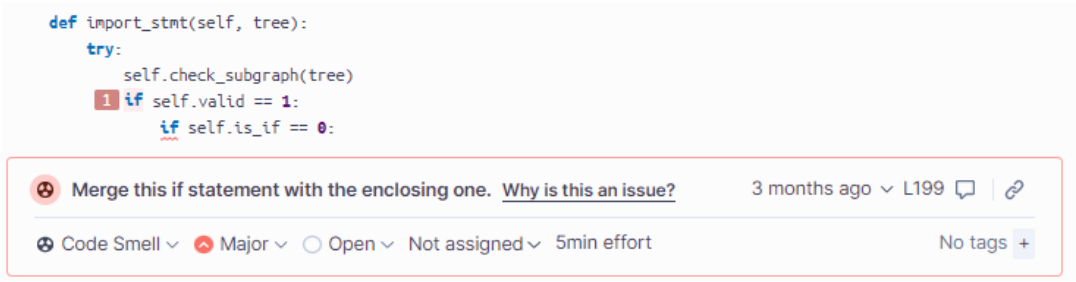
Şekil 3 Şekil_R_1



Şekil 4 Şekil_M_1



Şekil 5 Şekil_M_2



Şekil 6 Şekil_M_3

Şekil 7 Şekil_M_4

```
while len(self.stack_stmt) > c and (
        not self.stack_stmt[i].startswith('for') and not self.stack_stmt[i].startswith(
    'while')):  # or while
    i -= 1
    c += 1
    try:
        node_for = self.stack_stmt[i]
        indent_for = node_for.split()[2]
    except:
        #node_for = ''
```

Remove this commented out code. Why is this an issue? 3 months ago L800

Code Smell Major Open Not assigned 5min effort No tags +

```
            indent_for = -1
            print('invalid')
    return indent_for, node_for
```

Şekil 8 Şekil_M_5

```
                attributes.update(style='invis',
                                  color='yellow',
                                  label='cluster 2 label')
                self.A.add_node(n=self.func_name, label=text, shape='septagon')
                self.A.add_edge(self.node, self.func_name)
                self.stack_stmt.append('def ' + str(self.count_def) + ' ' + str(self.count_indent))
                self.stack_pos.append(tree.meta.end_pos)
                self.A.add_edge(self.last_main_node, self.func_name)
                func_node = self.A.get_node(self.last_main_node)
```

Remove the unused local variable "func_node". Why is this an issue? 3 months ago L248

Code Smell Minor Open Not assigned 5min effort No tags +

```
                self.first_subnode = self.func_name
```

Şekil 9 Şekil_M_6

```
@app.route('/', methods=['GET', 'POST'])
def hello():
    return app.response_class(render_template('home.html'))
```

Şekil 10 Şekil_S_1

Tablo 6 Pylint Aracı Sonuçlar Tablosu

Pylint	
Visit_Flow.py (8.02/10)	Bir işlev veya yöntemde çok fazla iç içe geçmiş blok bulunması
	Bazı değişken isimlendirmelerinin snake_case yapısına uymaması (A, n)
	Bir işlev veya yöntemde belge dizesinin bulunmaması

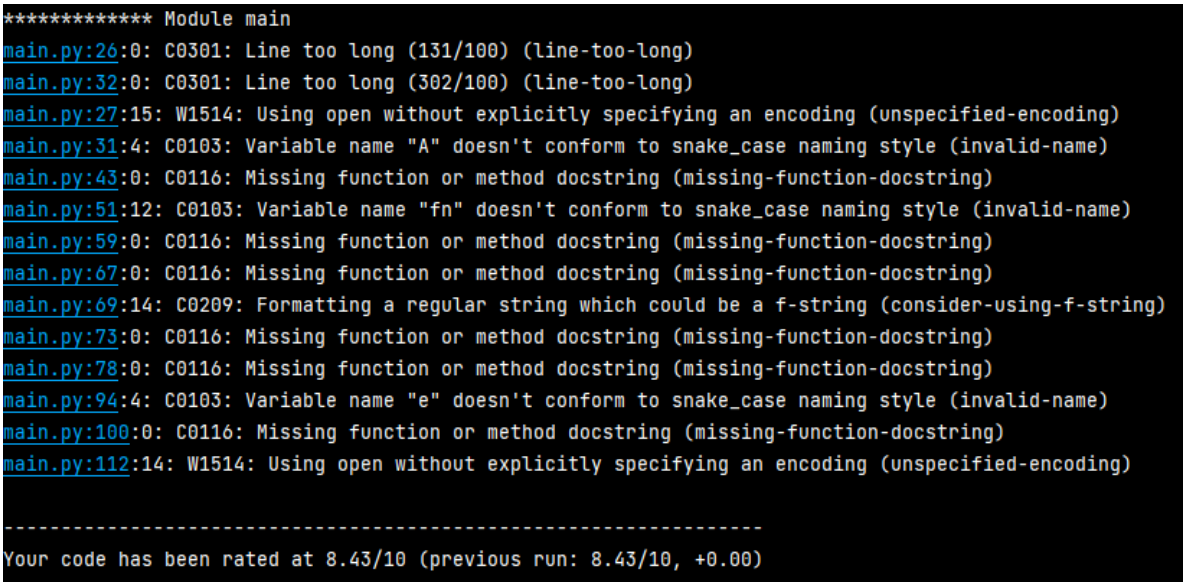
Şekil_VF	Yanlış import sıralaması
	Kullanılmayan değişken tanımlanması
main.py (7.14/10) Şekil_M	Dosyaların açılmasında kodlamanın belirtilmemesi (encoding = ‘utf-8’)
	Bazı değişken isimlendirmelerinin snake_case yapısına uymaması (A, fn)
	Bir işlev veya yöntemde belge dizesinin bulunmaması
	Gereğinden uzun kod satırlarının bulunması
path_finder.py (8.43/10) Şekil_PF	Kullanılmayan değişken tanımlanması (dir)

```
Visit_Flow.py:1026:4: C0116: Missing function or method docstring (missing-function-docstring)
Visit_Flow.py:1032:8: W0702: No exception type(s) specified (bare-except)
Visit_Flow.py:1035:4: C0116: Missing function or method docstring (missing-function-docstring)
Visit_Flow.py:1041:8: W0702: No exception type(s) specified (bare-except)
Visit_Flow.py:1044:4: C0116: Missing function or method docstring (missing-function-docstring)
Visit_Flow.py:1064:4: C0116: Missing function or method docstring (missing-function-docstring)
Visit_Flow.py:1068:8: W0702: No exception type(s) specified (bare-except)
Visit_Flow.py:1071:4: C0116: Missing function or method docstring (missing-function-docstring)
Visit_Flow.py:1075:8: W0702: No exception type(s) specified (bare-except)
Visit_Flow.py:102:8: W0201: Attribute 'lines' defined outside __init__ (attribute-defined-outside-init)
Visit_Flow.py:234:16: W0201: Attribute 'lines' defined outside __init__ (attribute-defined-outside-init)
Visit_Flow.py:278:16: W0201: Attribute 'lines' defined outside __init__ (attribute-defined-outside-init)
Visit_Flow.py:425:28: W0201: Attribute 'subgraph_node_size' defined outside __init__ (attribute-defined-outside-init)
Visit_Flow.py:9:0: R0904: Too many public methods (33/20) (too-many-public-methods)

-------------------------------------------------------------------
Your code has been rated at 8.02/10 (previous run: 8.02/10, +0.00)
```

Şekil 11 Şekil_VF (Visit_Flow.py pytlint sonucu)

```
path_finder.py:12:13: W0622: Redefining built-in 'dir' (redefined-builtin)
path_finder.py:12:13: W0612: Unused variable 'dir' (unused-variable)


-------------------------------------------------------------------
Your code has been rated at 7.14/10 (previous run: 7.14/10, +0.00)
```

Şekil 12 Şekil_M (main.py pylint sonucu)



Şekil 13 Şekil_PF (path_finder.py pylint sonucu)

4.2. Gözden Geçirme

Yazılım gereksinimleri dokümanı baz alınarak yapılan gözden geçirme testleri sonucunda, dokümanın anlaşılabilir bir dilde yazıldığı ve gereksinimlerin kategorize edilerek tablolar halinde aktarılması sonucu karmaşıklıktan uzak bir yapıda olduğu tespit edilmiştir. Bu test kapsamında doküman için herhangi bir olumsuzluğun rapor edilmesine ihtiyaç duyulmamıştır.

Tüm ekip tarafından kaynak kod üzerinde yapılan gözden geçirme testleri sonucunda riskli olabilecek ve eksik olarak görülen bölümler kaydedilmiştir. İlk yapılan gözden geçirme testi sonucunda, gözden geçirme testini finalize etmek adına, yazılım geliştiricisi ile birlikte kaydedilen kısımlar tekrardan gözden geçirilmiştir. Bu test sonucunda kaynak kod içerisinde problem yaratabilecek bölümler alt kısımda raporlanmıştır.

Tablo 7 Teknik Gözden Geçirme Sonuçlar Tablosu

Teknik Gözden Geçirme	
Visit_Flow.py	Visit_Flow.py içerisinde olması gerekenden uzun kod satırları bulunmaktadır.
Visit_Flow.py	Visit_Flow.py kaynak kodu içerisinde gereksiz yorum satırlarının olduğu görülmüştür.
Visit_Flow.py	Visit_Flow.py içerisinde yazılmış fonksiyonlarda iç içe if koşullarının karmaşıklığı fark edilmiştir.
Visit_Flow.py	Visit_Flow.py kaynak kodu içerisindeki fonksiyonların, sınıfların ve genel kod akışının tanımlarını içeren yorumların eksikliği tespit edilmiştir.
Visit_Flow.py	Visit_Flow.py kaynak kodu içerisinde yapılan değişken tanımlamaları

	karmaşık bulunmuştur.
main.py	main.py kaynak kodu içerisinde dosya açma işlemlerinin güvenilir olmayan yöntemler ile yapıldığı tespit edilmiştir.
main.py	main.py kaynak kodu içerisinde tanımlanmış fakat kullanılmamış değişkenler fark edilmiştir.
home.html	home.html dosyası içerisinde gereksiz yorum satırları olduğu görülmüştür.

5. Test Teknikleri

Projenin içeriği ve arayüzü göz önünde bulundurulduğunda test tekniği olarak Kara kutudan ziyade Beyaz kutu test teknikleri tercih edilmiştir çünkü projede girdi çıktıdan ziyade kod düzenin ve iç yapısının daha önemli olduğu ve bu nedenle de düzen, verim ve kodun işe yararlılığının test edilmesi, test sürecinin güvenliği ve doğru sonuç vermesi için önemli olduğu karar verilmiştir.

5.1 Beyaz Kutu Testleri

İlk olarak projemizin iç yapısı ve içeriği göz önünde bulundurularak ortaya çıkması mümkün hataların tahmin edilmesi ve bu kapsamda tasarlanan testlerin denenmesine dayanan **Hata Tahminleme** yapılması planlanmaktadır. Böylece yazılımı tasarlayan kişinin düşünceleri doğrultusunda test senaryolarının tasarlanması hedeflenmektedir. Ardından projede ortaya çıkabilecek kod ve arayüz bozulmaları tahmin edilerek bu kapsamda kodu ve arayüzü bozmaya yönelik yanlış sonuç vermesi beklenen **Negatif testlerin** yapılması hedeflenmektedir. Son olarak ise kodun içerisinde çokça bulunan if, else, while, switch gibi yapıların istenilen kod bölgelerinde doğru olarak çalışıp çalışmadığını kontrol etmek amacıyla **Karar Test Tasarım Tekniği** kullanılarak kodun içerisinde bulunan koşul durumlarının testinin yapılması hedeflenmektedir.

5.2 Kara Kutu Testleri

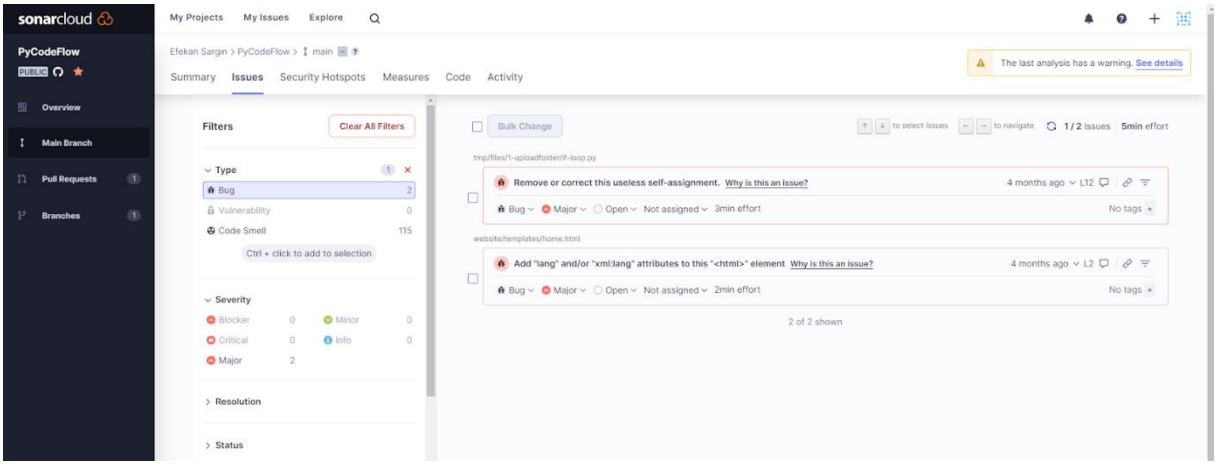
Kara kutu testinde projenin yapısı gereği herhangi bir test tekniğinden bağımsız olarak doğru girilen yazılım dosyalarının istenilen şekilde yine doğru olarak kod akış diyagramını yapısına uygun ve düzenli olarak verip vermediğini farklı kod dosya düzenleriyle test edilerek gözlemlenmesi hedeflenmektedir.

6. Test Araçları

6.1. Sonarcloud

SonarCloud, Pull Request ve Repository genelinde hataları ve güvenlik açıklarını yakalamaya yönelik çevrimiçi bir uygulamadır.

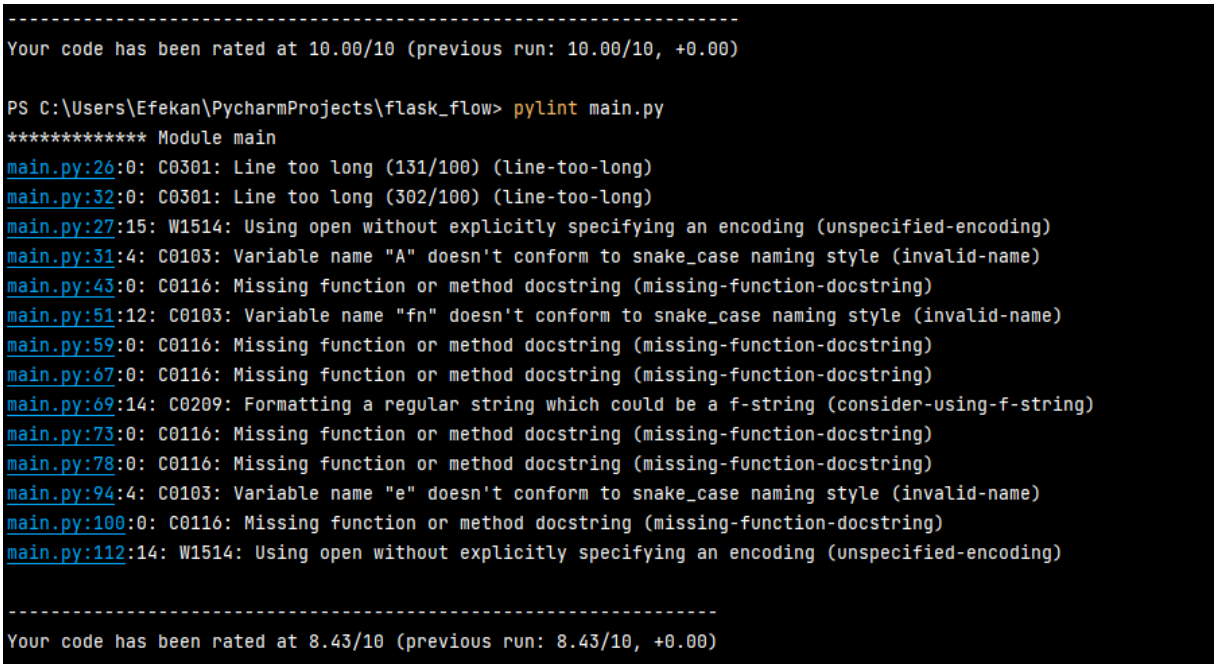
Şekil 14 Şekil_SC_1 (Sonarcloud sonucu)



Şekil 15 Şekil_SC_2 (Sonarcloud sonucu)

6.2. Pylint

Pylint, Python programlama dili için statik bir kod analiz aracıdır. Python stil kılavuzu olan PEP 8 tarafından önerilen stili takip eder.



Şekil 16 Şekil_PY (Pylint sonucu)

6.3. Arayüz Otomasyonu Araçları

6.3.1. Selenium

Selenium, tarayıcı otomasyonunu desteklemeyi amaçlayan bir dizi araç ve kitaplık için açık kaynaklı bir şemsiye projedir. Test komut dosyası dili öğrenmeye gerek kalmadan çoğu modern web tarayıcısında işlevsel testler yazmak için bir oynatma aracı sağlar.

Arayüz Otomasyonu uygulamamıza temel araç olarak selenium kullanılmaktadır. Selenium üzerine, node.js ile, inşaa edilen uygulamanın diğer bileşenleri aşağıdaki başlıklarda verilmiştir.

6.3.2. CodeceptJS

CodeceptJS, özel bir BDD tarzı sözdizimi ile modern bir uçtan uca test frameworküdür. Testler, kullanıcının bir sitedeki eyleminin doğrusal bir senaryosu olarak yazılır.
BDD, yani Behavior Driven Development, bir özelliğin davranışını düz metindeki örnekler aracılığıyla tanımlamayı içeren bir yaklaşımdır.

CodeceptJS yardımı ile Selenium üzerine koşturacağımız test senaryolarını yazmakta ve çalıştırmaktayız.

6.3.3. Allure

Allure Framework, esnek, hafif, çok dilli bir test raporu aracıdır. Periyodik koşturulan testlerden maximum yaralı bilgiyi ortaya çıkarmayı amaçlar. CodeceptJS ile uyumlu

çalışabilmektedir. CodeceptJS ile senaryolarımızı koştururken, Allure uyumlu sonuç dökümanları oluşturulmasını sağlayıp, oluşan sonuçlar ile de Allure raporunu otomatik bir şekilde, local bir web sayfası olarak, alabilmekteyiz.

7. Test Uygulamaları, Sonuçlar ve Yorumlamalar

7.1. Test Teknikleri ve Sonuçları

Bu alanda sisteme uygulanan testlerden ve sonuçları doğrultusunda alınan aksiyonlardan bahsedilmiştir.

7.1.1. Beyaz Kutu

Proje beyaz kutu test aşamasında iki adet ana beyaz kutu testimiz olmuştur bunlar;

Negatif Testler: Proje içerisinde kullanıcı tarafından bir girdi girilebilecek yerleri test etmek için negatif test metodu kullanılmıştır. Bu sayede kullanıcının yanlış girebileceği alanları istemli bir şekilde kullanıcı perspektifi göz önüne alınarak yanlış doldurulmuştur. Testler sonucunda görülmüştür ki proje sadece zip ve rar dosya türlerini işleyebilmesine rağmen girdi olarak herhangi bir dosya tipini kabul etti görülmüştür. Bu durumda bildirilmiştir ve olası çözüm olarak yanlış dosya türleri seçilmesi durumunda sistem dosya tipini almamalı ve hatta bir uyarı mesajı göstermelidir olarak geri dönüş yapılmıştır.

Karar Kapsamı: Bu test kapsamı içerisinde sistem kodu içerisinde bulunan if, else, while ve switch gibi yapıların istenilen koşulları sağlayıp sağlamadığı test edilmiştir. Bunun için çokça farklı kod yapısı sisteme test edilmesi için verilmiştir. Bu adım oldukça uzun ve önemli idi çünkü kodun içerisinde bolca bulunan bu yapıların karşılaşabileceği çeşitli kodlar için doğru ve karşılaştırılabilir düzeyde düzgün olarak ağaç yapılarını oluşturması gerekmektedir. Testler sonucunda görülmüştür ki koda verdiğimiz bütün farklı test kodu öbeklerini doğru bir şekilde ağaç yapısına dönüştürmüştür.

7.1.2. Kara Kutu

Proje kara kutu test aşamasında iki adet ana kara kutu testimiz olmuştur bunlar;

Karar Tablosu: Projenin farklı test koşullarında verdiği farklı sonuçları test etmek için kullandık. Adımlarda geçen test durumlarımızın bazılar manuel bazıları ise otomatik olarak test edilmiştir.

Tablo 8 Karar Tablosu

Koşullar	Adım 1	Adım 2	Adım 3	Adım 4
Geçerli Dosya	E	E	H	H
Dosya İçeriği	E	H	E	H
Sonuç				
Yapı oluşturulabilir	E	H	H	H

Denklik Paylarına Ayırma: Bu testte ise girdi yerimize girebilen geçerli rar ve zip dosyaları test edilmiştir ardından girdi yerine girmemesi gereken diğer dosya tipleri denenmiştir. Görülmüştür ki girdi olarak alsa bile geçersiz dosya türleriyle bir çıktı göstermemiştir. Geçersiz dosya türleri için uyarı mesajı konulması konuşulmuştur.

7.1.3. Tecrübeye Dayalı

Proje tecrübeye dayalı test aşamasında bir adet ana tecrübeye dayalı testimiz olmuştur bunlar;

Hata Tahminleme: Bu testi proje arayüzünde bulunan Upload tuşunun özelliklerini test etmek için kullandık. Bu kapsamda herhangi bir dosya seçilmeden önce tuşa bastığımızda bize boş beyaz bir kutucuk açtığını gördük. Bu işlemin kullanıcı tarafından gerçekleştiği durumlarda karışıklık olabileceği için bu durumu önlemek için sistem içerisinden arayüzüne dosya seçme alanın boş olduğuna dair bir uyarı mesajı çıkarılabileceği konusunda geri dönüşte bulunmuştur.

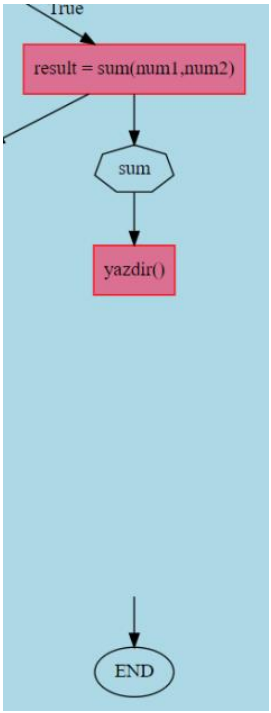
7.2. Birim Testleri Sonuçları

Proje 3.1.1.1.1 başlığı altında bulunan fonksiyonların testi gerçekleştirilmiştir. Visit_Flow.py kodunda bulunan fonksiyonlardan ‘loop_pop’; main.py kodunda bulunan fonksiyonlardan ‘make_tree’, ‘allowed_file’ için otomatik birim testi gerçekleştirilmiştir. Bu amaç doğrultusunda Python’ın unittest modülü kullanılmıştır. Yazılan otomatik birim testlerinin sonucunda tüm testler başarılı bir şekilde tamamlanmıştır. 3.1.1.1.1 başlığı altında bulunan diğer fonksiyonlara ise manuel test yapılmıştır. Yapılan testin sonuçlarında fonksiyonlar başarılı sonuçlar vermiştir.

```
def test_loop_pop(self):
    self.assertEqual(loop_pop(self, 'for 1 4', 'for', tree, 'for 1 4', 'for'), (1, 'for 1 4'))

def test_make_tree(self):
    base_path = os.getcwd()
    path = base_path + '/treefile'
    self.assertEqual(self, make_tree(path), tree)

def test_aloowed_file(self):
    self.assertTrue(allowed_file(self, 'visitflow.py'))
    self.assertFalse(allowed_file(self, 'pathfinder.py'))
```

Şekil 17 Şekil_BT (Otomatik birim testi)

Bunların yanında ön-uçta çalışan javascript fonksiyonları test edilmiştir. Svg elementleri içerisindeki node elementlerinin tıklanabilirlik scriptinde hata tespit edildi. Tıklanabilir elementlere tıklanmasıyla ana akışa eklenen yan akış diyagram düzgün bir şekilde eklenmektedir. Fakat ana akıştan çıkarılması için tekrar çıkarıldığında bazı görsel elementlerin çıkarılamadığı fark edilmiştir.



Şekil 18 Şekil_FE (Görsel hata)

7.3. Sistem ve Entegrasyon Testi Sonuçları

Genel olarak ön-uç ve arka-uç arasındaki entegrasyon test edilmiştir. Bu durumda Visit_Flow.py ve main.py kaynak kodları arasında Flask ile etkileşim sağlanmıştır. Bottom-up entegrasyon testi uygulanmıştır. Bu entegrasyon testi uygulanırken ilk olarak yapılan birim testlerinin başarılı sonuçlanması gerekmektedir. Manuel ve otomatik olarak yapmış olduğumuz birim testleri başarılı bir şekilde sonuçlandıktan sonra fonksiyonların bulunduğu kaynak kodların bütünlüğü test edilmelidir. Visit_Flow.py ve main.py kaynak kodlarına birim testleri uygulandığı için entegrasyon testi ile bu iki dosya arasındaki bütünlük test edilmiştir. Aralarındaki ilişki basit bir seviyede olduğu için test manuel olarak kolay bir şekilde yapılmıştır.

Bahsedilen entegrasyon testi başarılı bir şekilde sonuçlanmıştır. Birim testinin ardından kaynak kodda bütünlük de sağlanmıştır.

2.2. Gereksinimler, 2.3. Değerlendirme Senaryoları, 2.4. Test Durumları / Senaryoları başlıkları altındaki senaryo ve test durumlarına sistem testi uygulanmıştır. İlk olarak istenilen ister ve gereksinimlerin karşılanıp karşılanmadığı test edilmiştir. Bununla birlikte değerlendirme senaryoları kullanıcı kılavuzundan yararlanılarak test edilmiştir. Testlerin sonucunda herhangi bir hata tespit edilmemiştir. **Herhangi bir hata tespit edilemediği için bir**

tablo çıkarılması gereği duyulmamıştır. Belirtilen başlıkların altında tanımlanan veritabanına ait testler, veritabanı ait geliştirmeler devam ettiği için gerçekleştirilememiştir.

7.4. Arayüz Otomasyonu Testleri Sonuçları

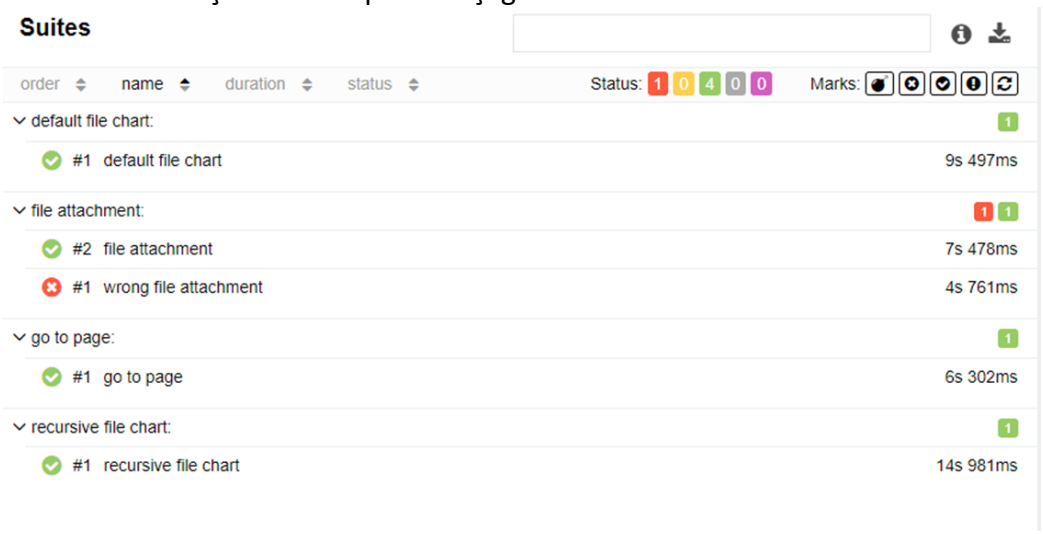
Araçlar başlığı altında bahsedilen Arayüz Otomasyonu araçları ile geliştirilen test sonrasında aldığımız çıktılar aşağıdadır.

Tanımlanan test durumları aşağıdaki tabloda verilmiştir. Test senaryoları, regresyon durumlarında koşuma uygun şekilde tasarlanmıştır.

Tablo 9 Arayüz Otomasyonu Test Sonucu Tablosu

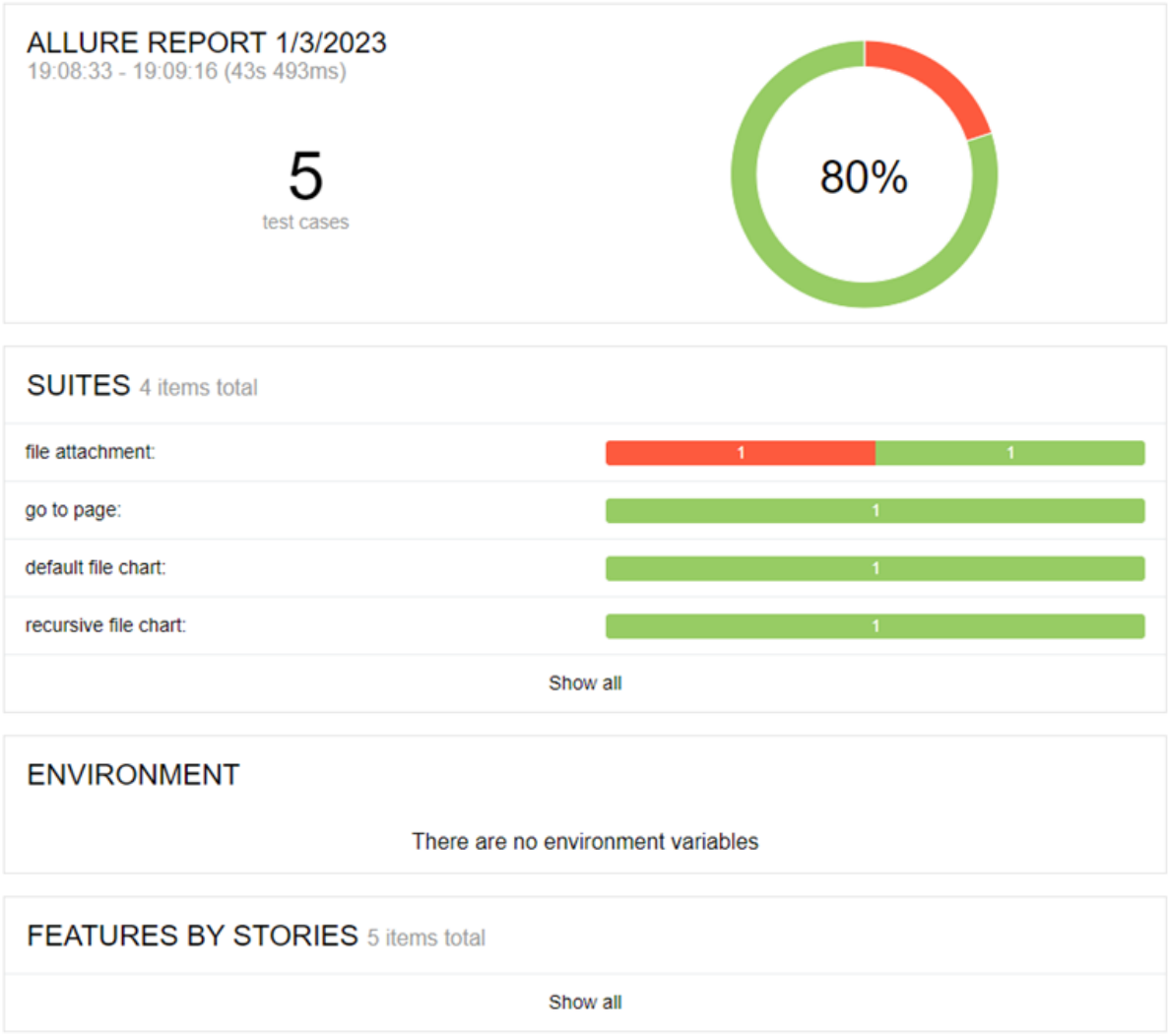
Özellik	Senaryo	Açıklama	Başarı
go to page	got to page	Web sayfasının başarılı bir şekilde açılması	+
file attachment	file attachment	Doğru dosya yüklemeye karşı sistemin davranışlarının ölçümü	+
	wrong file attachment	Yanlış dosya yüklemeye karşı sistemin davranışlarının ölçümü	-
default file chart	default file chart	Bir dosyaya tıklandığında grafiğin açılması	+
recursive file chart	recursive file chart	İçerisinde recursive yapı bulunan grafiğin açılması ve bu özelliğin kullanılması	+

Test durumları için Allure raporları aşağıdadır.



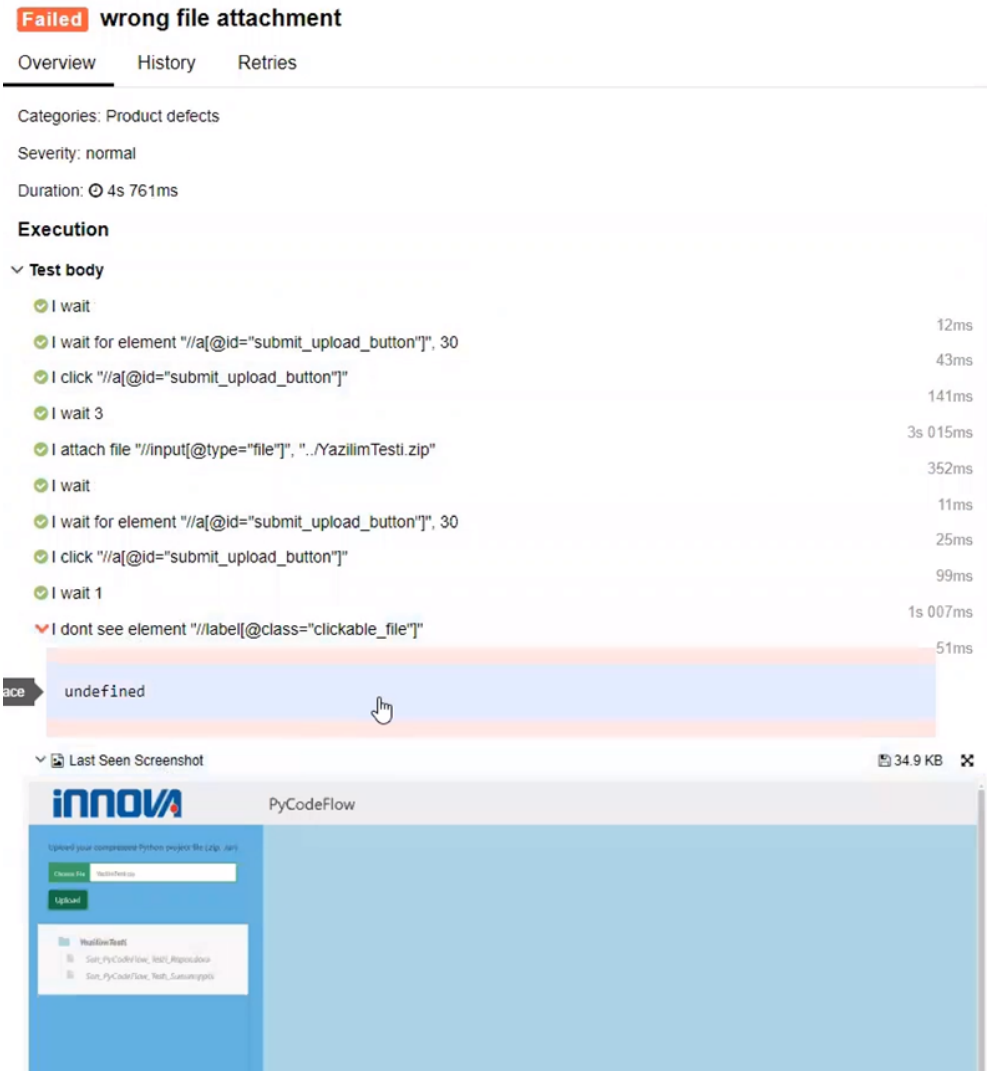
Şekil 19 Şekil_Al_1 (Allure raporu)

Hazırlanan 5 adet test duruma ait genel başarı raporu aşağıdaki gibidir.



Şekil 20 Şekil_Al_2 (Allure raporu)

Testler genel olarak başarı göstermiş olsa da **file attachment / wrong file attachment** senaryosu başarısız sonuçlanmıştır. Bu durum hakkındaki Allure raporu aşağıdadır.



Şekil 21 Şekil_Al_3 (Allure raporu)

Bu sayfada test adımlarını görebilmekteyiz. Fail durumunda sayfanın son halinin ekran görüntüsü, adımların altına eklenmektedir.

Görüldüğü üzere yanlış formatta dosya yüklemesi yapmamıza rağmen sol panelde .pptx uzantılı dosyalar görüntülenmektedir. Bu beklenmeyen bir hata durumudur.

Proje kaynak kodu ödev dosyaları arasına eklenmiştir. Rapor allureRunner.js dosyası koşturularak oluşturulmaktadır.

Şekil Listesi

Tablo Listesi