İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ
2010
FEN BİLİMLERİ ENSTİTÜSÜ

İş Akışı Programı ve Tasarımı

Yazılım Mühendisliği

Ana Bilim Dalı Dönem

Projesi

Mehmet Göktürk

ULUDAŞDEMİR

Proje Danışmanı: Dr. Öğr. Üyesi Mansur Alp Toçoğlu

Ocak 2024

İş Akışı Programı ve Tasarımı

Öz

Dökümantasyon tabanlı yürüyen iş süreçlerinin daha verimli hale getirilmesi için tasarlanması planlanan iş akış programı/arayüzü. İş akışı yönetimi onay süreçleri, izin formları gibi birtakım işlerin bilgisayar tabanlı yürütülebilir hale getirilmesini sağlar. İş akışı yönetimini kolaylaştırmak ve standart hale getirmek için iş akışı motorları kullanılmaktadır. Bu projede de SpiffWorkflow iş akışı motoru kullanılması planlanmaktadır.Bu çalışmada bir sürece ait belgelerin katılımcılar arasında dolaşımını otomatik hale getirecek bir sistem gerçeklenmesi düşünülmüştür.

Anahtar Sözcükler: İş akışı, Tasarım, Analitik yöntem, SpiffWorkflow, Dökümantasyon

WORKFLOW PROGRAM AND DESIGN

Abstract

Workflow program/interface designed to make documentation-based business processes more efficient. Workflow management enables some tasks, such as approval processes and permission forms, to be made computer-based executable. Workflow engines are used to facilitate and standardize workflow management. It is planned to use the SpiffWorkflow workflow engine in this project. In this study, it is planned to implement a system that will automate the circulation of documents belonging to a process among the participants.

Keywords: Workflow, Design, Analytical method, SpiffWorkflow, Documenta

İçindekiler

Şekiller Listesi

Bölüm 1

Giriş

Günümüzde birçok iş süreci bilgisayarlar tarafından yönetilir olmuştur. Kurumlar yapılan işteki verimliliği artırmak, zamandan ve materyalden tasarruf etmek için iş akışı yönetimi teknolojisini kullanmaktadırlar. İş Akışı Yönetimi Koalisyonu'na göre bir işin tamamlanması için geçen sürenin yaklaşık %90'lık kısmı bir yerden bir yere ulaştırma işine ayrılmakta, sadece %10'luk kısmı asıl gerçekleştirilecek faaliyete harcanmaktadır[6]. İş akışı yönetimi onay süreçleri, izin formları gibi birtakım işlerin bilgisayar tabanlı yürütülebilir hale getirilmesini sağlar.

İş akışı yönetimini kolaylaştırmak ve standart hale getirmek için iş akışı motorları kullanılmaktadır. Bu ödevde de SpiffWorkflow[2] iş akışı motoru kullanılmıştır.

Bu çalışmada bir sürece ait belgelerin katılımcılar arasında dolaşımını otomatik hale getirecek bir sistem gerçeklenmesi düşünülmüştür. Örnek olarak dönem ödevi süreci ele alınmıştır.

Dönem sonu proje ödevindeki belge süreci istek formu ve proje planı vermekle başlayıp sırasıyla ara rapor ve son rapor vermekle sonlanan bir süreçtir. Dönem projesi ödevini yapacak olan öğrenci önce sisteme kayıt olur. Daha sonra proje iş akışını başlatır. Bu aşamadan sonra yapacağı ilk iş istek formunu göndermektir. Bitirme istek formunu gönderdikten sonra belge onay bekleme durumuna geçer. Eğer öğrenciye atanan danışman belgeyi onaylarsa bir sonraki adım olan proje planı göndermeye geçilir. Eğer danışman onaylamazsa aynı iş tekrar öğrencinin karşısına çıkarılır. Diğer formlar için de benzer süreç gerçekleşir.

2.1 İş Akışının Modellenmesi[1]

İş akışı bir iş sürecinin tamamlanması için yürütülmesi gereken birbirine bağlı bir dizi adımdır.

Bir iş akışının önemli bileşenleri:

- Giriş parametrelerinden başka ek bir bağımlılığı olmayan iyi tanımlanmış bağımsız adımlar
- İş akışı içinde bir adımdan diğerine geçişi tanımlayan kurallar
- İlk adımı başlatacak girdi
- Sonuca ulaşıldığını belirten son adım

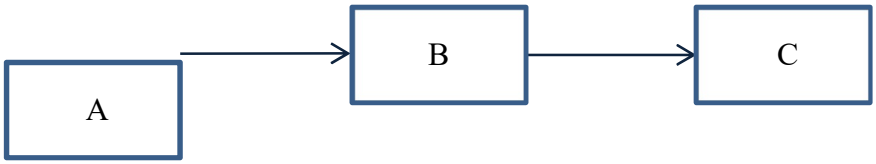
İş akışı yönetimi sayesinde

- İş sürecinin yönetimi ve kontrolü iyileştirilerek verimlilik artırılır.
- Değişen iş ihtiyaçlarına karşı daha esnek tasarım imkanı sağlar.
- Hizmet kalitesi artar.
- Bazı işlerin otomatik hale getirilmesiyle çalışan sayısı azaltılabilir.
- Bir çalışanın tüm organizasyonu öğrenmesine gerek kalmadığı için çalışanların eğitim maliyeti düşer.

İş akışlarının modellenmesinde iş akışı kalıpları kullanılır. Aşağıda sık kullanılan bazı iş akışı kalıplarının modelleri gösterilmiştir.

2.1 Sequence

İş akışının en temel kalıbıdır. Birbiri ardına gelen iş parçalarının oluşturulmasında kullanılır. Bu kalıpta, bir iş parçası kendinden öncekinin tamamlanmasından sonra etkinleştirilir. Bu kalıba bir örnek aşağıdaki gibi olabilir.

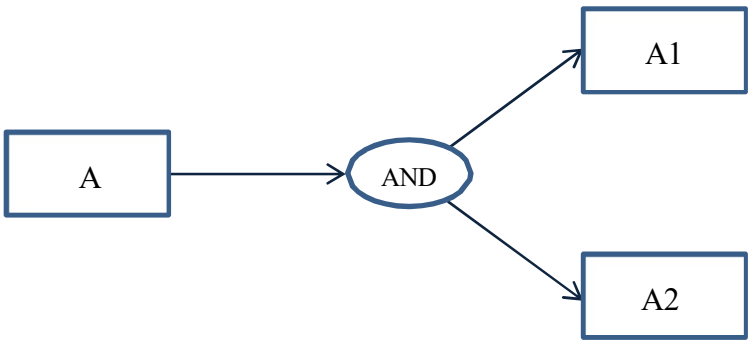


Şekil 1.1: Sequence kalıbı

Örnekteki B iş parçasının yürütülebilmesi için A iş parçasının tamamlanmış olması gerekir.

2.2 Parallel Split

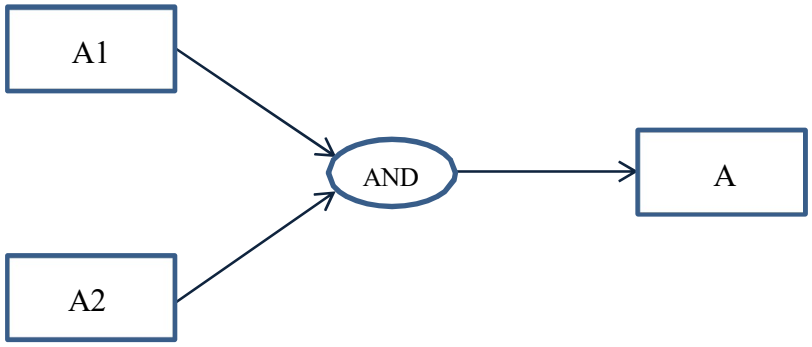
Bir kolun eş zamanlı olarak çalışan iki veya daha fazla kola ayrılmasıdır. AND-split olarak da adlandırılır. Bu kalıp aşağıdaki gibi modellenir.



Şekil 1.2: Parallel Split kalıbı

2.3 Synchronization

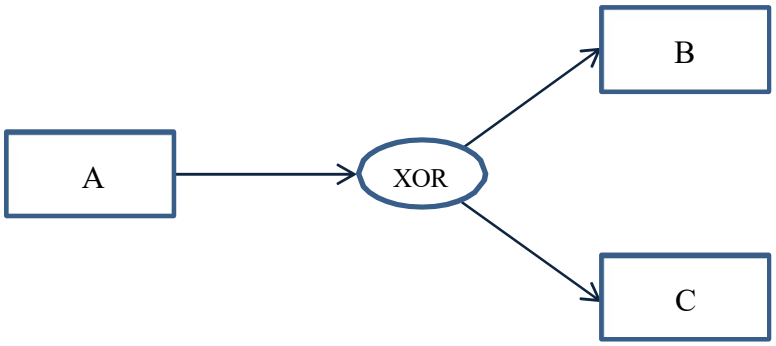
İki veya daha fazla kolun tek bir kolda birleşmesidir. AND-join olarak da adlandırılır.



Şekil 1.3: Synchronization kalıbı

2.4 Exclusive Choice

Belirtilen şarta göre çıkış kollarından bir tanesinin seçilmesidir. XOR-split olarak da adlandırılır.



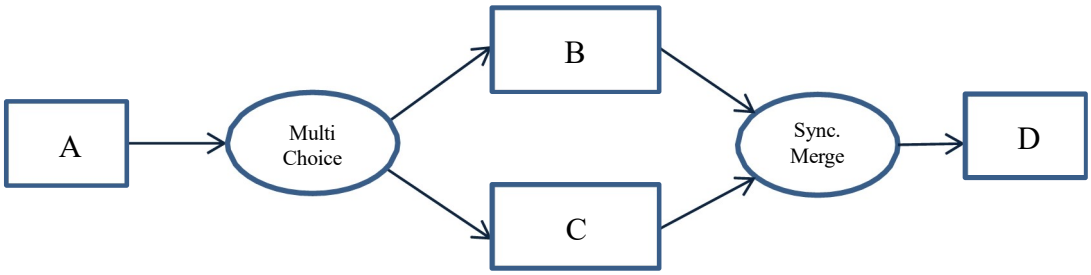
Şekil 1.4: Exclusive Choice kalıbı

2.5 Multi Choice

Belirtilen bir şarta göre bir veya daha fazla çıkış kolunun seçilmesidir. OR-split olarak da adlandırılır. Exclusive Choice kalıbında B veya C iş parçalarından sadece biri seçilebilirken, bu kalıpta B veya C iş parçalarından herhangi biri veya her ikisi seçilebilir.

2.6 Structured Synchronizing Merge

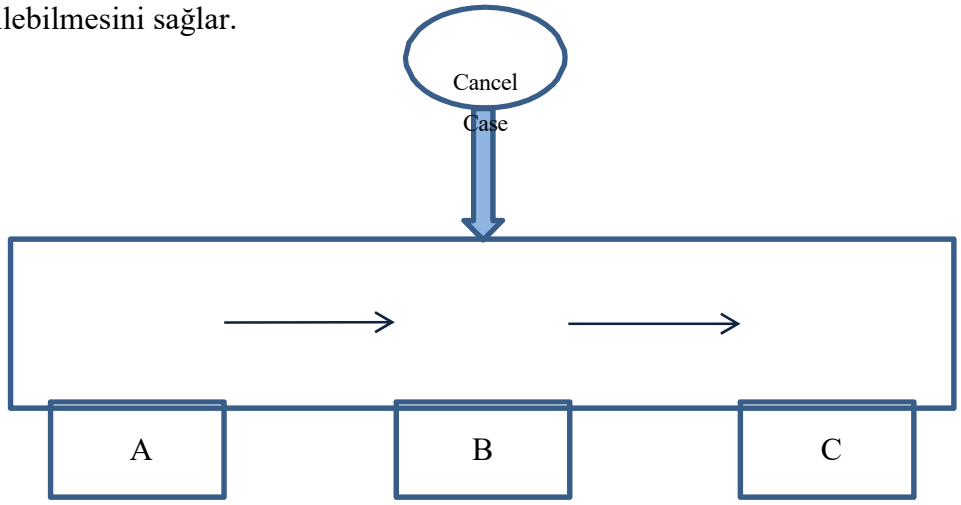
Daha önce ayrılmış iki veya daha fazla kolun tek bir kolda birleştirilmesidir. Öncesinde mutlaka tek bir multi-choice olmalıdır. Bu kalıp multi choice sırasında seçilen kolları birleştirir.



Şekil 1.5: Structured Synchronizing Merge kalıbı

2.7 Cancel Case

O anda çalışan veya gelecekte çalışma ihtimali olan iş parçaları dahil iş akışının tamamının iptal edilebilmesini sağlar.



Şekil 1.6: Cancel Case kalıbı

2.8 Cancelling Partial Join

Bu kalıpta m adet iş parçası içinden n tanesi(n < m) tamamlandığında kalan m-n tanesi

iptal edilir ve bir sonraki iş parçasına geçilir.

2.9 Persistent Trigger

Bir iş parçasının, iş akışı içindeki başka bir iş parçasından veya dış ortamdan gelen bir sinyal ile tetiklenebilmesini sağlar.

3. İş Akışı Motoru

Tanımlanan bu iş akışı modellerini çalıştırmak için iş akışı motoru kullanılır.

İş akışı motorları varlık tabanlı(entity-based) ve etkinlik tabanlı(activity-based) olarak ikiye ayrılırlar. Varlık tabanlı iş akışlarının merkezinde bir doküman ve bu dökümanın geçmesi gereken durumlar vardır. Etkinlik tabanlı iş akışlarının merkezinde ise bir işin tamamlanması için yapılması gereken bir dizi etkinlik(iş, task) vardır. Etkinlik tabanlı iş akışları varlık tabanlı olanlara göre daha karmaşık bir yapıya sahiptirler.[14]

İş akışı motoru

- Hem otomatik hem manuel adımların olduğu iş akışını depolama ve tanımlamaya imkan tanımalı
- Adımlar arasındaki durum geçişlerini anlamalı ve gerçeklemeli
- Bir adımın çıktısını alıp diğer adıma girdi olarak aktarabilmeli
- Manuel adımlar sırasında çalışmayı durdurup iş parçalarını sıraya alabilmeli
- Bir durum yönetim mekanizması sağlamalıdır

Bu projede etkinlik tabanlı SpiffWorkflow[2] iş akışı motoru kullanılacaktır.

3.1 SpiffWorkflow

SpiffWorkflow, Python ile yazılmış etkinlik tabanlı(activity-based) bir iş akışı motorudur. Bu iş akışı motoru için kullanılan kavramlar WorkflowSpec, Workflow, TaskSpec ve Task kavramlarıdır.

3.2 İş Akışının Tanımlanması

İş akışı Python kodu veya XML olarak tanımlanabilir. Bir WorkflowSpec değişik TaskSpec öğelerinin bir araya getirilmesiyle oluşur. Spec öğeleri serileştirilebilir. XML olarak oluşturulan dosyaların okunup bunlardan Python nesneleri üretilebilmesi için XmlSerializer sınıfı kullanılır.

Bu projedeki iş akışı tanımları xml olarak oluşturulmuştur. Projede xml dosyasındaki tanımdan WorkflowSpec üretmek için kullanılan fonksiyon Şekil 3.8'de verilmiştir.

```
def create_spec_from_xml(filename):
    """ create workflow spec from given xml file """

    serializer = XmlSerializer()

    with open(filename) as f:
        xml_data = f.read()

    wf_spec = serializer.deserialize_workflow_spec(xml_data, filename)

    return wf_spec
```

Şekil 1.7: WorkflowSpec oluşturma fonksiyonu

3.2.1 Task Yazma

SpiffWorkflow tarafından çalıştırılabilecek bir iş akışı tanımı *Start* taskı ile başlayıp *End* taskı ile bitmek zorundadır. *Start* taskı özel olarak *start-task* düğümü ile belirtilir. *End* taskı da herhangi bir taska *successor* düğümü olarak bağlanır. Sequence kalıbına uygun basit bir iş akışı Şekil 3.9'da gösterildiği gibi yazılabilir.

```
<process-definition name="dönem projesi" revision="1.2">

    <start-task>
        <successor>task_a</successor>
    </start-task>

    <task name="task_a">
        <successor>task_b</successor>
    </task>

    <task name="task_b">
        <successor>end</successor>
    </task>

</process-definition>
```

Şekil 1.8: Sequence kalıbı için xml örneği

İstenirse *description* etiketi içine açıklama yazılabilir. *successor* etiketi ilgili iş parçasından sonra gelecek olan iş parçasını belirtmek için kullanılır. Eğer bir *Exclusive Choice* yazılıyorsa *default-successor* ve *conditional-successor* etiketleri kullanılır. Ayrıca task ismi de *name* niteliği ile belirtilmek zorundadır.

3.2.2 Verilerin Yazımı

İş parçalarına ait veriler *data*, *pre-assign, post-assign, define* etiketleri kullanılarak yazılabilir. *data* etiketi kullanılarak yazılan veriler sadece o iş parçasına görünürdür fakat *pre-assign* veya *post-assign* kullanılarak yazılanlar diğer iş parçaları tarafından da görülebilir. *define* etiketi ile tanımlanan veriler daha sonra değiştirilemez.

```
<task name="task_a">
    <data name="test-data" value="1" />
    <successor>task_b</successor>
</task>
```

Şekil 1.9: data etiketi kullanımı

Şekil 3.10'da gösterilen örnekte *task_a* iş parçasının *test-data* alanına "1" değeri atanmıştır. Bu veriye Python kodu içinden *task_a* iş parçasının *task* değişkenine atandığı düşünülürse

```
task.get_spec_data("test-data")
```

şeklinde ulaşılabilir.

3.2.3 pre/post-assign Değişkenleri

pre-assign ve *post-assign* değişkenleri iş akışındaki tüm iş parçaları tarafından görülebilir. Yazım şekli *data* etiketinde olduğu gibidir. Şekil 3.11'de bir *Exclusive Choice* taskı ve *pre-assign* değişkeninin nasıl yazılıp kullanıldığı gösterilmiştir.

```
<exclusive-choice name="excl_choice">
    <description>Branch one of the tasks</description>
    <pre-assign name="test" value="1" />

    <default-successor>task_a</default-successor>

    <conditional-successor>
        <equals left-field="test" right-value="2" />
        <successor>task_b</successor>
    </conditional-successor>
</exclusive-choice>
```

Şekil 1.10: pre-assign etiketi kullanımı

Varsayılan durumda *test* değişkeni "2" değerine eşit olmadığı için task_a iş parçası çalıştırılacaktır. Eğer *test* değişkeninin değeri aşağıdaki gibi "2" olarak ayarlanırsa *task_b* iş parçası çalıştırılır.

```
task.set_data(**{task.task_spec.pre_assign[0].left_attribute:  "2"})
```

equals etiketi içinde *right-value* yerine *right-field* yazılıp *test* değişkeni doğrudan bir değer yerine başka bir değişkenin değeriyle de karşılaştırılabilir.

3.3 İş Akışının Çalıştırılması

İş akışının çalıştırılabilmesi için önce xml dosyasından WorkflowSpec nesnesinin oluşturulması gerekir. Oluşturulan bu nesne Workflow sınıfına parametre olarak aktarılarak bir iş akışı nesnesi yaratılır. Daha sonra bu nesne üzerinde *complete_task_from_id* veya *complete_next* metodları çağrılarak ilgili iş parçaları tamamlanır. Aşağıda örnek bir kullanım verilmiştir.

```
wf_spec = create_spec_from_xml("test.xml")
workflow = Workflow(wf_spec)

# ilk hazır taskı al
task = workflow.get_tasks(state=Task.READY)[0]

workflow.complete_task_from_id(task.id)
#workflow.complete_next()
```

Şekil 1.11: İş akışı örneğinin oluşturulması ve task tamamlama

3.4 PDF Form Verisinin Alınması

PDF formdaki alanlara girilen verileri almak aşağıdaki adımlardan oluşur:

- PDF formun tasarlanması aşamasında bölüm 3.4.4.'te anlatıldığı gibi verinin FDF formatında gönderilmesi seçilir.
- Formun gönderildiği url adresinin görünüm fonksiyonunda fdf verisi *request.data* ile alınır.
- Herhangi bir alanın değerine ulaşmak için form verisinin bulunduğu bir sözlük döndüren *get_form_fields* fonksiyonu kullanılır.

get_form_fields fonksiyonu içinde fdf formatındaki veri aşağıdaki düzenli ifade ile ayrıştırılır:

```
<<          # verinin başlangıcı
/T          # alan adı tanımlayıcısı
\((\w+)\)   # alan adı
/V          # alan değeri tanımlayıcısı
[\(|/]      # değer tanımlayıcısından sonra ( veya / karakteri
(.+?)       # alan değeri
\)?         # 0 veya 1 adet sağ parantez
>>          # verinin sonu
```

Fonksiyon `get_form_fields(fdf_string)["alan adı"]` şeklinde kullanılır.

3.5 pdftk Kullanımı

pdftk(pdf toolkit)[5] pdf formatındaki belgeler üzerinde birleştirme, belirli sayfaları ayırma, sayfaları döndürme, arka plana gömülü resim ekleme, fdf verisi ile pdf formu doldurma ve formu tekrar düzenlenemez hale getirme gibi birçok işlem yapılmasını sağlayan bir araçtır.

Bu projede pdftk, pdf formunu fdf verisiyle doldurup tekrar düzenlenemez halde bir pdf belgesi üretmek için aşağıdaki formatta kullanılmıştır:

```
pdftk <pdf form> fill_form <fdf dosyası> output <çıktı dosyası adı> flatten
```

3.6 Python Web Çatıları(Framework)[10]

Web frameworkler geliştiricilerin web tabanlı uygulamalar yazarken işini kolaylaştıran kütüphanelerdir. Bu web çatıları sayesinde geliştiriciler protokoller, soketler gibi düşük seviyeli ayrıntılarla uğraşmak zorunda kalmazlar.

Web çatıları, web uygulamalarını çalıştırabilmek için gerekli olan genel işlemleri doğrudan kendileri veya eklentileri aracılığıyla sağlarlar[11]. Yaygın olarak gereken işlemler şunlardır:

- URL yönlendirme
- HTML, XML, JSON gibi çıktı format şablonlarını oluşturma
- Veritabanı işlemleri
- CSRF saldırılarına karşı güvenlik sağlama

Web çatıları sağladıkları işlevselliğe göre "full stack" ve "micro" olarak ikiye ayrılırlar.

3.7 Full Stack Framework

"Full Stack Framework" olarak adlandırılan grup bir web uygulaması için gerekebilecek hemen her şeyi bünyesinde barındırır. Fakat sağlananların dışında bir işlevsellik gerekirse bunu eklenti halinde dahil etmek daha zordur. Bazı örnekler şunlardır:

- Django
- Pyramid
- Zope2
- web2py

3.8 Micro Framework

"Micro Framework" olarak adlandırılan grup yukarıdaki genel işlevsellikleri bünyesinde barındırarak en düşük seviyede işlevsellik sağlar fakat gerekli ek işlevsellikler eklentiler yardımıyla kolayca dahil edilebilir. Bazı örnekler şunlardır:

- Bottle
- Flask
- CherryPy
- Bobo

3.9 Basit Proje Dizini Yapısı

Flask uygulamalarında genel kabul olarak html dosyaları “templates”, css, js veya resim dosyaları “static” adlı alt klasörler içinde tutulur. Uygulamanın büyüklüğüne göre modüller ayrı paketler halinde düzenlenebilir. Basit dizin ağacı şu şekildedir:

```
/app
 application.py
 /static
    style.css
 /templates
    layout.html
    index.html
```

3.10 Flask Application Nesnesi

Flask sınıfının bir örneğidir. Uygulamanın temel nesnesidir. Parametre olarak uygulama modülünün veya paketinin ismini alır. Tek bir modül kullanılıyorsa parametre olarak “_name_” verilmelidir.

3.11 Şablonlar

Şablonlar uygulamanın arayüzünü oluşturan *.html* uzantılı dosyalardır. Flask uygulamalarında genel kabul olarak html dosyaları "templates" klasörü altında tutulur.

3.12 Jinja Template Engine

Flask uygulamaları varsayılan şablon motoru(template engine) olarak Jinja2 kullanırlar. Jinja2[8], Python için yazılmış bir şablon motorudur. Kullandığı sentaks Django sentaksından esinlenerek oluşturulmuştur.

Bir Jinja2 şablonu html kodları dışında değişkenler ve ifadeler de içerebilir. Değişkenler {{}} ifadesi içine konulur.

```
<p class="error">{{ error }}</p>
```

3.13 Şablon Kalıtımı (Template Inheritance)

Şablon kalıtımı özelliği tüm sayfalardaki genel öğeleri ve çocuk şablonların üzerine yazabileceği "block" öğelerini içeren temel bir şablon oluşturmayı sağlar. Şekil 3.22 ve Şekil 3.23'te gösterilen örnek durumu daha iyi özetler.

layout.html temel şablonu belirtir.

```
<!doctype html>
<html>
    <head>
        <title>{% block title %} {% endblock %}</title>
    </head>

    <body>
        {% block content %} {% endblock %}
    </body>
</html>
```

Şekil 1.12: Layout sayfası

index.html, *layout.html* içeriğini aynen kalıtır ve onun tarafından sağlanan *block* alanlarını kendine göre düzenler.

```
{% extends "layout.html" %}

{% block title %} Index Page {% endblock %}

{% block content %}
    <h1>Home page of Flask application</h1>
{% endblock %}
```

Şekil 1.13: Layout sayfasından türetilen anasayfa

3.14 Template Rendering

HTML kodunu render etmek için *render_template* fonksiyonu kullanılır. Fonksiyona parametre olarak render edilmek istenen html dosyasının ismi verilir. Template içindeki değişkenlere gönderilmek üzere ek parametre de alabilir. Örnek kod parçası Şekil 3.24'te gösterilmiştir.

```
from flask import render_template

@app.route('/hello')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Şekil 1.14: render_template fonksiyonu kullanımı

3.15 redirect ve url_for Fonksiyonları

redirect fonksiyonu kullanıcıyı farklı bir adrese yönlendirir. *url_for* fonksiyonu da belirli bir fonksiyona ait URL üretmek için kullanılır. İlk parametre olarak fonksiyon adını alır. Ayrıca URL içindeki değişken kısımlar için parametre verilebilir. Belirsiz değişken isimleri URL adresine sorgu terimi olarak eklenir.

```
from flask import redirect, url_for

@app.route('/')
def index():
    return redirect(url_for('login'))
```

Şekil 1.15: redirect ve url_for fonksiyonu kullanımı

Şekil 3.25'teki kod parçası kullanıcıyı index sayfasına gitmek istediğinde login fonksiyonu için üretilen URL adresine yönlendirir.

```
@app.route('/user/<username>')
def profile(username):
    return 'User %s' % username
```

Şekil 1.16: url_for kullanımı için örnek görünüm fonksiyonu

Şekil 3.26'da gösterilen *profile* görünüm fonksiyonu için *url_for* fonksiyonu

```
url_for('profile', username='Foo Bar')
```

şeklinde çağrılabilir.

3.16 Request Nesnesi

Bir URL adresi için istek yapıldığında bu adres ile ilgili bilgiler global "request" nesnesinde tutulur. "methods" niteliği kullanılarak request tipi(GET, POST vs.) belirlenebilir. *route* dekoratöründe parametre olarak request metodu

```
@app.route('/login', methods=['GET', 'POST'])
```

şeklinde belirtilebilir. Eğer sunucuya kullanıcı verisi gönderilecekse POST metodu kullanılır.

"args" niteliği üzerinden URL içindeki anahtar değer çiftlerine

```
request.args.get('key', '')
```

şeklinde ulaşılabilir. Form verilerine `request.form` üzerinden ulaşılır.

3.17 Session

"session" nesnesi kullanıcı bilgilerini depolamaya izin verir. "session" nesnelerini kullanabilmek için bir gizli anahtar ayarlanmak zorundadır. Bu

- app.secret_key = '...' veya
- SECRET_KEY = '...' şeklinde ayarlanabilir.

Gizli anahtar üretmek için os modülünün urandom() fonksiyonu kullanılabilir.

3.18 Blueprint

Blueprint bir uygulama üzerinde bir veya daha fazla kez kaydedilebilecek işlemler dizisidir. Büyük uygulamalar için uygundur. Flask isteklere cevap verirken görünüm fonksiyonları ve blueprint nesnelerini ilişkilendirir.

Blueprint özelliği kullanılarak birbiri ile ilişkili görünüm fonksiyonları bir modül içinde toplanarak daha modüler bir tasarım oluşturulabilir.

3.19 Blueprint Oluşturma

Örnek olarak projedeki *users* paketi için oluşturulan blueprint Şekil 3.27'de verilmiştir.

```
from flask import Blueprint
mod = Blueprint('users', __name__)
```

Şekil 1.17: Blueprint oluşturma

3.20 Blueprint Kaydetme

Şekil 3.27'de oluşturulan blueprint uygulama üzerinde Şekil 3.28'de gösterildiği gibi kaydedilir.

```
from users.views import mod as user_bp
app.register_blueprint(user_bp, url_prefix='/user')
```

Şekil 1.18: Blueprint kaydetme

Buradaki *url_prefix* değeri bu paketteki görünüm fonksiyonları url adreslerinin başına gelir. Şekil 3.28'deki gibi bir blueprint kaydedildiğinde o modül içindeki görünüm fonksiyonları uygulama üzerinde kaydedilmiş olur.

Kaydedilen görünüm fonksiyonlarına başka bir modül içinden ulaşmak için blueprint oluşturma aşamasında verilen isim(bu örnek için "users") ve fonksiyon adı kullanılır.

Örneğin *users* içindeki *login* fonksiyonuna ulaşmak için *users.login* kullanılır. Aynı modül içinden ulaşılmak isteniyorsa *.login* şeklinde göreceli yönlendirmeler yapılabilir.

3.21 Flask ile SqlAlchemy Kullanımı

SQLAlchemy Python için yazılmış, açık kaynak kodlu SQL ve ORM aracıdır. Bu projede Flask uygulaması ile SQLAlchemy kullanmak için Flask-SQLAlchemy[9] eklentisinden yararlanılmıştır.

3.22 Veritabanı Sınıflarının Tanımlanması ve Sorgulama

Sınıfları tanımlamadan önce *app.py* dosyası içinde veritabanı işleri için kullanılacak nesne Şekil 3.29'daki gibi oluşturulur.

```
from flask.ext.sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config.from_object('config')

db = SQLAlchemy(app)
```

Şekil 1.19: Veritabanı nesnesinin oluşturulması

Oluşturulan bu db nesnesi ilgili modüllerde içe aktarılarak kullanılır. Veritabanı ile ilgili sınıflar *db.Model* sınıfından türetilir. Şekil 3.30'da bir örnek gösterilmiştir.

```
class User(db.Model):

    __tablename___ = "user"

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    email = db.Column(db.String(120), unique=True)

    def __init__(self, username, email):
        self.username = username
        self.email = email

    def __repr__(self):
        return '<User %r>' % self.username
```

Şekil 1.20: Veritabanı nesnesi model sınıfı yazımı

Veritabanı sınıfları tanımlanırken tablo adı yazmak zorunlu değildir. *db.Model* sınıfı normal bir Python sınıfı gibi davranır fakat model üzerinde sorgu yapılabilmesini sağlayan *query* fonksiyonunu sunar. Modeller oluşturulduktan sonra tüm tablolar db nesnesi üzerinden

create_all ve *drop_all* metodları çağrılarak oluşturulup kaldırılabilir.

Şekil 3.31'de birkaç sorgu örneği gösterilmiştir.

```
User.query.all()

User.query.filter_by(username='deniz').first()

User.query.filter(User.email.endswith('@itu.edu.tr')).all()

User.query.order_by(User.username)

User.query.get(1)
```

Şekil 1.21: Veritabanı sorgu örnekleri

İlk sorgu *user* tablosundaki tüm kayıtları döndürür. İkinci sorgu kullanıcı adı "deniz" olan ilk kaydı döndürür. Üçüncü sorgu email adresi "itu.edu.tr" ile sonlanan tüm kayıtları döndürür. Dördüncü sorgu kayıtları kullanıcı adına göre sıralayarak döndürür. Son sorgu ise birincil anahtarı "1" olan kaydı döndürür.

Görünüm fonksiyonları içinde *first* yerine *first_or_404, get* yerine *get_or_404* metodları da kullanılabilir. Bu durumda eğer bir kayıt veritabanında bulunamazsa geriye 404 hata kodu döndürülür.

3.23 foreign key, relationship Kullanımı

Flask-SQLAlchemy ile iki tablo arasında ilişki kurarken *db.ForeignKey()* ve *relationship()* kullanılır. Projedeki *transaction* ve *pdf* tabloları arasında Şekil 3.32'de gösterildiği gibi ilişki kurulmuştur.

```
class Transaction(db.Model):

    __tablename___= "transaction"

    transaction_id = db.Column(db.String(40), primary_key=True)
    datetime = db.Column(db.DateTime)
    workflow_id = db.Column(db.String(40),db.ForeignKey('workflow.workflow_id'))
    pdfs = db.relationship('Pdf', backref='transaction', lazy='dynamic')

class Pdf(db.Model):

    __tablename___= "pdf"

    pdf_id = db.Column(db.String(40), primary_key=True)
    name = db.Column(db.String(120))
    path = db.Column(db.String(255))
    transaction_id = db.Column(db.String(40),
                               db.ForeignKey('transaction.transaction_id'))
```

Şekil 1.22: Dış anahtar kullanımı

Şekil 3.32'deki örnekte *pdf* tablosunda *transaction* tablosuna *transaction_id* üzerinden bir dış anahtar tanımlanmıştır. *transaction* tablosunda ise *pdf* tablosuyla ilişki olduğu belirtilmiştir. *pdf.transaction* ifadesi bu pdf ile ilgili tüm *transaction* kayıtlarını döndürür. `"lazy="dynamic""` parametresi bulunan kayıtları hemen veritabanından yüklemek yerine geriye bir sorgu nesnesi döndürür.

3.24 Ekranda Mesajların Gösterilmesi

Flask, mesajları göstermek için *flash* fonksiyonunu kullanır. Bu fonksiyon parametre olarak ekranda gösterilecek mesajı alır. Tüm mesajlara erişmek için *get_flashed_messages* fonksiyonu kullanılır.

```
<ul class=”flashes”>
    {% for message in get_flashed_messages() %}
        <li>{{ message }}</li>
    {% endfor %}
</ul>
```

Şekil 1.23: Ekranda mesajların gösterilmesi

Html dosyası içine yazılan Şekil 3.33’teki kod parçasına Python kodu içinden

```
flash('gönderilecek mesaj')
```

şeklinde istenen mesaj gönderilebilir.

Ayrıca mesajlar kategorilere ayrılabilir. Varsayılan kategori “message” kategorisidir. Farklı kategoride mesaj gönderilmek istendiğinde *flash* fonksiyonuna ikinci bir parametre verilerek belirtilir.

```
{% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
        <ul class=”flashes”>
            {% for category, message in messages %}
                <li class="{{ category }}">{{ message }}</li>
            {% endfor %}
        </ul>
    {% endif %}
{% endwith %}
```

Şekil 1.24: Mesajların kategorilere ayrılması

Şekil 3.34’teki kod parçasına Python kodu içinden

```
flash('gönderilecek mesaj', 'error')
```

şeklinde mesaj gönderilebilir.

3.25 Hata Ayıklama Modu

Debug desteği aktif hale getirilirse sunucu kod üzerinde değişiklik olduğunda kendini tekrar yükler. Ayrıca hata durumunda bir hata ayıklayıcı sağlar. Bu mod iki şekilde aktif hale getirilebilir:

- app.debug = True
  app.run()
- app.run(debug=True)

Hata ayıklama modu sadece geliştirme aşamasında açık olmalıdır.

3.26 HTTP Durum Kodları[7]

- **200 OK:** Yapılan isteğin(request) başarılı bir şekilde alındığını ve anlaşıldığını gösterir. Cevapta(response) gönderilen bilgi istekte hangi metodun(GET,POST vs.) kullanıldığına bağlı olarak belirlenir.

- **201 Created:** İsteğin başarılı bir şekilde gerçekleştirilip yeni bir kaynak yaratıldığını gösterir.

- **202 Accepted:** İsteğin işlenmek üzere kabul edildiğini fakat henüz tamamlanmadığını gösterir.

- **204 No Content:** İsteğin başarılı bir şekilde işlendiğini fakat cevap gövdesinde herhangi bir içeriğin döndürülmediğini gösterir.

- **301 Moved Permanently:** İstenen kaynağın yeni bir kalıcı URI adresine atandığını gösterir.

- **304 Not Modified:** Kaynağın en son yapılan istekten bu yana değişmediğini gösterir. Döndürülen cevapta kaynak gövdesi gönderilmez.

- **400 Bad Request:** İsteğin hatalı yazım sonucu sunucu tarafından anlaşılamadığını gösterir.

- **401 Unauthorized:** İsteğin kullanıcı yetkilendirmesi gerektirdiğini gösterir.

- **403 Forbidden:** Bu hatada sunucu isteği anlamıştır fakat gerçekleştirmeyi reddeder. 401 hatasının aksine yetkilendirilmiş olmak bir fark yaratmaz.

- **404 Not Found:** İstenen kaynağın bulunamadığını gösterir.

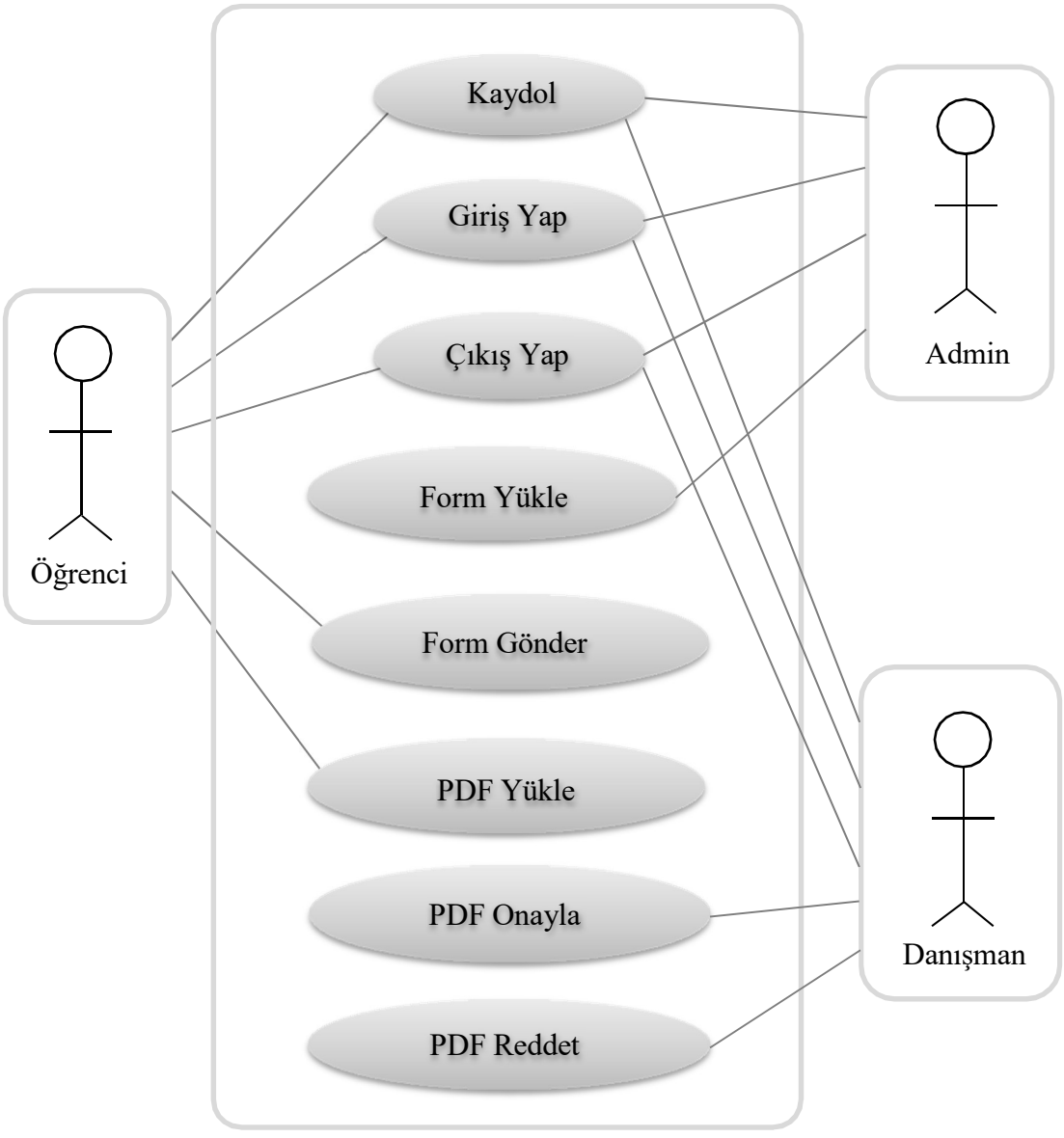
- **500 Internal Server Error:** Sunucunun isteği gerçekleştirmesini engelleyen beklenmeyen bir durum ile karşılaştığını gösterir.

- **503 Service Unavailable:** Sunucunun o an yapılan isteği aşırı yüklenme gibi birtakım sorunlardan dolayı karşılayamadığını gösterir.

4.ANALİZ VE MODELLEME

4.1. Kullanım Senaryoları

Sistemi öğrenci, danışman ve admin olmak üzere üç tip kullanıcı kullanmaktadır. Use case diyagramı Şekil 4.1'de gösterilmiştir.



Şekil 1.25: Kullanım senaryoları diyagramı

SG1: Kaydol

Birincil aktör: Öğrenci, danışman, admin

Ön koşullar: Kayıt bağlantısı ana sayfada görünmektedir.

Son koşullar: Kullanıcı veritabanına eklenmiştir.

Ana başarılı senaryo(temel akış):

1. Kullanıcı ana sayfadaki *REGISTER* bağlantısına tıklar.
2. Sistem kayıt formunu gösterir.
3. Kullanıcı bir kullanıcı adı ve iki kere aynı şifreyi girer.
4. Sistem kullanıcı kaydını yapar.

Uzantılar(alternatif akışlar):

3a. Şifreler eşleşmez.

1. Sistem hata mesajı verir ve şifrelerin yeniden girilmesini ister.
2. Kullanıcı bilgileri tekrar girer.

4a. Sistemde aynı kullanıcı adına sahip başka bir kullanıcı kaydı vardır.

1. Sistem hata mesajı verir ve bilgileri yeniden ister.
2. 3 numaralı adıma dönülür.

SG2: Giriş Yap

Birincil aktör: Öğrenci, danışman, admin

Ön koşullar: Giriş yapma bağlantısı ana sayfada görünmektedir.

Son koşullar: Kullanıcı sisteme giriş yapmıştır.

Ana başarılı senaryo:

1. Kullanıcı ana sayfadaki *LOGIN* bağlantısına tıklar.
2. Sistem giriş formunu gösterir.
3. Kullanıcı, kullanıcı adını ve şifresini girer.
4. Sistem kullanıcının sisteme giriş yapmasını sağlar.

Uzantılar:

4a. Kullanıcı kaydı sistemde bulunamaz.

1. Sistem hata sayfasını gösterir.
2. İlk adıma dönülür.

SG3: Çıkış Yap

Birincil aktör: Öğrenci, danışman, admin

Ön koşullar: Kullanıcı sisteme giriş yapmıştır.

Son koşullar: Kullanıcı sistemden çıkmıştır.

Ana başarılı senaryo:

1. Kullanıcı ana sayfadaki *LOGOUT* bağlantısına tıklar.
2. Sistem kullanıcının çıkış yapmasını sağlayarak ana sayfaya döner.

SG4: Form Yükle

Birincil aktör: Admin

Ön koşullar: Admin sisteme giriş yapmıştır.

Son koşullar: Form başarıyla yüklenmiştir.

Ana başarılı senaryo:

1. Admin form yükleme bağlantısına tıklar.
2. Sistem form yükleme ekranını gösterir.
3. Admin yükleyeceği pdf formu seçip *upload* butonuna tıklar.
4. Sistem form yükleme işini tamamlar.

Uzantılar:

3a. Yüklenen form pdf formatında değildir.

1. Sistem hata mesajı verir ve tekrar yükleme ekranını gösterir.
2. 3 numaralı adıma dönülür.

4a. Aynı isimde bir form sistemde mevcuttur.

1. Sistem hata mesajı verir ve tekrar yükleme ekranını gösterir.
2. 3 numaralı adıma dönülür.

SG5: Form Gönder

Birincil aktör: Öğrenci

Ön koşullar: Öğrenci sisteme giriş yapmıştır. Doldurulması gereken pdf formu sistemde mevcuttur.

Son koşullar: Doldurulan pdf formu başarıyla gönderilmiştir. İş akışı güncellenmiştir. Transaction kaydı yapılmıştır.

Ana başarılı senaryo:

1. Öğrenci form gönderme taskına tıklar.
2. Sistem boş formu ekrana getirir.
3. Öğrenci formu doldurur ve *SUBMIT* butonuna tıklar.
4. Sistem form verisini alıp çıktı pdf dosyası oluşturur, iş akışını günceller, transaction kaydı yapar.
5. Öğrenci gönderdiği pdf form için onay bekleme durumuna geçer.

SG6: PDF Yükle

Birincil aktör: Öğrenci

Ön koşullar: Öğrenci sisteme giriş yapmıştır.

Son koşullar: PDF belgesi başarıyla yüklenmiştir. İş akışı güncellenmiştir. Transaction kaydı yapılmıştır.

Ana başarılı senaryo:

1. Öğrenci sıradaki pdf gönderme taskına tıklar.
2. Sistem pdf yükleme ekranını gösterir.
3. Öğrenci yükleyeceği pdf belgesini seçip *upload* butonuna tıklar.
4. Sistem pdf belgesini kaydeder, transaction kaydı yapar ve iş akışını günceller.
5. Öğrenci gönderdiği pdf belgesi için onay bekleme durumuna geçer.

Uzantılar:

3a. Yüklenen form pdf formatında değildir.

1. Sistem hata mesajı verir ve tekrar yükleme ekranını gösterir.
2. 3 numaralı adıma dönülür

4a. Aynı isimde bir pdf sistemde mevcuttur.

1. Sistem hata mesajı verir ve tekrar yükleme ekranını gösterir.
2. 3 numaralı adıma dönülür.

SG7: PDF Onayla

Birincil aktör: Danışman

Ön koşullar: Danışman sisteme giriş yapmıştır. Öğrenci gönderdiği pdf için onay bekleme durumundadır.

Son koşullar: İş akışı güncellenmiştir. Transaction kaydı yapılmıştır.

Ana başarılı senaryo:

1. Danışman karşısına gelen onay bekleme taskına tıklayarak pdf belgesini indirir.
2. Danışman *APPROVE* butonuna tıklar.
3. Sistem iş akışını günceller ve transaction kaydı yapar.

SG8: PDF Reddet

Birincil aktör: Danışman

Ön koşullar: Danışman sisteme giriş yapmıştır. Öğrenci gönderdiği pdf için onay bekleme durumundadır.

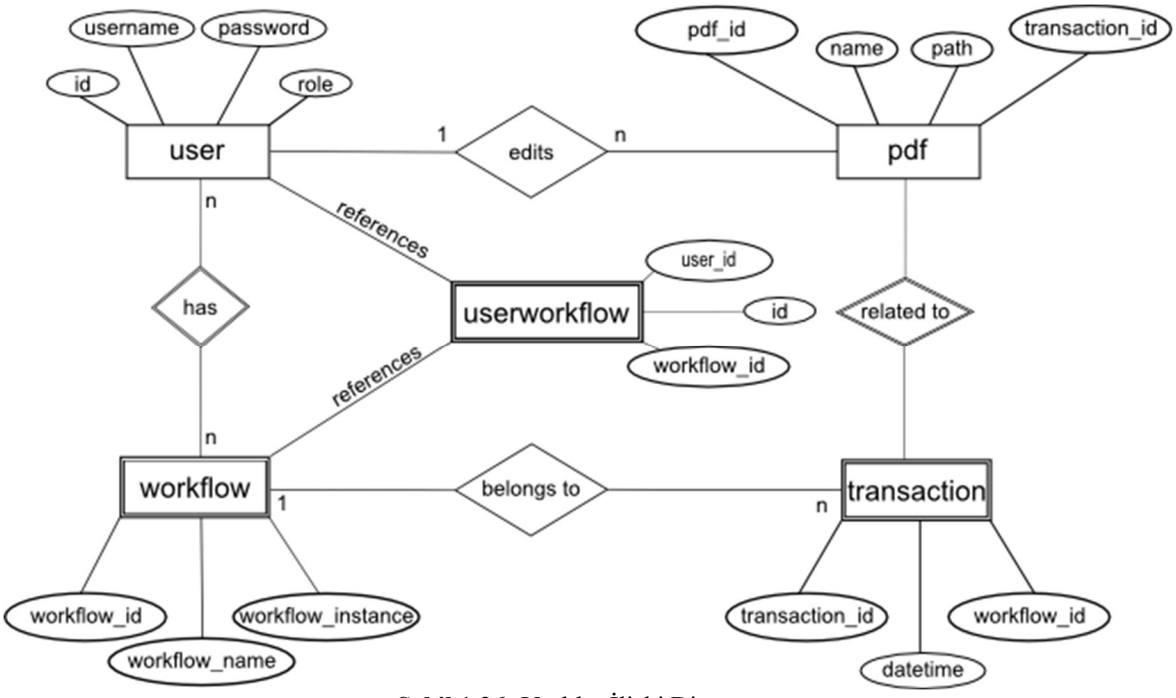
Son koşullar: İş akışı güncellenmiştir. Reddedilen pdf ve kaydı silinmiştir.

Ana başarılı senaryo:

1. Danışman karşısına gelen onay bekleme taskına tıklayarak pdf belgesini indirir.
2. Danışman *REJECT* butonuna tıklar.
3. Sistem pdf ve kaydını siler ve iş akışını günceller.

4.2. Veri Modeli

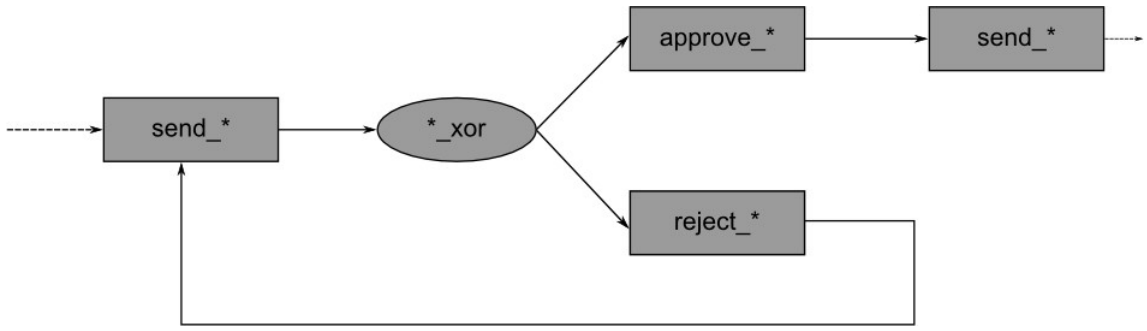
Projeye ait varlık-ilişki diyagramı Şekil 4.2'de gösterilmiştir.



Şekil 1.26: Varlık - İlişki Diyagramı

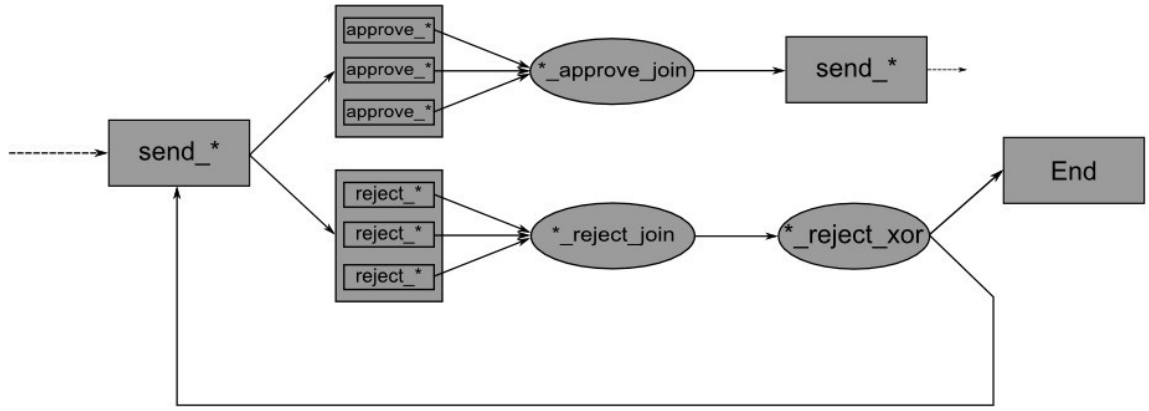
4.3. Dönem Ödevi İş Akışı Modeli

Bu projede gerçeklenen bitirme ödevi süreci için iki farklı model oluşturulmuştur. Şekil 5.8 ve Şekil 5.9'da bu modellerden birer parça gösterilmiştir. Tam iş akışı modeli bu parçaların birbirine eklenmesiyle oluşur.



Şekil 1.27: Bitirme Ödevi İş Akışı İlk Model

Şekil 5.8'de gösterilen model *project_spec.xml* dosyasında tanımlanmıştır. Bu modele göre önce öğrencinin önüne bir gönderme işi çıkar. Tanım dosyasında *send_** iş parçasına *pdf_form* anahtarıyla bir veri alanı eklendiyse öğrencinin karşısına doldurulması gereken bir pdf formu çıkar. Aksi takdirde bir dosya yükleme ekranı ile karşılaşılır. Bir pdf belgesi gönderildikten sonra öğrenci onay bekleme durumuna geçer. Eğer danışman formu onaylarsa *approve_** iş parçası tamamlanarak bir sonraki adıma geçilir, onaylamazsa *reject_** iş parçası tamamlanıp *send_** adımına geri dönülür.



Şekil 1.28: Bitirme Ödevi İş Akışı İkinci Model

Şekil 5.9'da gösterilen model *project_spec2.xml* dosyasında tanımlanmıştır. Bu modele göre ilk modelde olduğu gibi önce öğrencinin önüne bir gönderme işi çıkar. Öğrenci pdf

belgesini gönderdikten sonra onay bekleme durumuna geçer. m adet kullanıcıdan n tanesi(bu model için m=3 ve n=2) onaylarsa *_*approve_join* iş parçası tamamlanır ve bir sonraki adıma geçilir. Aksi takdirde m kişiden n tanesi reddederse *_*reject_join* ve

*_*reject_xor* iş parçaları tamamlanarak *send_** adımına geri dönülür.

Üçüncü model *project_spec3.xml* dosyasında tanımlanmıştır. Bu model üstte anlatılan iki modelden karışık parçalar içermektedir.

İş akışı tanım dosyaları yazılırken iş parçalarının adları yukarıdaki şekillerde gösterilen formatta olmalıdır.

5.SONUÇ

Proje planının amaç kısmında belirtilen sürece ait belgelerin katılımcılar arasında dolaşımını otomatik hale getirecek bir sistem gerçeklenmesi hedeflenmiştir. Başarım kriteri olarak verilen, örneğin dönem projesi ödevi süreci olduğu bir adet iş akışı sistem üzerinde gerçeklenmiştir. Buna ek olarak dönem projesi ödevi süreci ile ilgili iki model daha çalışılmıştır.

Projede gerçeklenemeyen durumlardan biri bir öğrencinin aynı anda birden fazla iş akışını çalıştıramamasıdır. Ayrıca formlarda Türkçe karakter kullanma sorunu çözülemediğinden girdilerin Türkçe'ye özel karakterler içermemesi gerekmektedir.

Genel olarak iş akışı sistemleri üzerinde çalışacak olanlar iş akışı kalıpları hakkında bilgi sahibi olunmuştur. Ayrıca bir iş akışı motorunun yürütebileceği bir iş akışının nasıl tanımlanacağı ve daha sonra bu motoru kullanarak tanımlanan iş akışının nasıl çalıştırılacağı öğrenilmiştir.

6.KAYNAKLAR

[1] N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar, "Workflow Control-Flow Patterns: A Revised View", BPM Center Report BPM-06-22, BPMcenter.org, 2006.

[2] SpiffWorkflow [Online]. Available: https://github.com/knipknap/SpiffWorkflow

[3] Adobe Systems Incorporated, "Interactive Features," in *PDF Reference,* 5th ed. ch. 8, sec. 6, pp. 669-670.

[4] Flask [Online]. Available: http://flask.pocoo.org/docs/

[5] pdftk [Online]. Available: http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/

[6] Understanding Workflow [Online]. Available: http://docs.oracle.com/cd/E17984_01/doc.898/e14729/understand_workflow.htm

[7] HTTP Status Codes [Online]. Available: http://www.restapitutorial.com/httpstatuscodes.html#