Ulakbus Documents Documentation

Release 0.0.1

Mustafa Tola, Osman Sonmezturk, Ali Riza Keles

September 08, 2015

Contents

1 İçerik: **1**
1.1 Yazılım Geliştirme ve Test Döngüsü 1
1.2 Yazılım Tasarım Analizi Belgesi 12

CHAPTER 1

İçerik:

1.1 Yazılım Geliştirme ve Test Döngüsü

1.1.1 Yazılım Geliştirme Modeli

Yazılım geliştirme modeli döngüsel artımlı (iterative and incremental) modele bağlı XP [1] olacaktır.

Geliştirme ve Test Döngüsü

Extreme Programming modeline uygun şekilde 3 haftalık dönemlerde küçük sürüm planı yapılacak ve yazılım geliştirme ve test döngüsü bu sürüm planına bağlı olarak ilerleyecektir.

Sürüm planına dahil edilen işler (issue) geliştiriciler tarafından uygun bir branchte çözülecektir. Geliştiriciler, geliştirme faaliyetleri boyunca aşağıda detayları belirtilen otomatik ve manuel testleri build aşamasından önce uygulayacaklar, ancak bu testlerden geçen kaynak kod sonraki aşamaya geçebilecektir. Kurulum ve yayınlama aşamasında ise bu aşamanın testleri yapılacak ve sonuçlar geliştiricilere bildirilecektir. Ayrıca geliştirilen özelliğe göre kabul testleri de bu aşamada yapılacaktır. Bu geliştirme döngüsü 3 hafta boyunca artımlı şekilde ilerleyecektir.

Bu 3 haftanın sonunda ise sürüm adayı (release candidate) çıkartılacak ve bunun üzerinde kabul testleri ve önceden hazırlanmış test senaryoları, kullanılabilirlik, performans, güvenlik gibi testler yapılacaktır. Bu testlerin kabul sınırları içinde geçilmesi halinde sürüm ortaya çıkartılacaktır.

Bunlara ek olarak her gece, gecelik derlenmiş kod (nightly builds) yayınlanacak, yazılımın tüm bileşenlerine ait tüm otomatik testler bu aşamada gerçekleşecektir.

Sürüm Planı

Sürüm planı 3 hafta geliştirme + 1 hafta kabul test süreçleri şeklinde planlanır. 7 ay boyunca toplam 7 sürüm çıkarılacaktır. Sürüm planı ihtiyaç analizi isterleri ve YTA belgesi ışığında yeni özelliklerin planlanması, önceki sürümden kalan hataların kapatılması, topluluk geri bildirimlerinden seçilen işlerin (issues) tamamlanması hedefleriyle yapılır.

Depolar

Her bileşen kendi deposunda yaşam döngüsüne devam eder. Birbirlerini etkileyen issuelar için referans verilir. Başlıca geliştirme depolarımız: **-SpiffWorkFlow:** İş akışı kütüphanesi geliştirme deposudur. Orjinalden fork edilmiştir.

-Pyoko: Pyoko Riak/Solr ORM geliştirme deposudur. Zetaops tarafından geliştiriliyor.

[1] http://www.extremeprogramming.org

-Zengine: Zengine Framework geliştirme deposudur. Zetaops tarafından geliştiriliyor.

-Zaerp (Ulakbus): Ana uygulama backend geliştirme deposudur. Zetaops tarafından geliştiriliyor.

-Zaerp-UI (Ulakbus-ui): Ana uygulama frontend geliştirme deposudur. Zetaops tarafından geliştiriliyor.

-ZCloud: Bulut araçları geliştirme deposudur. Zetaops tarafından geliştiriliyor.

Yardımcı kütüphaneler ile fork edilmiş kütüphaneler haricindeki depo isimleri, daha sonra Ulakbim tarafından verilecek isimlerle değiştirilecektir.

Topluluk

Roller

Geliştiriciler Depolara kod katkısında bulunacak topluluk üyeleridir. Geliştirici Rehberleri ve Git Workflow[**] Belgesinde açıklanan akışa uygun şekilde geliştirme faaliyetlerine katılırlar.

Beta Test Edicileri Patch ve Minor sürümleri test ederek geri bildirimlerde bulunarak geliştirme faaliyetine katkıda bulunurlar. **Analiz Uzmanları** Yüksek Öğrenim Kanunu, Akademik Birimlerin Yönetmelikleri, akademinin yerleşik teammülerini bilen, projenin kapsamına detaylarıyla hakim topluluk üyeleridir. Topluluğun talep ettiği yeni özellikler, iş akışlarının değiştirilmesi, kanun ve yönetmeliklerdeki değişikliklerin projeyi nasıl etkileyeceği gibi konularda tavsiyelerde bulunurlar. **Topluluk Moderatörleri** Topluluğun tartışmalarını kolaylaştırmak, konu başlıklarını bağlantılandırmak, tartışmaların gidişatını sorularla belirli hedeflere yönlendirmek gibi görevleri olan topluluk üyeleridir.

Geliştirici ve Katkıcı Rehberleri

- **Geliştirme Ortamı Kurulum Rehberi**
- **UI Geliştirci Rehberi**
- **Zaerp Geliştirici Rehberi**
- **Workflow ve Spiff WF Rehberi**
- **Zato ve Servis Yazma Rehberi**
- **Git Workflow**

Test Döngüsünün Amaçları

- Yazılım geliştirme döngüsünün denetlenebilir, kolay yönetilebilir ve ölçülebilir hale gelmesini sağlamak,
- Problemleri somutlayarak, çok sayıda yazılımcının daha kolay işbirliği yapabilmesine yardımcı olmak,
- Her bir yazılım parçasını çok yönlü şekilde zamanında test ederek, geliştirme döngüsünün sonraki aşamalarına en az hata ile devam etmek,
- Bir bileşende yapılan geliştirmenin diğer bileşenleri nasıl etkilediğini zamanında görebilmek,
- Kod kalitesini arttırmak,
- Kod yazım desenleri açısından bütünlük sağlamak ve okunabilirliği arttırmak,
- Kurulum sırasında yazılım bileşenleri ve birbirlerine olan bağımlılıkları doğrulamak,
- Yazılımın farklı platformlarda ve farklı ortam değişkenleriyle başarılı bir şekilde kurularak, beklenen şekilde çalıştığından emin olmak,

- Yazılımın beklenen şekilde çalışmasının ardından, önceden belirlenmiş çeşitli yük testleri altında aynı şekilde davranmaya devam ettiğinden emin olmak,
- Ortaya çıkan ürünün, ister belgesindeki işlevleri karşılayıp kaşılamadığını doğrulamak,
- Ürünün kullanım kolaylığı, kullanıcı deneyimi, performans açısından tatmin edici ve standartları karşıladığından emin olmaktır.

1.1.2 Sürüm Planına Bağlı Geliştirme Döngüsü Testleri

Eşli Gözden Geçirme (Peer Code Review)

Geliştiriciler, kodlarını çalıştıkları branchtan, master brancha merge etmeden önce bir diğer geliştirici ile birlikte gözden geçireceklerdir. Bu gözden geçirme sırasında aşağıdaki kontrol listesine uygunluk aranacaktır:

- **Kod Stili:** Kod, Statik analiz araçları tarafından yakalanamayan method ve değişken isimlerinin proje standartlarına uygunluğu gibi kriterlere karşı incelenir.
- **Belgelendirme:** Mümkün olduğunca yorum satırlarına gerek duyulmayan, anlaşılır kod yazılmalıdır. Ancak çeşitli nedenlerle kolayca anlaşılmayan bir kod öbeği varsa, bunun nedeni ve nasıl çalıştığı belgelendirilmelidir.
- **Girdilere Karşı Savunma:** Kullanıcıdan ya da üçüncü parti servis ve uygulamalardan gelen veriler, temizlenip biçimlendirilmeli, hata denetiminden geçirilmeli ve gerekiyorsa try/except blokları içerisinde işlenmelidir.
- **Test Edilebilirlik:** Sınıf ve metodlar birim testlerinin kolayca yazılabilmesine olanak verecek şekilde tasarlanmalıdır. Arayüzler (interface) mümkün olduğunca test ortamında taklit edilebilir olmalıdır.
- **Testler ve Kapsam:** Kodun tamamını kapsayan, doğru tasarlanmış yeterli sayıda birim testi yazılmış olmalıdır. Dış servislere bağımlı işlevlerin testi için gerekli mocking kütüphane ve sunucuları kullanılmalıdır.
- **Ayarlanabilirlik:** Uygulamanın çalışmasını ve davranışını etkileyen, dosya dizin yolları, açılır menüde gösterilecek seçenek sayısı gibi değerler ya kullanıcı tarafından ya da uygulamanın konfigurasyon standardına uygun şekilde (çevre değişkenleri) ile ayarlanabilir olmalıdır.
- **Çöp Kod:** Yorum satırı haline getirilmiş kod olmamalıdır. Silinen herşey sürüm kontrol sisteminden geri getirilebilir.
- **Yapılacaklar:** Todo olarak bırakılmış eksiklerin, sorun çıkarmayacağından emin olunmalıdır.
- **Döngüler:** Döngüler uzunluk ve döngüden çıkış kriterlerinin uygunluğuna karşı denetlenmelidir.
- **Mevcudiyet Denetimi:** Nesneler, kullanılmadan önce, o kapsamda mevcut olup olmadıklarına karşı denetlenmelidir. Bu denetimler, birçok hatanın kaynağında yakalanmasını sağlar.
- **Kod Tekrarı:** Aynı işi yapan kodların tekrar yazılmasından kaçınılmalıdır. Bu amaçla özellikle projeye sonradan katılan geliştiricilerin, mevcut utility metodlarından haberdar olmaları sağlanmalıdır.

Arkauç Testleri

Bileşen (Birim) Testleri

Sistemin arkaucunu oluşturan bileşenlerin tümü py.test test frameworkü kullanılarak test edilecektir. Birim testleri, kodun en az %60'ını kapsayacaktır (code coverage). Uygulamayı oluşturan tüm bileşenlerin birim testleri, kendi ana dizinleri altında "tests" dizininde tutulur. "py.test" komutu, proje ana dizini altında çalıştırıldığında, ismi "test" ile başlayan tüm Python dosyalarını tek tek tarayıp, içlerinde yine ismi "test" ile başlayan metodları çalıştırır. Örnek bir birim test aşağıda görülebilir.

```
from tests.data.test_data import data
from tests.data.test_model import Student
def test_model_to_json_compact():
        st = Student(**data)
        st.join_date = data['join_date']
        st.AuthInfo(**data['AuthInfo'])
        for lct_data in data['Lectures']:
                lecture = st.Lectures(**lct_data)
                lecture.ModelInListModel(**lct_data['ModelInListModel'])
                for atd in lct_data['Attendance']:
                        lecture.Attendance(**atd)
                for exam in lct_data['Exams']:
                        lecture.Exams(**exam)
        clean_value = st.clean_value()
        assert data == clean_value
```

Örnek birim testi 1 Py.test, standard "assert" ifadesinin testin başarılı olup olmadığının kontrolü için kullanır. Bu sayede testlerin hazırlanması, yeni geliştiriciler için neredeyse hiçbir ek öğrenme süreci gerektirmez.

Yukarıdaki test, benchmark eklentisiyle birlikte aşağıdaki gibi bir çıktı verecektir.

```
================== test session starts ==================
rootdir: /home/whogirl/Works/pyoko, inifile:
plugins: benchmark
collected 4 items
tests/test_model_to_json.py
— benchmark: 1 tests, min 5 rounds (of min 25.00us), 1.00s max time,
Name (time in us) Min Max Mean StdDev Rounds Iterations
test_model_to_json 214.0999 41221.8571 319.0611 1019.8894 1629 1
================== 1 passed in 1 .37 seconds ==================
```

Test frameworkünün, kod kapsam analiziyle birlikte çalıştırılması sonucu aşağıdaki gibi bir çıktı elde edilecektir. Bu örnekte pyoko modülünün test kapsam oranı %58 olarak görünmektedir.

py.test –cov pyoko ================== test session starts ================== platform darwin – Python 2.7.6 – py-1.4.27 -

Name pyoko/__init__	Stmts 1	Miss 0	Cover 100%
pyoko/db/base	165	118	28%
pyoko/db/connection	5	0	100%
pyoko/db/schema_update	20	10	50%
pyoko/db/solr_schema_fields	1	1	0%
pyoko/exceptions	11	0	100%
pyoko/field	46	8	83%
pyoko/lib/__init__	1	0	100%
pyoko/lib/py2map	22	17	23%
pyoko/lib/utils	16	5	69%
pyoko/model	106	7	93%
pyoko/settings	2	0	100%
TOTAL	397	166	58%

================== 4 passed in 3.14 seconds ==================

HİTAP gibi test ortamı sunmayan üçüncü parti servislerle veri alışverişi yapan modüllerin testleri, harici servisin istek

/ yanıt setlerini mimik eden Wiremock [2] gibi bir simulatöre karşı yapılacaktır. Bu amaçla üretim ortamında servise gönderilen ve alınan veri trafiği kaydedilecek ve simulatör bu verilerle "eğitilecektir".

Pyoko

Veri erişim katmanı (DAL) olarak görev yapacak olan Pyoko kütüphanesi için yazılacak birim testleri, veri doğruluğu ve API işlevlerine ek olarak çalışma hızı ve bellek kullanımı gibi kriterleri de göz önünde bulunduracaktır.

SpiffWorkflow Engine

Üçüncü parti bir kütüphane olarak projeye eklenmiş olan SpiffWorkflow'un geliştirilmesi ve bakımı uygulamanın ihtiyaçları doğrultusunda sürdürülecektir. Buna ek olarak, BPMN iş akışlarının doğruluğunun devamlı olarak sınanabilmesi için entegre bir test kaydetme ve çalıştırma modülü geliştirilecektir.

İş Akışı (Workflow) Testleri

Sistemin tüm işlevlerinin üzerine inşa edileceği BPMN iş akışları, verilen girdilerle beklenen davranışı gösterip göstermediğine karşı test edilecektirler. Böylece iş akışları üzerinde yapılacak güncellemelerin, amaçlanan dışında yan etkilere neden olmadığından emin olunması sağlanacaktır.

Benchmark Testleri

İş akışı motoru, Pyoko gibi görev kritik modüllerin performansı pytest-benchmark eklentisi kullanılarak devamlı olarak ölçülüp kaydedilerek bu modüllerin performanslarındaki zamana bağlı değişim takip edilecek ve olası gerilemeler önlenecektir.

Servis Testleri

Uygulamanın birçok işlevi Zato ESB üzerinde çalıştırılacak mikro servisler üzerinden sunulacaktır. Bu servislerin işlevselliği ve API uyumluluğu zato-apitest frameworkü ile yazılacak testler ile sınanacaktır.

Kural Motoru (Rule Engine) Testleri

Uygulamanın iş mantığının önemli bir kısmını oluşturan kural setleri, belirli girdilerle beklenen çıktıları verip vermediklerine karşı denetelenmelidir.. Bu amaçla kural setleri standart birim testleri içerisinde kural motoru ile işletilerek beklenen çıktıyla eşleştirilecektir.

Kurulum ve Yayınlama Aşaması Testleri

Kurulum ve Yayınlama (Build Release) aşamasında Buildbot aracılığı ile kurulum ve kütüphane bağımlılık testleri uygulamanın tüm bileşenlerine ait birim testleri entegrasyon testleri uygulanacaktır.

Test Sunucuları ve Geliştirme Test Döngüsü

Her iş (issue) kendi geliştirme branchinde minik commitler ile geliştirilir. Bu branchlere yapılan pushlar buildbot u tetikleyip, kurulum ve yayınlama aşamasını başlatır. Bu branchten elde edilen kod diğer kütüphaneler ile birlikte biraraya getirilip kurulum işlemi yapılır ve gerekli testler çalıştırılır. Testleri geçen kaynak kod ve uygulama, geliştiricinin erişebileceği bir porttan yayınlanır.

[2] http://wiremock.org/

branch issue/58 → push → buildbot run tests → branch deployed at port 9091 branch issue/59 → push → buildbot run tests → branch deployed at port 9092 branch issue/60 → push → buildbot run tests → branch deployed at port 9010

Sonuca kavuşturulan işler (issues) elle master branch ile birleştirilir (merge). Masterdaki bu değişiklik geliştirme aşamasındaki gibi buildbot u tetikler. Kurulum ve yayınlama işlemi bu branche karşı yapılır. Yayınlama sabit bir porttan yapılır (8080).

Bunun yanısıra gecelik derlenmiş kod (nightly builds) da master branchlerden gerkçekleşir ve aynı portta yayınlanır.

master → manual merge → buildbot run tests → master deployed at 8080 master → automatic buildbot nightly build (backend + UI) → master deployed at 8080

Master Merge ve Nightly Builds işlemleri sırasında şu bileşenler biraraya getirilir: 1. final git master repo status 2. application artefacts (xml files, json files, binary files, zip files, config files, certs) 3. db schema migration file

Yayına Alma (Production)

Sürüm adayı haline gelen master branchte bulunan kaynak kod, aşağıda detaylı şekilde anlatılan sürüm öncesi kabul testlerinden geçer. Bu testlerin başarılı olması halinde, semantik sürümlendirme sistemine [3] göre etiketlenir (tagging).

Semantik sürümlendirme sistemine göre kullanılacak desen MAJOR.MINOR.PATCH şeklindedir. Buna göre 3 haftalık küçük sürümler MINOR, gündelik çözülen işler PATCH, önceden belirlenmiş hedefleri kapsayan fazların sonunda ise MAJOR değerleri arttırılır.

MINOR sürümler çıktıkça, buildbot taglenmiş sürümdeki depoları production ortamında yayına alır. Gerekli dosyaları kopyalar ve veritabanı şemalarını yeni sürümlere göç ettirir.

Kullanıcı Arayüzü Testleri

Kullanıcı Arayüzü AngularJS ile Model-View-Controller (MVC) yapısı ile programlanacaktır. Modül yapısı aşağıdaki örnekte olduğu gibidir:

```
app/
        dashboard/
            dashboard.html (template)
            dashboard.js (Controller ve Model tanımlarının olduğu dosya)
            dashboard.test.js (Testlerin yazıldığı dosya)
            ... (diğer modüller)
            app.css (stil dosyası)
            app.js (Uygulamanın tanımlandığı yapılandırıldığı dosya)
    karma.conf.js (testlerin çalışma zamanı yapılandırmalarını içeren dosya)
    e2e-tests/
        #protractor.conf.js (e2e testlerini çalıştıran protractor yapılandırma dosyası) #scenar-
        ios.js (e2e test senaryolarının bulunduğu dosya)
```

Bileşen (Birim) Testleri

Uygulamada *.test.js dosyaları modüllerin Unit testlerinin barındığı dosyalardır. Unit testler Jasmine test uygulama çatısı kullanılarak yazılır. Uygulamanın Giriş (Login) modülü için yazılmış bir örnek aşağıdaki gibidir:

[3] http://semver.org/

```
describe('zaerp.login module', function () {
        beforeEach(module('zaerp.login'));
        describe('login controller', function () {
            it('should have a login controller', inject(function (){
    expect('zaerp.login.LoginCtrl').toBeDefined();
            }));
        });
    });
```

Bu test örneğinde "login controller"ının tanımlanmış olması gerekliliği test edilmektedir. Kullanıcı arayüzü unit testleri karma test yürütücüsü (test runner) ile çalıştırılır. Bunun için yukarıda açıkladığımız yapıda da görüleceği gibi "karma.conf.js" ismiyle bir yapılandırma dosyası bulunmaktadır. Karma yapılandırma örneği aşağıdaki gibidir:

```
module.exports = function (config) {
    config.set({
        basePath: './',
            files: [
                'app/bower_components/angular/angular.js',
                'app/bower_components/angular-route/angular-route.js',
                'app/bower_components/angular-mocks/angular-mocks.js',
                'app/app.js',
                'app/components/**/*.js',
                'app/login/*.js',
        ],
        autoWatch: true,
        frameworks: ['jasmine'],
        browsers: ['ChromeCanary'],
        plugins: [
            'karma-chrome-launcher',
            'karma-firefox-launcher',
            'karma-jasmine',
            'karma-junit-reporter'
        ],
        junitReporter: {
            outputFile: 'test_out/unit.xml',
            suite: 'unit'
        }
    });
};
```

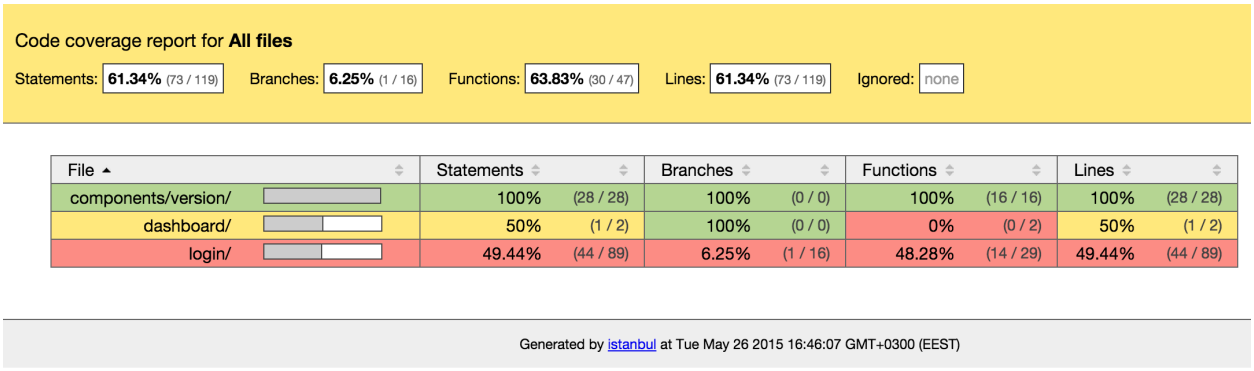
Bu yapılandırmada test dosyalarının hangileri olduğu ve testlerin çalışması için uygulama bağımlılıkları (dependencies) "files" anahtarında, hangi test uygulama çatısı kullanılacağı "frameworks" anahtarında, hangi tarayıcının kullanılacağı "browsers" anahtarında ve eklentiler "plugins" anahtarında belirtilmektedir.

Unit testler nodejs kullanılarak uygulama kök dizininde "npm test" komutuyla çalıştırılır. Örnek bir test çıktısı aşağıdaki gibidir:

INFO [watcher]: Changed file "zetaops/ng-zaerp/app/login/login_test.js". Chrome 45.0.2412 (Mac OS X 10.10.3): Executed 8 of 8 SUCCESS (0.409 secs / 0.063 secs)

Bu çıktıdan 8 test senaryosunun başarıyla geçtiği görülmektedir (Executed 8 of 8 SUCCESS (0.409 secs / 0.063 secs)).

Birim testlerinin kodun ne kadarını kapsadığı yine karma ile incelenecektir. Karma testler çalıştıktan sonra coverage/ dizini altında bir html dosyası oluşturarak kod kapsama oranını yayınlar. Örnek html çıktı sayfası şu şekildedir:



Code coverage report for **All files**

Statements: **61.34%** (73 / 119) Branches: **6.25%** (1 / 16) Functions: **63.83%** (30 / 47) Lines: **61.34%** (73 / 119) Ignored: none

File	Statements		Branches		Functions		Lines	
components/version/	100%	(28 / 28)	100%	(0 / 0)	100%	(16 / 16)	100%	(28 / 28)
dashboard/	50%	(1 / 2)	100%	(0 / 0)	0%	(0 / 2)	50%	(1 / 2)
login/	49.44%	(44 / 89)	6.25%	(1 / 16)	48.28%	(14 / 29)	49.44%	(44 / 89)

Generated by istanbul at Tue May 26 2015 16:46:07 GMT+0300 (EEST)

Kabul Testleri

Kabul testleri e2e-tests dizini altındaki "scenarios.js" dosyasına yazılır. Testler Jasmine test uygulama çatısı ile yazılacaktır. Örnek bir test senaryosu aşağıdaki gibidir:

```
describe('dashboard', function () {
        beforeEach(function () {
                browser.get('index.html#/dashboard');
        });
        it('should redirect to login if not logged in', function (){
                        expect(element.all(by.css('[ng-view] h1')).first().getText()).
                            toMatch(/Zaerp Login Form/);
                });
});
```

Yukarıdaki örnekte tarayıcı uygulamanın "dashboard" sayfasını çağırmakta eğer giriş yapılmamışsa "login" sayfasına yönlendirmesi beklenmektedir.

Bu testler Protractor ile çalıştırılır. Protractor, Selenium web-driver'larını angularJS ile kullanmak için özelleştirmeler barındıran bir çözümdür. Örnek yapılandırma dosyası aşağıdaki gibidir:

```
exports.config = {
                allScriptsTimeout: 11000,
                specs: [
                        '*.js'
                ],
                capabilities: {
                        'browserName': 'chrome'
                },
                baseUrl: 'http://localhost:8000/',
                framework: 'jasmine',
                jasmineNodeOpts: {
                defaultTimeoutInterval: 30000
        }
};
```

Bu yapılandırma dosyasında testlerin çalıştırılacağı tarayıcı (browserName), url (baseUrl), uygulama çatısı (framework) timeout süreleri gibi özellikler yapılandırılır. Kabul testleri kök dizinde "protractor e2e-tests/protractor.conf.js" komutu ile çalıştırılır.

Manuel Testler

Tarayıcılara has hatalar, görsel düzenlemeler ve diğer otomatik olarak test edilemeyen arayüz özellikleri ve fonksiyonları manuel olarak test edilecektir.

1.1.3 Sürüm Öncesi Kabul Testleri

Test Senaryoları

Ihtiyaç analiz belgelerinde belirtilen kullanıcı senaryolarına uygun şekilde test senaryoları yazılacaktır. Test senaryolarının amacı, ihtiyaç analizinde ortaya çıkan gereksinimlerin, geliştirme faaliyeti sonucu ortaya çıkan ürün ile karşılanıp karşılanmadığıdır.

Sürüm aşamasında önceden yazılmış test senaryoları, kullanıcılar tarafından manuel şekilde uygulanır ve sonuçlar raporlanır. Bazı test senaryoları otomatik olarak da gerçeklenebilirler.

Bizim uygulamamızda test senaryoları bir veya birden çok iş akışından (workflow) oluşan eylemler dizisi şeklinde olacaktır. Birden çok aktör ve ön koşulu içinde barındıran, problemi yeterli karmaşıklık düzeyine getirecek belirli sayıdaki öğrencinin ders seçimi ve sonuçlarının web sitelerinde yayınlanması veya öğrenciler için ders programının hazırlanması gibi..

Kullanılabilirlik Testi

Uygulama ekranları, uluslararası kabul görmüş kullanılabilirlik ilkeleri (w3c) ve KAMİS (Kamu Internet Siteleri Rehberi)'nin temel aldığı TS EN ISO 9241-151 (İnsan-Sistem Etkileşiminin Ergonomisi Standartları), WCAG ve ISO/IEC 40500:2012 (Web İçeriği Kullanılabilirlik Standartları ve Kriteri) standartlarında uygunluğu test edilecektir.

Bu amaçla genel bir kontrol listesi (checklist) hazırlanmıştır:

Genel Görünüm

- Klavye kısayollarıyla gezinmek mümkün mü?
- Klavye kısayollarıyla gezinmek kolay mı?
- Sayfalar otomatik olarak yenilenmemeli
- Website iletişim bilgileri, referansları uygun bir alanda mı?
- Servis/hizmet/uygulama bilgilerine kolayca erişiliyor mu?
- Görme engelliler için erişilebilirlik düzenlenmiş mi?
- Grid sistem kullanılmış mı?
- Klavye kullanımı sitedeki tüm işlemleri kapsıyor mu?
- Kullanıcılara içerikleri okuyabilmeleri için yeterli zaman verililyor mu?
- Hukuki ya da mali sonuçları olan işlemlerde kullanıcının hata yapma olasılığı azaltılmalıdır.

Anasayfa

- Amacı kolay anlatıyor mu?
- Yapmak istediği işleme kolay ulaşılıyor mu?
- Sayfa görünümü pozitif bir intiba bırakıyor mu?
- Giriş yapan kullanıcı ismi yer alıyor mu?
- Büyük değişiklikler ana sayfadan duyuruluyor mu?

- Konum ve iletişim bilgileri yer alıyor mu?
- Lisans, sözleşme gibi statik sayfalara linkler var mı?
- Sayfadaki imajlar ve/veya videolar amaçla alakalı mı?
- Site hem www alt alanadıyla hem alt alanadı olmadan erişilebilir mi?
- Sitede yapılacak temel işlemler ana sayfada yer alıyor mu?

Yönetim Paneli

- İçerikler kullanıcı rolüyle ilgili mi?
- Uyarılar zamanında ve etkili şekilde gösteriliyor mu?
- Uyarılar öncelik ve önem derecelerine göre renklendirilmiş mi?
- Birden fazla role sahip kullanıcılar için roller arası geçişi sağlayan bir buton var mı?

Erişilebilirlik

- İmajların "alt" özellikleri kullanılmış mı?
- İçerik stil dosyası (css) olmadan da okunabilir mi?
- Bağlantılar, butonlar ve seçim kutuları kolayca tıklanabilir mi?
- örnek erişilebilirlik testi: http://achecker.ca/checker/index.php

Site İçi Yönlendirme

- Önemli bağlantılar sayfanın hareketli öğelerinde olmamalı
- Linkler alfabetik olarak sıralanmamalı, gruplanmalı
- Kullanıcı sitede hangi sayfada olduğunu kolayca farkedebilmeli
- Yönlendirme bağlantıları her sayfada görünür mü?
- Bağlantılar açıklayıcı mı?
- Title'da site ve o sayfanın kısa bir açıklaması var mı?
- Site url'si akılda kalıcı mı?

Arama

- Bir arama kutusu var mı?
- Arama kutusu her sayfada görünür mü?
- Arama kutusu yeterince geniş mi?
- Arama sonuçları kategorilendiriliyor mu?

Bağlantılar

- Önemli komutlar bağlantı yerine buton olarak gösterilmeli, örn: kaydet gibi
- Linkler kolayca farkedilir mi?
- Kırık (erişilemeyen) link olmamalı

Şablon

- Öğünemli içerikler öncelikli olarak gösteriliyor mu?
- Site şablonu farklı ekran boyutlarında ölçekleniyor mu?
- Birbiriyle alakalı bilgiler gruplandırılmış mı?

- Tüm sayfalarda tutarlı mı?
- Sayfalar çok sıkışık olmamalı

Formlar

- Formlar kolay doldurulabilir mi?
- Form alanlarının açıklamaları var mı?
- Alanların alması gereken değerler kullanıcıya gösteriliyor mu?
- Çok uzun açılır menüden kaçınılmış mı?
- Form alanlarının isimleri açık ve anlaşılır mı?
- Form onay butonu var mı?
- Hata mesajları ilgili form alanının yanında yer alıyor mu?
- Birden fazla adımdan oluşan formlar için hangi adımda olduğu anlaşılıyor mu?

İçerik

- Metin ve arkaplan rengi arasında yeterli derecede kontrast var mı?
- İçerik gözle taranabiliyor mu?
- İçerik temiz bir dille yazılmış mı?
- İletişim bilgileri açık şekilde yazılmış mı?
- İçerik kullanışlı ve güncel mi?
- Dil kurallarına uyuyor mu?
- İçerik sıralaması anlamlı mı?
- İçeriklerin ayırt edilebilmesi ya da doğru anlaşılabilmesi için renk kullanımına dikkat edilmiş mi?
- Hareketli içerikler kullanıcılar tarafından kontrol edilebiliyor mu?
- Tekrarlı içerikler pas geçilebiliyor mu?
- Metin öğeleri yeniden boyutlandırılabilir mi?

Her ekran kontrol listesi formu ile birlikte açılır. Test kullanıcıları bu formu doldurup kaydederler. Sonuçlar ilgili servise raporlanır.

Performans Testleri

Load Tests

Yük testleri, uygulamanın belirli parçalarının yoğun trafik altındaki davranışlarını ölçmek amacıyla yapılır. Yayınlama aşamasında önceden belirlenen yük değerleri ile otomatik şekilde gerçekleştirilecektir. Bu amaçla geçici sanal makineler oluşturulacak, testler bu makineler üzerinde gerçekleştirilecektir. Temel test aracımız Tsung [4] olarak seçilmiştir. Tsung birçok farklı protokolde detaylı şekilde özelleştirilebilen requestler hazırlamaya olanak vermektedir.

[4] http://tsung.erlang-projects.org/

Ağ Kullanımı ve Web Sayfa Başarımı

Ağ Kullanımı uygulama modüllerinin gerektiğinde çağırılacak şekilde düzenlenmesi (lazy load), statik dosyaların (javascript, css, imaj ve diğer dosyalar) optimize ve minimize edilmesi gibi konuları içerir. Bu süre çevrimiçi araçlar ve tarayıcılar kullanılarak test edilir.

Render Performansı

Sayfa render süresi kod tekrarı, optimizasyonu, DOM kullanımı gibi bilinen gerekliliklere göre kısalmaktadır. Sayfa bileşenlerinin yüklenme süresinden sonra gereken tüm fonksiyonların çalıştırılması ve stillerin uygulanması süresi render performansıdır. Tarayıcının yeteneklerine bağımlı olsa da belirlenecek minimum değerin altında olmamalıdır. Selenium ile test edilecektir.

Güvenlik Testleri

Uygulamanın güvenlik test ve kontrolleri için Open Web Application Security Project (OWASP) [5] topluluğunun yayınladığı test rehberinde [6] yer alan testlerin bazıları kullanılacaktır. 11 konu başlığı altında toplanan testlerin uygulamamız için uygun olanları seçilerek her sürüm öncesi kabul testleri aşamasında uygulanacaktır.

Kontroller Sistem Hakkında Bilgi Toplama, Yapılandırma ve Yayınlama, Kimlik Yönetimi, Kimlik Doğrulama ve Yetkilendirme, Oturum Yönetimi, Girdi Geçerliliği, Hata Ayıklama, Şifreleme, İş Mantığı, İstemci Tarafı Testleri başlıkları altında yapılacaktır.

Yapılacak testler ayrıca ISO 27002 bilgi güvenliği standartlarında belirlenen kriterlerin tamamlanması için kuruma destek olacaktır.

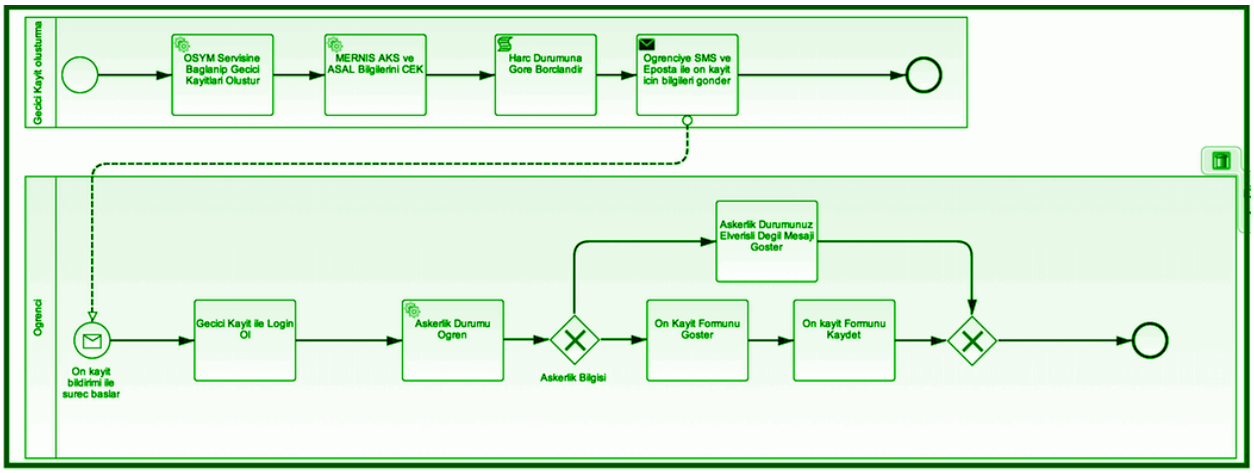
1.2 Yazılım Tasarım Analizi Belgesi

1.2.1 Sistem Mimarisi

İş Akışı (Workflow) Tabanlı Uyguluma

İş Akışı (Workflow) Tabanlı Uygulumalar, iş süreçlerini yeterince küçük adımlara bölerek, iş akışlarının aktörlerini ve her bir adımda işlenecek veriyi net bir şekilde tanımlayarak kolay yönetilebilirlik ve esneklik sağlarlar. İş süreçlerinin tamamıyla olmasa bile belli oranda otomatikleştirilmesi ve yazılım tarafından işlenebilir olması değişen ihtiyaçları karşılamak için kolaylık sağlamaktadır. Sadece iş akışı şemalarında yapılacak değişikliklerle uygulamaya yön vermek mümkündür.

[5] https://www.owasp.org/
[6] https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf



Örnek İş Akış Şeması

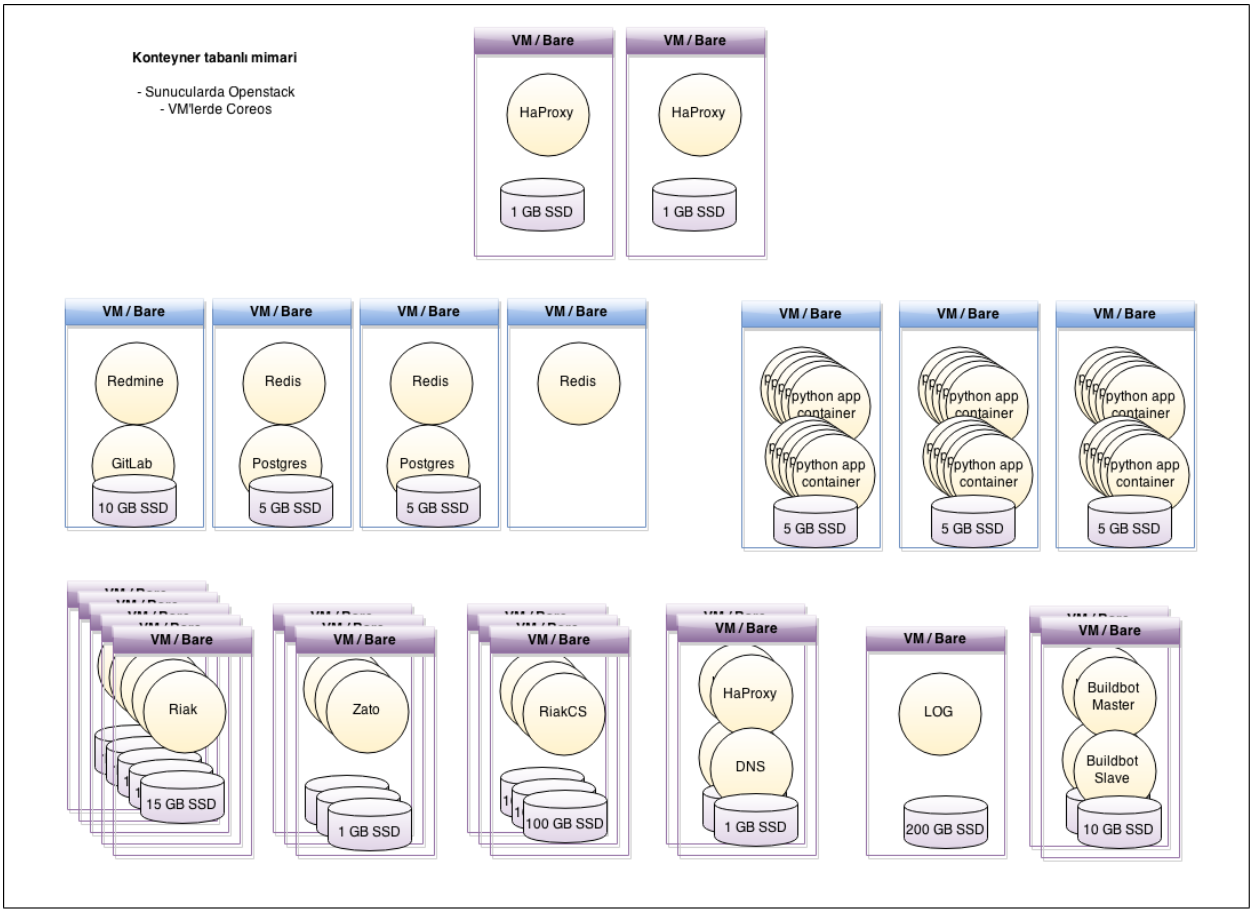
İş süreçleri yönetimi için, BPMN2 kural setine uyan Workflow Management yapısı seçilmiştir. Python dünyasında WorkFlow Management ve BPMN desteği bulunan SpiffWorkFlow kütüphanesi geliştirilmeye açık en uygun aday olduğu için seçilmiştir.

İş süreçlerinin görsel olarak oluşturulması için de Camunda Modeler seçilmiştir.

Kurumsal Veri Yolu Kullanımı

Kurumsal Veri Yolu (ESB) uygulamanın parçaları ve dış servisler arası iletişimi düzenler. Veri kaynaklarını mikro ölçekte, yeniden kullanılabilir servisler haline getirerek bir iletişim düzeni kurar. Özellikle dış servislerle veri alışverişini kendi üzerine alarak uygulama içerisinde daha sade ve yönetilebilir kod yazmaya olanak verir. Neden KVY gerektiği ile ilgili, tarafımızdan çevrilmiş bulunan, açıklayıcı ve detaylı yazıyı verilen bağlantıda [7]_ okuyabilirsiniz.

1.2.2 Sunucu ve Servis Yerleşimi



VM Konfigürasyonları:

Small: Standart: Large:	2 GB RAM, 4 GB RAM, 8 GB RAM,	2 VCPU 4 VCPU 6 VCPU	2 GB Ephemeral SSD 10 GB SSD

Açıklama	VM Türü	Adet	DİSK	OS
Riak Nodes	Large	5	400 GB SSD	Coreos
RiakCS	Standart	3	400 GB SSD	Coreos
Redis	Large	3	20 GB SSD	Coreos
Zato	Standart	3		Coreos
HaProxy	Standart	2	1 GB NonSSD	Coreos
HaProxy	Large	2	10 GB SSD	Coreos
Logging	Standart	2	400 GB NonSSD	Coreos
Apps	Small	3		Coreos
Buildbot	Small	4	10 GB SSD	Coreos
Fab	Standart	1	40 GB SSD	Ubuntu
Sandbox	Standart	1	80 GB NonSSD	Ubuntu

Image özellikleri:

Coreos [8]_ Ubuntu [9]_

1.2.3 Tercih Edilen Yazılımlar

Riak [10]_

Riak, dağıtık bir veritabanı sistemidir. Verileri sunucular arasında dağıtarak yüksek erişilebilirliği sağlar. Riak'ın tercih edilmesindeki sebepler aşağıda sıralanmıştır:

- Çok kullanıcılı / çok rollü sistemde verilerin sürekli erişilebilir olması,
- Aynı veriye eş zamanlı erişerek okuma / yazma işlemlerinin, veriye erişimi bloke etmeyerek gerçekleştirilmesi,
- İş modeline uygun senaryolara göre ortaya çıkacak uyuşmazlıkların çözümlenebilmesine olanak vermesi,
- Zamanla büyüyen verinin sürekli yedekli olarak yönetilmesi,
- Veri tasarım şeklinin, RDBMS kalıpları dışında yapılabilmesi,
- Verinin "strong consistent" olarak tutulabilmesi sayesinde "ACID" benzeri bir tutarlılığa sahip olmak,
- Sistem tüm Türkiye çapında kullanılsa bile, değişik veri merkezlerinde "multi homed" olarak çalışabilir olması.

RiakCS [11]_

RiakCS, Riak veritabanı sisteminin üzerine kurulu nesne depolama (object storage) sistemidir. Sistemde kullanıcı tarafından üretilecek dokumanlar (resimler, doc, pdf vb) RiakCS ile saklanacaktır. RiakCS de dağıtık bir sistemdir. RiakCS ile yüksek erişilebilir, ölçeklenebilirlik dağıtık bir bulut depolama sistemi elde edilmiş olacaktır. RiakCS API (Uygulama Programlama Arayüzü) yaygın kullanılan Amazon S3 ile uyumludur.

Postgresql

Postgresql, Zato Clusterı tarafından kullanılmaktadır. Cluster bilgileri gibi Zato'nun iç operasyonlarını ilgilendiren veriler saklanacaktır.

Redis [12]_

Redis, bellekiçi key/value veritabanı sistemidir. Uygulama ile Riak arasında geçici depolama ve hızlı okuma işlemleri için kullanılacaktır. Bu sayede veritabanı üzerindeki yük hafifleyecek, veriye erişim çok yüksek hızlara çıkıp uygulama performası artacaktır.

Redis, ayrıca Zato tarafından benzer amaçlar için de kullanılacaktır.

Zato [13]_

Zato Python ile geliştirilmiş Kurumsal Hizmet Veriyolu (Enterprise Service Bus) yazılımıdır. Zato ile iş akışlarına uygun olarak, uygulamalar arası veri trafiği, mikro servisler haline getirilerek düzenlenecektir. Zato sadece sistem içi operasyonlar için değil aynı zamanda dış kaynaklarla olan iletişimi de üzerine alıp onları sistemin içerden erişebileceği mikro servislere dönüştürecektir. Bu da dış dünya ile uygulamanın tıpkı içerdeki gibi benzer desenler ile konuşabilmesini sağlayarak tutarlılık sağlayacaktır.

HaProxy [14]_

High Available (Yüksek Erişilebilirlik) Proxy, hem kullanıcı arayüzeyleri aracılığı ile gelecek istekler (requests), hem dışarıya açılan servislere yapılacak çağrılar, hem de sistem içi bileşenlerin birbirleri ile olan iletişimleri sonucu doğacak trafiği dengelemek ve yüksek erişilebilirliği sağlamak için kullanılacaktır.

1.2.4 Bulut araçları

Docker [15]_

Docker uygulama ve servislerin konteynerlar şeklinde sanallaştırılarak Linux sistemleri üzerinde çalıştırılmasını sağlar. Docker uygulama ve servislerin yönetimini ve ölçeklenmesini kolaylaştırır. Bütün bileşenler kontenyerlar içinde servisler şeklinde çalışacaktır. Uygulama ve diğer tüm bileşenler bu sayede ihtiyaçlar ölçüsünde kolayca ölçeklenebilecektir.

Consul [16]_

Servislerin ve üzerlerinde çalıştıkları sistemlerin erişilebilirliği, yeni açılan veya herhangi bir sebeple çalışması kesintiye uğrayan, kapanan servislerden haberdar olmak için bütün host sistemlerde çalışacak servistir.

Systemd

Systemd linux sistemler için neredeyse standart hale gelmiş modern servis yonetim aracıdır. Konteynerlar haline gelen uygulama parçacıkları systemd servisleri şeklinde yönetilecektir.

Etcd

Etcd bir sytemd servisi olarak çalışacak ve cluster çapında data alışverişi yapmak için kullanılacaktır. Ortam değişlenleri, değişen ayarlar, Consul ve benzeri servislerin haberleşmesi için kullanılacaktır.

Confd

Confd başta haproxy gelmek üzere sistem servislerinin yeni durumlarına göre yeni ayar dosyaları üretme ve ilgili servisleri yeniden başlatma işini üstlenmektedir.

Flannel

Flannel cluster içinde çalışan servisler (docker konteynerları) için özel bir ağ katmanı oluşturur. Bu sayede servisler bu özel ağ üzerinden birbirleri ile konuşabilirler. Fleet * Fleet, konteyner haline getirilen servislerin cluster çapında systemd ye bildirilmesi ve yönetilmesinden sorumludur. Fleet yazılan bir servis için hazırlanan tanımlama dosyası (unit files) gereklerine uygun olarak, uygun gördüğü makinelerde çalıştırmaktan, bir başka makineye taşımaktan veya durdurmaktan sorumludur. Github ** Github [17]_ temel proje yönetim ve geliştirme alanımızdır. Birçok geliştiricinin alışık olduğu bu ortam, katkıcıların kolayca dahil olmalarına olanak vermektedir. Açık kaynaklı yazılım projeleri geliştirme teamüllerine uygun bir ortamdır. Git sürüm yönetim sistemini kullanmaktadır. Geliştirici ve kullanıcı topluluğun teknik tartışmaları, geri bildirimleri Github'ın sağladığı ilgii araçlarla yapılacaktır.

Continuous Integration & Continuous Delivery

Uygulama kaynak kodu ve/veya sistem/ortam ayarları değişiklikleri üzerine, uygulamanın test edilmesi, belirlenen ortamlarda kurulum ve yayınlanma işlerinin otomatik şekilde yapılması, elde edilen sonuçların geliştiricilerle paylaşılması ve raporlanması, geliştirme süreçlerini kolaylaştırmakta, hızlandırmakta, problemlerin kaynaklarını tespit etmeye yardımcı olmaktadır.

Projede, bu amaçla Buildbot [18]_ kullanılacaktır. Buildbot ile üretilen her türlü sonuç, log, rapor projede ilgili taraflara çeşitli kanallardan iletilecektir.

1.2.5 Logging

Kayıt Türleri

DEBUG: Geliştici ve sistem yöneticileri için, servis veya uygulamaların çalışmaları hakkında açıklayıcı bilgiler sunar. Bu bilgiler geliştirme evresinde ve sorun çözme aşamalarında kullanılır.

INFO: Servis veya uygulamaların önemli adımlarının sonuçlarına, durum değişikliklerine ilişkin detaylı bilgiler içerir. Kullanıcı giriş yaptı, yeni ders eklendi, servis yeniden başladı vb..

WARN: Servis veya uygulamaların beklenen dışında davranışlar göstermesi hakkında bilgiler içeren kayıtlardır. Hata olmamakla birlikte bir servise erişememek, diske yazamamak gibi geçici problemlerin sebep olduğu aksaklıkların bildirilmesini kapsar. Uygulama veya servis kesintiye uğramaz fakat nasıl yönlendirildiğine bağlı olarak bir süre sonra yeniden deneyebilir, raporlayabilir, başka bir yöne doğru ilerleyebilir.

ERROR: Uygulamanın bir adımında beklenen işlevi yerine getirememesi sonucu ortaya çıkan kayıtlardır. Servis veya uygulama kesintiye uğramaz fakat ilgili adım muhtemelen elle müdahale gerektirecek bir problemle karşı karşıyadır.

FATAL: Uygulama yada servisin, veri kaybına da neden olabilecek bir hizmet kesintisine uğraması durumunda tutulan kayıtlardır.

Log Yönetimi

Hem uygulama hem de uygulamanın çalışacağı ortam bileşenlerinin her birinden toplanacak loglar, merkezi bir loglama sisteminde toplanacaktır. Sistemin anlık olarak izlenmesi, olağandışı gelişmelere uygun aksiyonlar alınması, uzun vadede geliştirme süreçlerine geribildirim olarak dönmesi amacıyla toplanan kayıtlar analiz edilecektir.

Bu amaçla Logstash [19]_ , Kibana [20]_, Elasticsearch [21]_ üçlüsü kullanılacaktır. Logstash ve Elasticsearch logların toplanması, filtrelenmesi, analiz edilmesi, Kibana ise görselleştirilmesi için kullanılacaktır.

Sistem ve Servis Logları

Uygulamanın üzerinde çalışacağı donanım, işletim sistemi, cluster ve bunlar üzerinde çalışacak servisler hakkında şu loglar tutulacaktır:

- Coreos ve sanallaştırma araçlarından elde edilen loglar,
- Sisteme yapılan girişler,
- Açılan kapanan konteynerların durumları hakkındaki loglar,
- Konteyner haline gelmiş servislerden
- Load Balancer erişim, hata, health check logları
- Riak ve RiakCS cluster yönetimi, riak admin logları
- Riak ve RiakCS kimlik dogrulama ve yetkilendirme logları

- Zato servis hata logları
- Zato iç ve dış servisler için doğrulama ve yetkilendirme logları

Sistem ve Servis Log Analizi

- Çalışması duran servislerin tespit edilmesi ve aksiyon alınması,
- Servislerden gelen hata loglarının, bellek durumu, cpu yükleri, disk doluluğu, network problemleri gibi donanım ve ağ logları ile birleştirilerek, aralarında bir kolerasyon olup olmadığının anlaşılması,
- Servislerin durmasından hemen önceki işlemlerin tür ve yoğunluğunun önceki servis durmaları ile ortaklık gösterip göstermediğine bakılarak, örneğin disk i/o, vtye belirli sayıların üzerinde yazma vb gibi işlemlerin servislere olan olumsuz etkilerin ve buna yol açan sebeplerin anlaşılması,
- Clustered servislerden gelen loglara bakarak yük dağıtımının dengeli bir şekilde yapılıp yapılmadığının anlaşılması,
- Consul ile birlikte monitoringe yardımcı olması

Kullanıcı Arayüzü Logları

Kullanıcı arayüzünde oluşacak çalışma zamanı hataları tarayıcı konsoluna düşmektedir. Bu loglar yakalanarak sunucu tarafındaki log tutucuya gönderilerek kaydedilecektir.

Arayüz fonksiyonları logları belirtilen log seviyelerinde tutulacaktır..

Prod başlığında belirtilen maddeler ışığında arayüz logları için stacktrace.js kullanılacaktır.

incele:

http://logstash.net/docs/1.1.1/outputs/riak#setting_bucket

http://underthehood.meltwater.com/blog/2015/04/14/riak-elasticsearch-and-numad-walk-into-a-red-hat/

Notlar:

CEP için loglarla nasıl bir relation kuracağız? Loglardan event trigger etmek nasıl?

Refleksler:

- Duran servisleri yeniden başlatmak
- Ölenlerin yerine yenisini başlatmak
- Ağır yük altında olan servisleri genişletmek
- Hafif yük altında olan servisleri daraltmak
- Ölçeklenecek serviler için sistem kaynaklarının yetersizliğini tespit edip yeni kaynaklar eklemek veya kaynak ihtiyacını bildirmek. Mümkünse clustera yeni nodelar

otomatik eklemek.

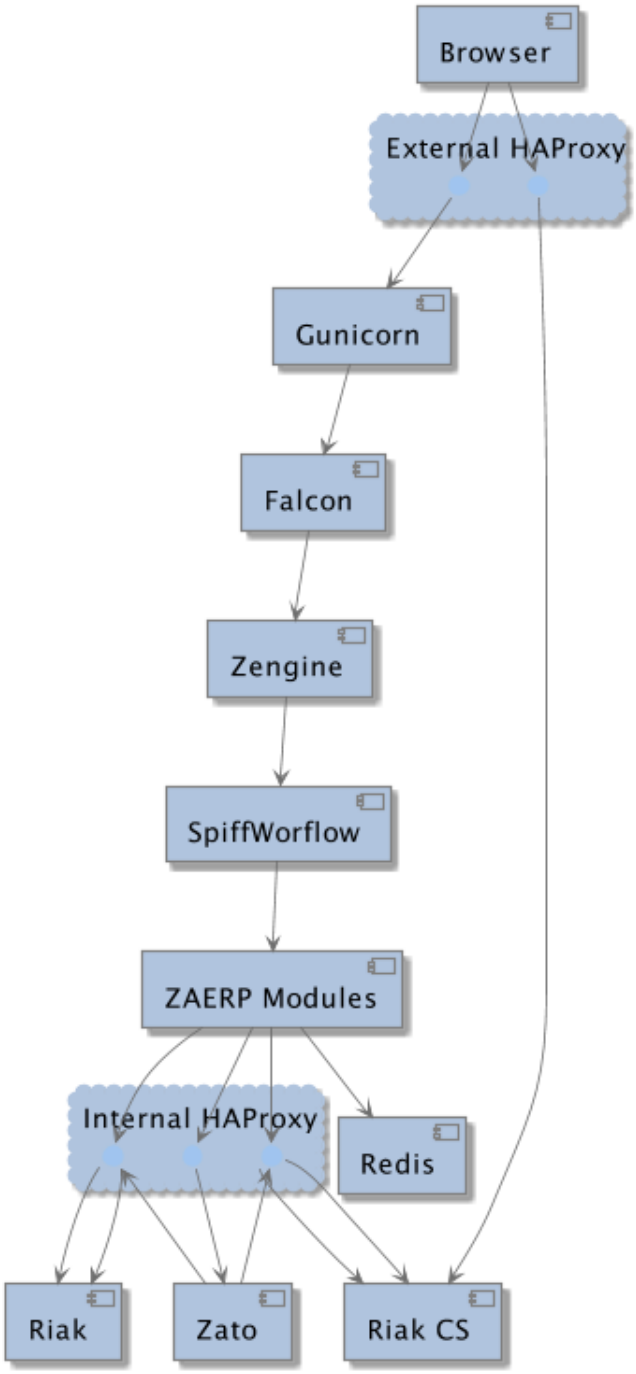
- Kronik hale gelen problemlerin tespiti ve bilgilendirilmesi. Muhtemel konfigurasyon problemleri demek.
- Application loglarindan gelen uyarilar

Fleet API kullanarak clusterda tanımlı servisleri başlatmak / durdurmak mümkün. Node ekleyip çıkarmak için Openstack / GCE API ile konuşmamız gerekir. Notification eposta veya sms ile mümkün. Yukarıdakilere ek başka ne gibi aksiyonlar olabilir?

1.2.6 Tercih Edilen Yazılım Bileşenleri

Arka Uçta Kullanılan Bileşenler

Genel Sistem Akış Şeması



Modül / Bileşenlerin Genel Görünümü

- zaerp

    -zdispatch

requestleri karsilayip ilgili is akislarina yonlendiren falcon web çatısı dosyalari yer alacaktir

    -bin

çalıştırılabilir uygulamalar. örn: bpmn packager.

    -lib

yardımcı kütüphane ve fonksiyon setleri

    -modules

bazıları kendi alt dizinlerine sahip olan uygulama modulleri.

    -auth

örnek authentication modülü

        -models %user.py
        %auth.py
        %employee.py
        %unit.py

    -services

bu dizinde Zato mikro servis dosyaları yer alacaktır.

    -workflows

bu dizinde iş akışı paketleri bpmn dosyaları yer alacaktır

- tests

methodlar, uygulama birimleri ve uygulama geneli icin yazilan unit testleri yer alacaktır

- docs

    -geliştiriciler

        %diagrams

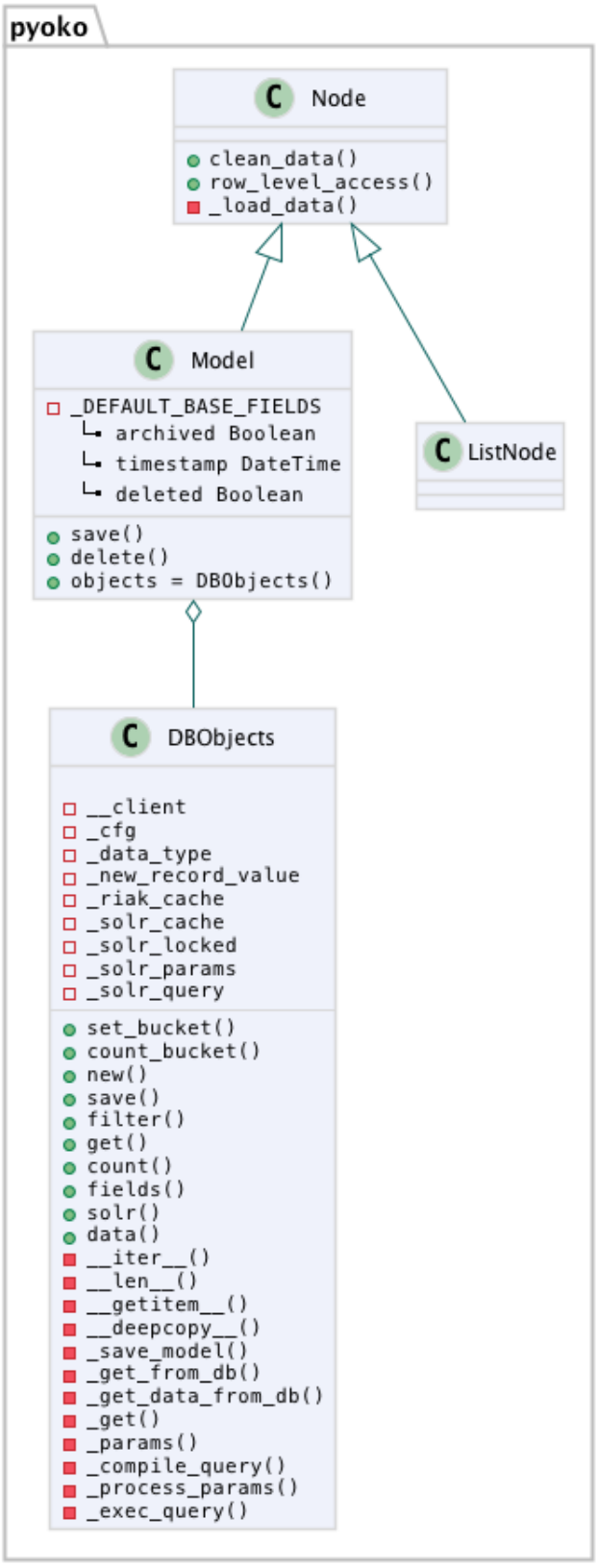
        %api

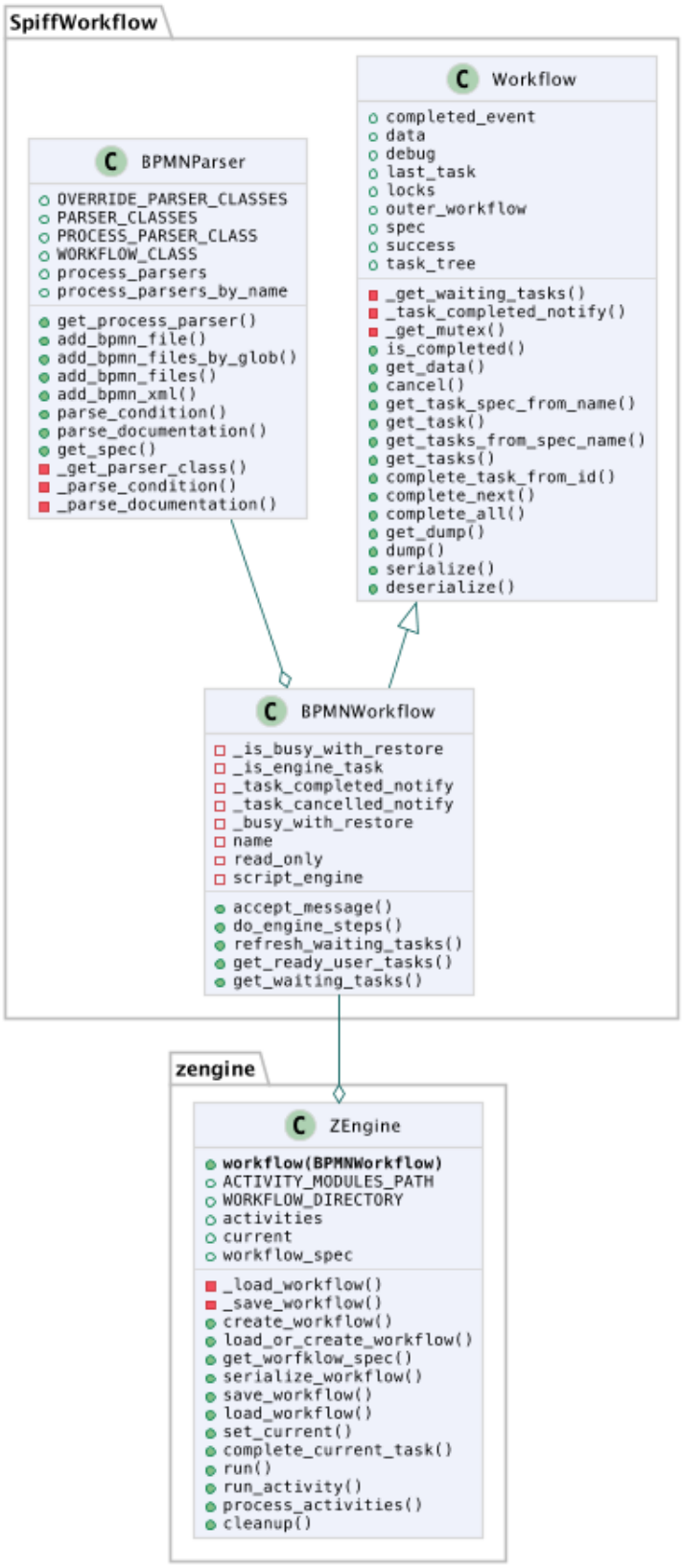
    -son kullanıcılar

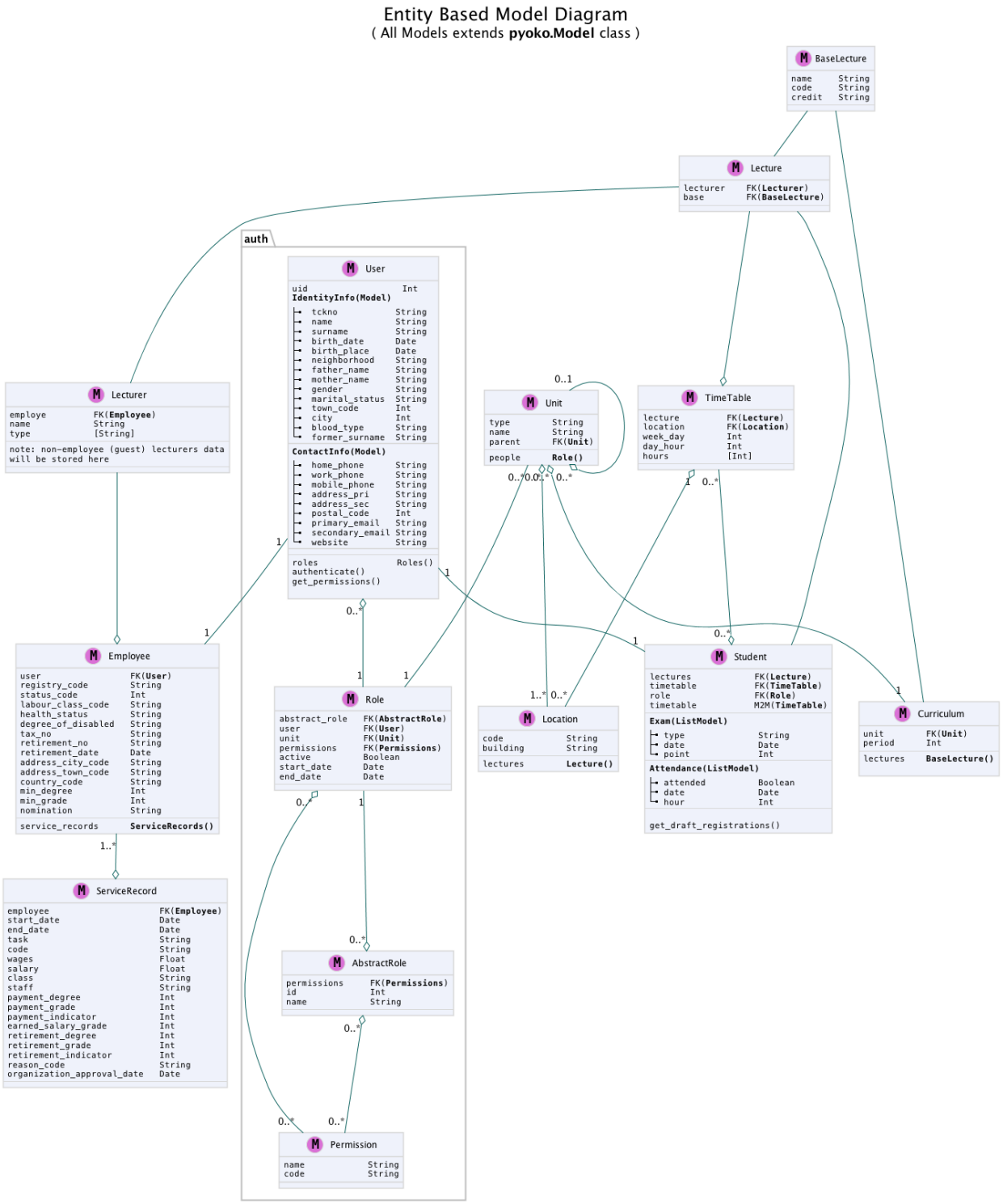
    -sistem yöneticileri

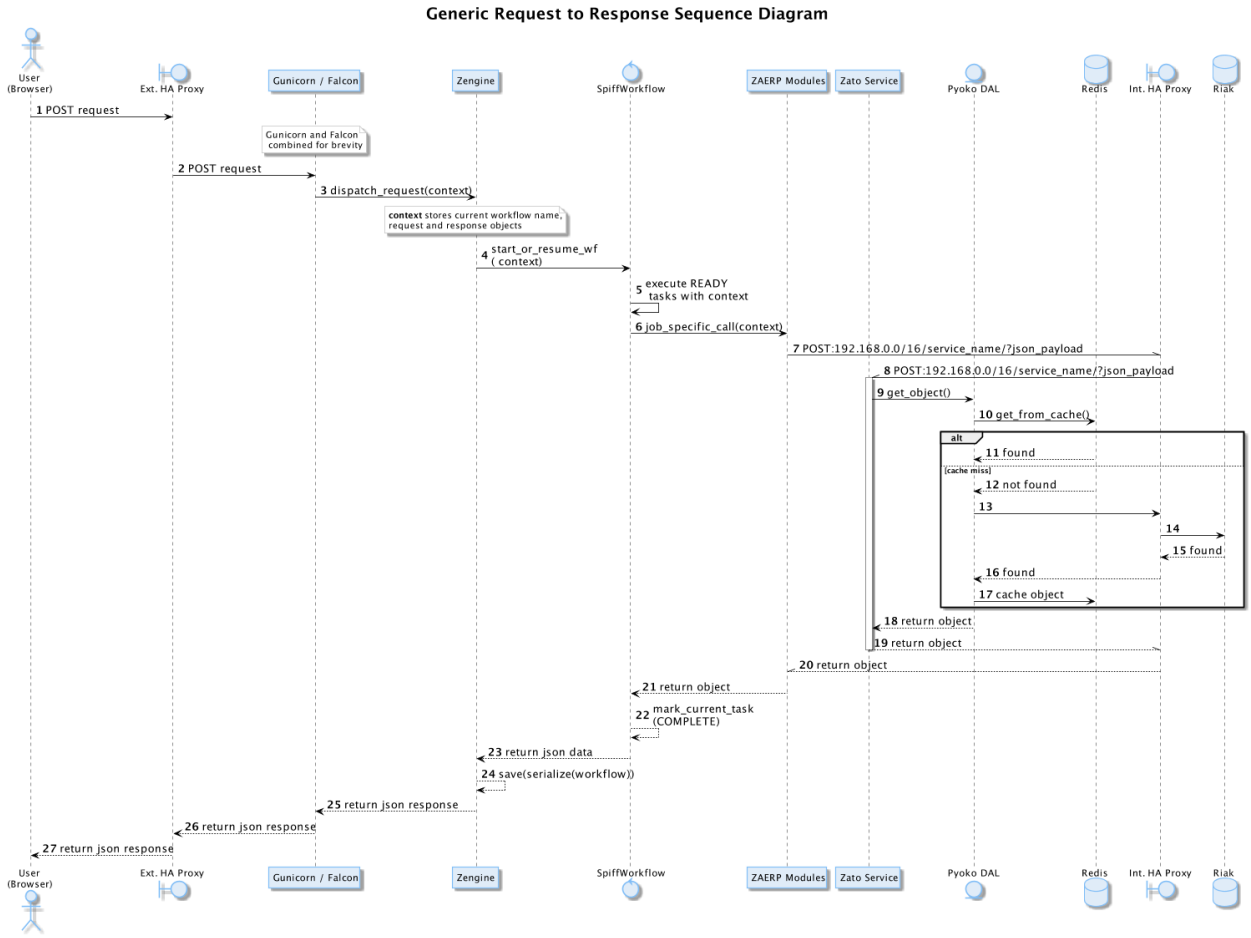
kod, api, kullanici, gelistirici, sistem yoneticisi dokumanlari yer alacaktir.

Uygulamanın veri ve iş mantığının şu ana kadar planlanan yapısını gösteren class diagramlar aşağıda görülebilir.


pyoko
Node
clean_data()
row_level_access()
_load_data()
Model
_DEFAULT_BASE_FIELDS
archived Boolean
timestamp DateTime
deleted Boolean
save()
delete()
objects = DBObjects()
ListNode
DBObjects
__client
_cfg
_data_type
_new_record_value
_riak_cache
_solr_cache
_solr_locked
_solr_params
_solr_query
set_bucket()
count_bucket()
new()
save()
filter()
get()
count()
fields()
solr()
data()
__iter__()
__len__()
__getitem__()
__deepcopy__()
_save_model()
_get_from_db()
_get_data_from_db()
_get()
_params()
_compile_query()
_process_params()
_exec_query()


SpiffWorkflow
Workflow
completed_event
data
debug
last_task
locks
outer_workflow
spec
success
task_tree
_get_waiting_tasks()
_task_completed_notify()
_get_mutex()
is_completed()
get_data()
cancel()
get_task_spec_from_name()
get_task()
get_tasks_from_spec_name()
get_tasks()
complete_task_from_id()
complete_next()
complete_all()
get_dump()
dump()
serialize()
deserialize()
BPMNParser
OVERRIDE_PARSER_CLASSES
PARSER_CLASSES
PROCESS_PARSER_CLASS
WORKFLOW_CLASS
process_parsers
process_parsers_by_name
get_process_parser()
add_bpmn_file()
add_bpmn_files_by_glob()
add_bpmn_files()
add_bpmn_xml()
parse_condition()
parse_documentation()
get_spec()
_get_parser_class()
_parse_condition()
_parse_documentation()
BPMNWorkflow
_is_busy_with_restore
_is_engine_task
_task_completed_notify
_task_cancelled_notify
_busy_with_restore
name
read_only
script_engine
accept_message()
do_engine_steps()
refresh_waiting_tasks()
get_ready_user_tasks()
get_waiting_tasks()
zengine
ZEngine
workflow(BPMNWorkflow)
ACTIVITY_MODULES_PATH
WORKFLOW_DIRECTORY
activities
current
workflow_spec
_load_workflow()
_save_workflow()
create_workflow()
load_or_create_workflow()
get_worfklow_spec()
serialize_workflow()
save_workflow()
load_workflow()
set_current()
complete_current_task()
run()
run_activity()
process_activities()
cleanup()


Entity Based Model Diagram
(All Models extends pyoko.Model class)
BaseLecture
name String
code String
credit String
Lecture
lecturer FK(Lecturer)
base FK(BaseLecture)
auth
User
uid Int
IdentityInfo(Model)
tckno String
name String
surname String
birth_date Date
birth_place Date
neighborhood String
father_name String
mother_name String
gender String
marital_status String
town_code Int
city Int
blood_type String
former_surname String
ContactInfo(Model)
home_phone String
work_phone String
mobile_phone String
address_pri String
address_sec String
postal_code Int
primary_email String
secondary_email String
website String
roles Roles()
authenticate()
get_permissions()
Lecturer
employe FK(Employee)
name String
type [String]
note: non-employee (guest) lecturers data will be stored here
Unit
type String
name String
parent FK(Unit)
people Role()
TimeTable
lecture FK(Lecture)
location FK(Location)
week_day Int
day_hour Int
hours [Int]
Employee
user FK(User)
registry_code String
status_code Int
labour_class_code String
health_status String
degree_of_disabled String
tax_no String
retirement_no String
retirement_date Date
address_city_code String
address_town_code String
country_code String
min_degree Int
min_grade Int
nomination String
service_records ServiceRecords()
Role
abstract_role FK(AbstractRole)
user FK(User)
unit FK(Unit)
permissions FK(Permissions)
active Boolean
start_date Date
end_date Date
Location
code String
building String
lectures Lecture()
Student
lectures FK(Lecture)
timetable FK(TimeTable)
role FK(Role)
timetable M2M(TimeTable)
Exam(ListModel)
type String
date Date
point Int
Attendance(ListModel)
attended Boolean
date Date
hour Int
get_draft_registrations()
Curriculum
unit FK(Unit)
period Int
lectures BaseLecture()
ServiceRecord
employee FK(Employee)
start_date Date
end_date Date
task String
code String
wages Float
salary Float
class String
staff String
payment_degree Int
payment_grade Int
payment_indicator Int
earned_salary_grade Int
retirement_degree Int
retirement_grade Int
retirement_indicator Int
reason_code String
organization_approval_date Date
AbstractRole
permissions FK(Permissions)
id Int
name String
Permission
name String
code String



SpiffWorkflow Engine

BPMN 2.0 notasyonunun önemli bir kısmını destekleyen, Python ile yazılmış bir iş akış motoru (workflow engine) uygulaması olan SpiffWorkflow incelenmiştir. Mevcut haliyle, tüm ihtiyaçlara cevap veremeyeceği tespit edildiğinden, ZetaOps tarafından genişletilerek yazılmaya devam edilmektedir. Genişletilmiş hali ile bu kütüphane tüm uygulamanın hareket zeminini oluşturmaktadır.

Zetaops sürümü olan kütüphane ile, uygulama iş mantığının anahatları BPMN 2.0 notasyonuna uyumlu XML diagramlarından okunarak işletilecektir. Öğrencilerin sisteme giriş yapmasından arka planda çalışacak zamanlanmış görevlerin işletilmesine kadar tüm iş akışları, bu iş akış motoru tarafından yönetilecektir.

Pyoko

Riak veri şemalarının Python nesneleri olarak modellenmesi, bu modeller arasında bağlantılar tanımlanabilmesi, verilerin kayıt sırasında şema tanımlarına göre doğrulanması, kayıtlı verilerin pratik bir API ile sorgulanabilmesi ve geliştirme süresince bu şemalarda yapılacak değişikliklerin sürümlendirilerek saklanabilmesiamacıyla arka uçta Riak ve Redis'i kullanan Pyoko kütüphanesi ZetaOps tarafından geliştirilmektedir.

ZEngine

ZetaOps tarafından geliştirilmekte olan ZEngine, SpiffWorkflow'u taban alan basit bir web çatısıdır. Bu yapıda önyüze yönelik her iş akışının bir URLsi olmakta ve o anda işletilmekte olan iş akışı adımında referans verilen uygulama

nesnesi (view class) request ve response nesneleri ile çağırılmaktadır.

Kural Motoru (Rule Engine)

Uygulamanın, kanun ve yönetmelik değişikliklerine bağlı olarak zamanla değişebilecek tanım ve kurallara dayalı iş mantığı, merkezi bir depodan kolayca güncellenebilecek ve sistem yöneticileri tarafından düzenlenebilecek kural setleri ile tanımlanacaktır.

Zato Servisleri

SOAP, REST, JSON, XML, CSV, PB gibi farklı protokol ve veri tipleriyle konuşan servislerin dönüşümü Zato ESB üzerinde yapılacaktır. Harici istemciler ve farklı modüller tarafından ihtiyaç duyulan işlevsellikler Zato ESB üzerinde çalışan mikro servisler olarak sunulacaktır. Uygulamanın hizmet sağlayıcı olduğu her durumda REST stili kullanılacaktır.

Falcon WSGI Framework

Çok hafif ve hızlı bir web çatısı olan Falcon, WSGI sunucusundan gelen requestleri Zengine'e aktarmak için kullanılacaktır. Kullanıcı oturumları tarayıcı çerezleri ve Redis tabanlı olarak bu katmanda yönetilecektir.

Gunicorn WSGI Server

Gunicorn, Python tabanlı, WSGI uyumlu az sistem kaynağı tüketen hızlı bir web sunucusudur.

Raporlama ve Analiz

Önceden oluşturulan standart raporlara ek olarak, indekslenmiş veri üzerinde gelişmiş sorgulamalar yaparak her türlü günlük ihtiyaca yanıt verebilecek esnek bir raporlama altyapısı geliştirilecektir. Verilerin çok çeşitli şekillerde incelenmesine ve derlenmesine olanak veren Python tabanlı analiz betiklerine ek olarak, indekslenmemiş büyük miktarda veri üzerinde MapReduce işlemleri yapabilmek için Erlang betikleri de kullanılabilecektir.

Kullanıcı Arayüz Bileşenleri

- Angular.js

AngularJS, MVC (Model View Controller) deseni sağlayan bir javascript uygulama çatısıdır. Kullanıcı arayüzü işlemlerini gerçekleştirecek tüm fonksiyonlar için kullanılır. AngularJS standart sunucu taraflı yazılım geliştirme tekniklerini önyüze uygulayan ve önyüz geliştirmeyi hızlandıran bir uygulama çatısıdır. Karmaşık uygulamalarda DOM yönetimini başarıyla gerçekleştirir ve bu sayede uygulamanın kesintisiz ve sorunsuz çalışmasını sağlar.

- Karma

Karma, Uygulama fonksiyonları için yazılmış testleri uygulayan test sürücüsüdür. Uygulamamızda Jasmine test çatısı testlerinin çalıştırılmasında kullanılır. Geliştiricinin her bir test ortamı için ayrı ayrı yapılandırma dosyası oluşturmadan tek bir yapılandırma ile testleri çalıştırabilmesini sağlar.

- Selenium

Selenium, E2E testlerin çalıştırıldığı test platformudur. Kullanıcının tarayıcıda gerçekleştireceği işlemlerin sunucudan dönecek sonuca kadar test edilmesini sağlar.

- Protractor

Protractor Selenium E2E testleri için bir çözüm enteratörü uygulama çatısıdır. Angularjs için Selenium özelleştirmeleriyle daha etkin ve bekleme sürelerini optimize ederek daha kısa sürede test edilmesini sağlar.

- Jasmine

Jasmine, javascript testleri için kullanılan bir uygulama çatısıdır. Uygulama fonksiyonlarının testlerinde başarılı sentaksı ile geliştirme sürecini hızlandırır.

- Bower

Bower, uygulamada kullanılacak paketlerin yönetimi için kullandığımız paket yönetim aracıdır. Uygulamanın gerektirdiği paketlerin kurulum esnasında eksiksiz şekilde ve sürüm uyumlu olarak kurulumunu sağlar.

- Grunt

Grunt javascript uygulamaları için bir görev yürütücüsüdür. Küçültme, derleme, paketleme, testler gibi tekrarlanan görevleri otomasyon ile yürütmek için kullanılır.

- Nodejs

Nodejs javascript uygulamaları için sunucu taraflı çalışma zamanı ortamıdır (runtime environment). Uygulama geliştirilirken bower, jasmine, karma gibi araçların kullanılması için gereklidir. StackTrace.js

- npm

npm nodejs için paket yönetim aracıdır. Uygulamanın geliştirme ortamı için gerekliliklerinin yönetilmesini sağlar.

- Bootstrap3

Bootstrap3 grid sistem standardına uygun uyumlu (responsive) arayüz geliştirmek için kullanılan html, css vs javascript uygulama çatısıdır. Uygulamanın değişik ekran boyutlarında ve farklı cihazlarda sorunsuz çalışması için kullanılır.

Kullanıcı arayüz tasarımında uyulacak kurallar ve ilkeler

- Tüm tasarım bileşenleri html5 standardına uyacaktır.
- Tasarım, kullanıcı arayüzü temiz ve tutarlı modeller temel alınarak anlamlı, kullanışlı ve amaca hizmet edecek şekilde organize etmelidir.
- Basit ve sık yapılan işlemleri kolayca gerçekleştirebilmeli, kullanıcıyla açık ve kolay iletişim kurabilmeli, uzun işlemler için kullanışlı kısayollar sağlamalıdır.
- Kullanıcı arayüzü tasarımı, yapılacak işlemler için tüm ihtiyaç duyulan opsiyonları ve materyalleri kullanıcının dikkatini dağıtmadan ve tam şekilde verebilmelidir.
- Tasarım kullanıcıyı değişiklikler halinde bilgilendirmeli, kullanım esnasında oluşacak hataları kullanıcının anlayacağı şekilde sunabilmelidir.
- Tasarım bileşenleri tekrar kullanılabilir olmalıdır.
- Tasarım tüm ekran çözünürlüklerinde düzgün çalışabilmelidir.
- Tasarım özürlü [22]_ kullanıcılar için "mümkün" olduğu kadar kolay bir kullanım sunabilmelidir.

Kullanıcı veri girişi ilkeleri

- Kullanıcı verileri güvenli şekilde ve amaca yönelik geçerlilik kuralları çerçevesinde girilebilmelidir.
- Kullanıcı daha az vuruş kullanarak kısa sürede veri girebilmelidir. Bunun için otomatik tamamlayıcılar, açılır menüler gibi kolaylaştırıcı bileşenler kullanılmalıdır.

Arayüz tasarımı ilkeleri

- Arayüz farklı amaçlar için kullanılacak farklı bölümlerden oluşmalıdır.
- Kullanıcı her zaman sistemde nerede olduğunu ve hangi bilgilerin ona gösterildiğini bilmelidir.
- Arayüz kolay kullanımlı ve estetik olmalıdır.
- Arayüzün kullanımı kolay öğrenilebilmelidir.
- Arayüz kullanıcının minimum eforuyla çalışabilmelidir.

Hata mesajları, uyarılar ve gösterilecek diğer bilgi ilkeleri

- Kullanıcı hatalar hakkında anlaşılır şekilde bilgilendirilmelidir.
- Uyarılar kullanıcının etkileşimini kesintiye uğratmayacak şekilde gösterilmelidir.
- Tekrar eden durumlarda kullanıcı deneyimini kesintiye uğratmamalı ve tekrarlı hatalar farkedilerek ona göre gösterilmelidir.
- Kullanıcının yapacağı işlemle alakasız bilgiler arayüzde yer almamalıdır.

Modül Yapısı ve Klasör Hiyerarşisi

app/ Uygulamanın yer aldığı dizindir.

bower_components/ Bower paket yönetimi ile uygulama kullanılan harici paketlerin tutulduğu dizindir.

components/ Uygulama ortak bileşenlerinin bulunduğu dizindir.

dashboard/

Yönetim paneli teması, controller, view ve testlerinin bulunduğu dizindir. her bir modül için benzer bir dizin buunacaktır.
+dashboard.html +dashboard.js +dashboard_test.js

app.css/ Uygulama genel css dosyası

app.js/ Uygulama ana javascript dosyası

index.html/

e2e-tests/

protractor.conf.js/ E2E testlerinin yapılandırıldığı dosyadır. E2E testleri için protractor kullanılmaktadır.

scenarios.js/ E2E test senaryolarının yazılı olduğu dosyadır.

node_modules/ Uygulamada kullanılan nodejs modüllerinin yer aldığı dizindir.

bower.json/ Uygulama bağımlılıklarının yapılandırıldığı dosyadır.

karma.conf.js/ Karma testleri için yapılandırma dosyasıdır.

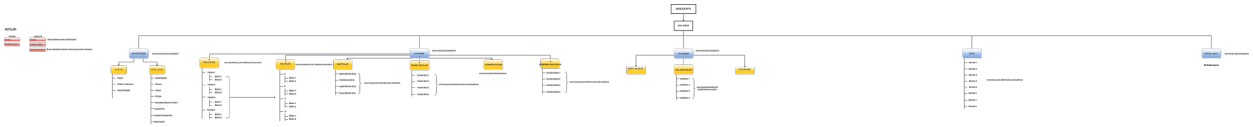
package.json/ Uygulama nodejs bağımlılıklarının yapılandırıldığı dosyadır.

Ekran Listesi

Uygulamada belirlenen yetkilendirme şemasına göre, yetkili olan kişinin bir ya da daha çok kontrol paneli (dashboard) olabilir. Her bir ekran görünümü belirli parçalardan oluşacaktır. Giriş talebi yapıldığında yetki türüne göre kullanıcının ekranı bileşenleri bir araya getirilerek uygun hale getirilir (render) ve gösterilir. Sistemin kullanacağı ortak bileşenler birer Angular [23]_ modülü olacaktır.

Navigasyon Diagramı

Üniversite web uygulaması ve katmanlarını içerir.



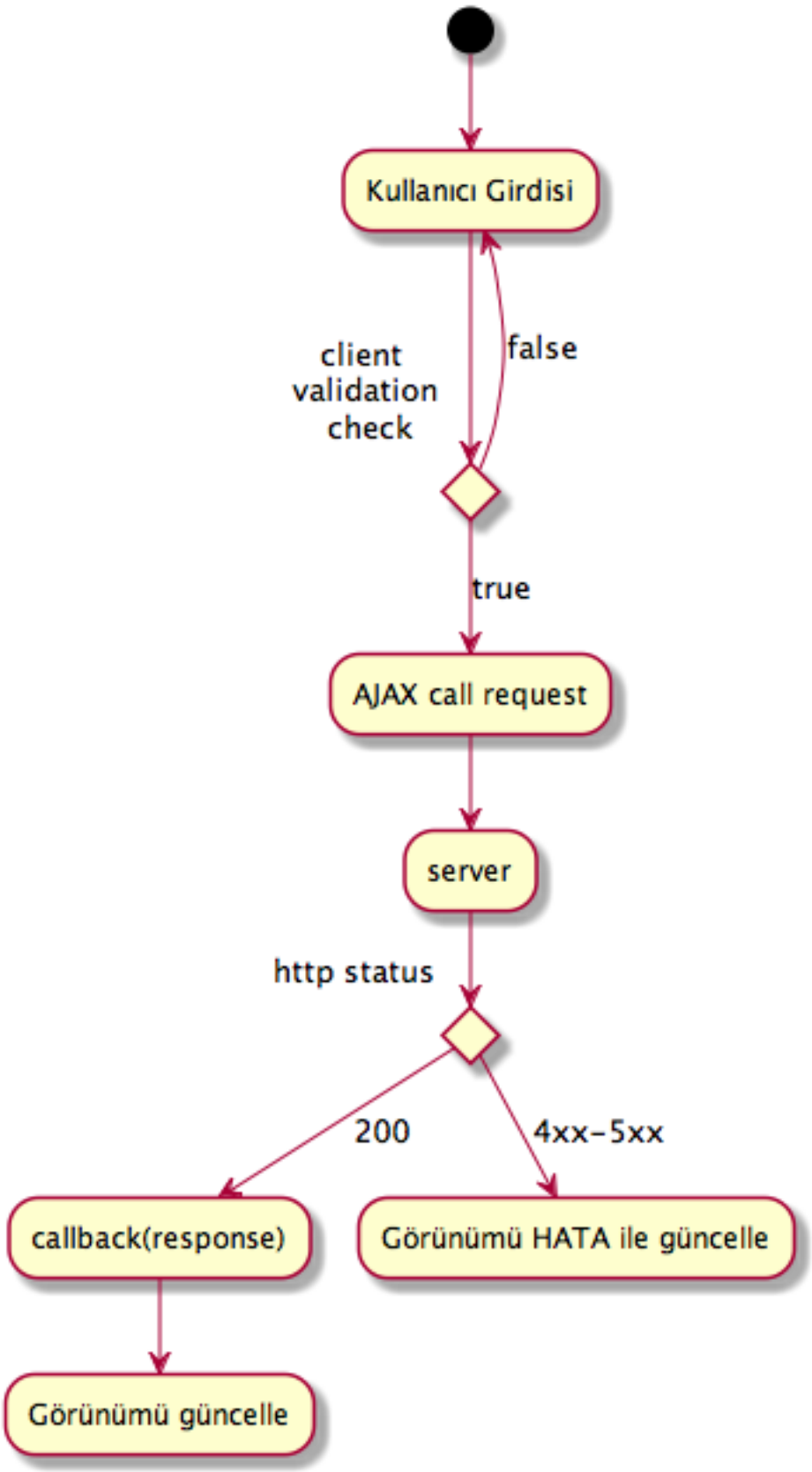
Çevrimiçi Yardım ve Rehberler

Kullanıcıların arayüzü kolayca öğrenebilmelerini sağlamak ve kullanım esnasında karşılaşacakları sorunlar sırasında yardım etmek amacıyla çevrimiçi yardım ve rehber araçları kullanılır.

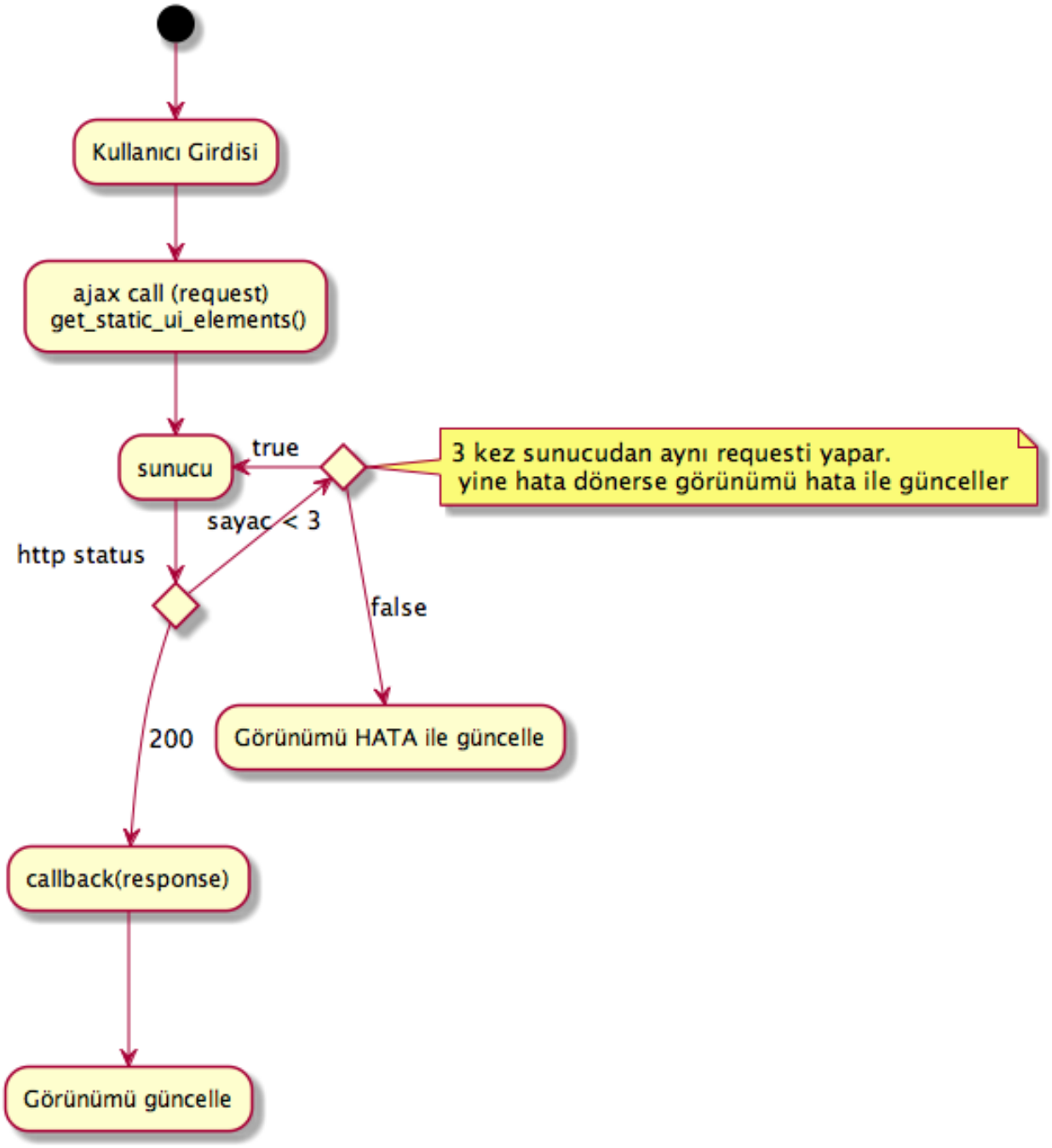
Sisteme ilk kez giriş yapan kullanıcıyı yönlendirmek ve arayüzün işlevleri hakkında bilgilendirmek amacıyla rehber araçları kullanılır.

Kullanıcı sistemin bir noktasında sorunla karşılaştığında bağlamsal olarak derhal konuyla ilgili yardıma erişmesi için çevrimiçi yardım araçları kullanılır.

Şekil: Kullanıcı Formları Request/Response Cycle Talep/Cevap Yaşam Döngüsü


Kullanıcı Girdisi
client validation check
false
true
AJAX call request
server
http status
200
4xx-5xx
callback(response)
Görünümü HATA ile güncelle
Görünümü güncelle

Şekil: Kullanıcı Arayüz Bileşenleri Request/Response Cycle Talep/Cevap Yaşam Döngüsü



1.2.7 Veri Tasarımı

Geçici Veri

Uygulamanın sık kullandığı veriler Redis üzerinde önbelleklenecektir. Bu önbellek verileri işlemci ve veritabanı yükü açısından pahalı işlemlerle hesaplanmış değerleri ve veritabanından sık sık okunan verileri içerir. Pyoko kütüphanesi üzerinden yapılan doğrudan veri çağrıları (get request) otomatik olarak önbellekleneren veri tabanı üzerindeki sorgu

yükü hafifletilecektir.

Önbellekleme haricinde, kullanıcı oturumları da Redis üzerinde tutulacaktır. Kullanıcının o an geçerli yetkileri, oturum boyunca yaptığı işlemlerle ilgili durum bilgileri de yine kullanıcı oturumu içerisinde tutulacaktır.

Kalıcı Veri

Verilerin kalıcı olarak saklanacağı Riak, basit anahtar-değer çiftlerinden map, set, counter gibi gelişmiş veri tiplerine, nihayetinde tutarlılıktan (eventually consistent) kesin tutarlılığa (strong consistency) kadar çeşitli veri saklama kiplerini destekleyen gelişmiş bir NoSQL veri tabanıdır. JSON biçiminde saklanacak olan veriler, Riak'ın dahili Apache Solr entegrasyonunu sayesinde istenilen incelikte indekslenmekte ver sorgulanabilmektedir.

Kalıcı olarak depolanacak tüm veri sürümlendirilerek saklanacaktır. Bu sayede her hangi bir kaydın son 100 sürümü ya da son 10 yıl içindeki tüm sürümlerine istenildiği an ulaşılabilecektir. Sürüm sayısına ya da süreye göre ne kadar geriye dönük saklama yapılacağı her bucket için kendi model tanımı altında yapılacaktır.

Veritabanı seviyesinde herhangi bir şablon kısıtı olmamasına rağmen, veriyi tutarlı biçimde saklayabilmek ve hızlı bir şekilde sorgulayarak erişebilmek için tüm veriler iç içe Python sınıfları şeklinde modellenecek, bu modeller kayıt esnasında JSON şeklinde biçimlendirilerek saklanacak ve yine modelde tanımlandığı şekilde indekslenecektir.

Test ve Prod ortamları için farklı bucketlar kullanılacak, değişen konfigurasyon ve yük testleri için geçici Riak clusterları açılacaktır.

Veri ve Veri Şablonu Göçü

Uygulamanın yaşamı boyunca veri şablonlarında yapılacak güncellemeler ve bu güncellemeler nedeniyle verinin kendisinde yapılması gerekecek veri göçleri Pyoko kütüphanesi ile sürümlendirilecek ve yönetilecektir. Uygulamayı bir üst sürüme güncellemek ya da önceki sürüme dönmek için gerekli veri tabanı işlemlerini içeren göç betikleri geliştirme aşamasında Pyoko yardımıyla türetilecek ve kod deposuna eklenecektir. Gerektiğinde, elle özel göç betikleride yazılacaktır.

Varlıklar (Entities)

Uygulama içinden çağrılan tüm veri adları İngilizce olacaktır. Her bir entity için ayrıca bu belgeye ek belgeler oluşturulacaktır.

- Personel
- Student
- Program
- Lecture
- Unit

Dış Veri Kaynakları

Sistem birçok veri kaynağı ile konuşabilecek, ihtiyaç duyulan veri alışverişini sağlayacaktır. Bu dış kaynakların biçemleri XML, JSON şeklinde değişik olabilir. Konuşulacak servislerin detayları aşağıdaki gibidir.

LDAP

Birçok üniversitede doğrulama ve yetkilendirme gibi amaçlar için aktif şekilde kullanılan LDAP sistem tarafından desteklenecektir. LDAP'ta yapılan değişiklikler sisteme düzenli şekilde yansıtılacak, sistem gerektiğinde LDAP şemalarında değişiklik yapabilecektir. Özellikle göç aşamaları gibi LDAP kullanımının kaçınılmaz olduğu zaman ve şartlar için öngörüşmüştür.

KBS

Kamu Harcama ve Muhasebe Bilişim Sistemi (KBS) Maliye Bakanlığı tarafından sağlanan, kamu kurumlarında tahakkuk ve ödeme işlemlerinin otomasyonunu sağlayan bir edevlet uygulamasıdır. Üniversitelerde de birçok mali işlem KBS aracılığıyla gerçekleştirilmektedir. KBS sisteminin el verdiği ölçüde entegrasyon sağlanacaktır.

HİTAP

HİTAP(Hizmet Takip Projesi), devlet memurlarının hizmetlerinin takibi amacıyla Sosyal Güvenlik Kurumu tarafından geliştirilmiş edevlet uygulamasıdır. Personel bilgilerinin iki yönlü güncellenmesi için HİTAP servisi ile düzenli şekilde veri alışverişi yapılacaktır. HİTAP bir SOAP servisidir.

ASAL

ASAL Servisi, Milli Savunma Bakanlığı tarafından sağlanan yurttaşların askerlik durumlarını sorgulayabildikleri bir edevlet uygulamasıdır. Bu uygulama ile web servisi şeklinde konuşup, erkek öğrenci ve personin askerlik durumları karşılıklı olarak takip edilecektir.

ÖSYM

Öğrenci kayıtlarının otomatizasyonuna yardımcı olmak için, yerleştirme bilgileri ÖSYM'nin sunduğu

YÖKSİS

YÖKSİS (Yükseköğretim Bilgi Sistemi) YÖK tarafından kurulan yükseköğretim kurumlarının çeşitli bilgilerinin merkezi şekilde saklandığı sistemdir. YÖK üniversitelerden YÖKSİS'e düzenli veri girişi beklemektedir. Ayrıca YÖK üniversitelerde açılan bölüm ve programların bilgilerinde çeşitli güncellemeler yapmakta, bu güncellemelerin verinin bütünlüğü ve tutarlılığı için en kısa sürede üniversitelerin sistemlerine aktarılması gerekmektedir. YÖKSİS entegrasyonu bu ihtiyaç ve sorunlara çözüm olacaktır. YÖKSİS bir SOAP servisidir.

AKS

Adres Kayıt Sistemi, Nüfus ve Vatandaşlık İşleri tarafından sağlanan bir edevlet hizmetidir. Sistemimiz bu hizmet ile tam entegrasyon halinde olacak ve sisteme kayıtlı kişilerin adres bilgilerini bu sistemdeki kayıtlar ile güncelleyecektir.

MERNİS

AKS gibi merkezi kimlik hizmetidir. Sistemde kayıtlı kişilerin kimlik bilgileri MERNİS ile uyumlu şekilde saklanacaktır.

BANKALAR

Öğrenci Harç ve ödeme işlemlerinin takip edilmesi için bankaya açılacak olan servistir. Banka öğrencilerin ödemeleri gereken miktarları bu servis aracılığı ile öğrenir ve ödeme bilgilerini sisteme geri bildirir. Bizim tarafımızda açılacak servis REST türünde olacaktır.

SMS

Öğrenci ve personele SMS bildirimleri yapmak isteyecek öğrenciler üniversiteler kendi servislerini Zato ESB ile kolayca yazabileceklerdir.

1.2.8 Rol ve Yetki Kontrolü (ACL - Access Control List)

Rol ve Öznitelik tabanlı hibrid bir yetkilendirme ve veri erişim kontrol modeli kullanılacaktır. Kurgulanacak sistem, Midpoint IDM gibi kimlik yönetimi sistemleri ile dış kimlik kaynaklarıyla (LDAP, veritabanları) REST metoduyla veri alışverişi yapabilecektir.

Rol Tabanlı Yetkilendirme (Rol Based Authorization Control)

Rol ve yetkiler, Akademik ve İdari Birimler (Units), Soyut Roller(Abstract Roles), İş Akışı(WorkFlows), İş Akışı Adımları(WorkFlow Tasks) ve Kullanıcı (User) kavramlarının kesişimleri sonucu ortaya çıkarlar.

Kullanıcıların bir birimde, tanımlanmış herhangi bir role (bölüm sekreteri, öğretim elemanı, öğrenci vb.) dahil olmaları onları belirli workflowların belirli adımları için yetkili olmalarını sağlayacaktır. Örnek verecek olursak:

Birim: Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü Soyut Rol: Bölüm Başkanlığı Kullanıcı: Ayşe Bilgin, Öğretim Üyesi, Prof. İş Akışı: Ders, Öğretim Elemanı Paylaşımı. Bu iş akışının 2 aktörü vardır. Paylaşımı yapan bölüm sekreteri, bu paylaşıma onay veren bölüm başkanı. İş Akışı Adımları: İş akışı, yineleyen düzeltme - gözden geçirme ve nihayetinde onay ve ilgililere bildirim adımlarından oluşmaktadır.

Ayşe Bilgin, bölüm başkanı olarak, sadece kendi bölümü ile ilgili olarak bu iş akışının ilgili adımları için otomatik olarak yetkilendirilmiş olacaktır.

Öte yandan Ayşe Bilgin bir başka birimde, geçici olarak, bir rol ile sadece belli görevleri yapmak üzere görevlendirilmiş olsun. Bu durumda Ayşe Bilgin'in yetkileri, ilgili rolün sahip olduğu yetkilerin (iş akışı adımları yetkileri) bir kısmı ile genişletilebilecektir. Bu da ancak ilgili iş akışı adımlarının ilgili birim ve rol ile somutlanmasının yetki olarak tarif edilmesiyle mümkün olabilmektedir.

Öznitelik Tabanlı Yetkilendirme (Attribute Based Authorization Control)

Bu modelde ise yukarıda tarif edilmiş yetkilerin, mekan, zaman, geçici sınırlamalar gibi özelliklere göre daraltılması veya genişletilmesidir. Belirli yetkilerin ancak belirli zaman aralıklarında, belirli mekanlarda ve kullanıcının sahip olduğu özniteliklerin belirli eşleşmeleri sağladığı durumlarda gerçekleşmesi durumudur.

İzindeki veya dışarıda görevlendirilmiş bir personelin belirli iş akışlarını yürütememesi, kritik iş akışlarının mesai saatleri içinde veya okul içinden yapılması vb.

Satır ve Hücre Seviyesinde Erişim Kontrolü

Yukarıdaki modellere göre yetkilendirilen kullanıcılar, belirli bir buckettaki kayıtların hangilerine erişebilecekleri ve erişebildikleri bu kayıtların hangi alanlarını görebilecekleri konusunda sınırlandırılacaklardır. Bu sınırlamaların hemen hepsi veri erişim katmanı (pyoko) seviyesinde işletilecektir.

Yukarıda öğrenci listesini gösteren örnek view metodunda Student modelindeki tüm kayıtlar istenmesine karşın, erişim katmanı model içerisine tanımlanmış olan satır tabanlı kısıtları işleterek o an giriş yapmış olan personelin, sadece kendi bölümündeki öğrencileri görmesine müsade etmektedir. Benzer şekilde eğer cell_filters süzgeci tanımlandıysa, veri tabanından dönen veriler, kullanıcının görmeye yetkili olmadığı hücreler filtrelendikten sonra view metoduna döndürülür. Veri erişim kontrolü mümkün olduğunca model seviyesinde yapılarak, olası hata ve güvenlik açıklarının en aza indirgenmesi hedeflenmektedir.

İki Seviye Yetkilendirme

Kullanıcılar bazı kritik işlemler için ikinci bir parola ile yetkilendirilirler. Kullanıcılara ait bazı kayıtları silme, nihayi onayları verme, toplu kayıt işlemleri gerçekleştirme gibi durumlarda ikinci bir adımda bu işlemlerin kritik oldukları anımsatılır ve kendilerine ait başka bir parola ile devam etmeleri sağlanır.

Yetki Devri

Rol veya role ait bazı yetkiler farklı kullanıcılara devredilebilirler. Devredilen yetkiler tek tek iş akışı adımları veya bir rolün sahip olduğu tüm yetkiler şeklinde belirlenebilir. Yetki devri belirli sürelidir. Yetki devredilen kullanıcı için geçici bir rol tanımlanır. Kullanıcı bu geçici rol ile kendi rolü arasında geçiş yaparak ilgili görevleri yerine getirebilir.

Notes: İncelenecek diğer konular aşağıdadır.

http://www.simplecloud.info/ https://github.com/concordusapps/python-scim https://www.openhub.net/p/gripped
http://wiki.openid.net/w/page/12995226/Run%20your%20own%20identity%20server
https://pypi.python.org/pypi/authentic2/2.0.1

OAUTH 2 buna nasıl yaklaşacağız? SSO Federation (shibboleth) sistemimizle olan iletişimini ele alacak mıyız?

1.2.9 Test Döngüsü

Yazılım geliştirme ve buna bağlı test döngüsü belgesinde detaylı olarak verilmiştir.

1.2.10 Yerelleştirme

Yazılımın temel dili Türkçedir. Çoklu dil desteği sistemin doğal özelliklerinden birisidir. Gettext kullanılacaktır.

1.2.11 Güvenlik Ölçümleri

Güvenlik test ve kontrolleri "Yazılım Geliştirme ve Test Döngüsü" belgesinde detaylı olarak verilmiştir.