ETS ( Engine Test System ) SDS



문서정보

OFFICIAL

Edit

marked by 이동민/ASD서비스개발팀 on Oct 01, 2019 17:26

프로젝트명 / 버전

ETS ( Engine Test System ),

-



[~~RAS-135~~](https://works.ahnlab.com/browse/RAS-1355)

[~~5~~](https://works.ahnlab.com/browse/RAS-1355)

ETS SDS 작성

RESOLVED

소속/ 작성자

[이동민/ASD서비스개발](https://docs.ahnlab.com/display/~dongmin.lee)

[팀](https://docs.ahnlab.com/display/~dongmin.lee)

DOCS 내 모든 문서의 생성, 열람, 편집은 시스템 로그에 기록/관리되며, 문서의 중요도가 높은 문서는

열람, 편집을 제한해야 하는 경우 페이지 제한 기능 이

용 가능



CONFIDENTIALITY/SECURITY WARNING

이 문서에 포함되어 있는 정보는 AhnLab의 자산이며, AhnLab의 서면 허락 없이 사용되거나, 재 가공 되거나, 외부로 반출 할 수 없습니다.

이 문서에

대한 열람 및 보관할 수 있는 사람은 분실, 도난, 반출에 대한 책임을 자동 승계하는 것에 대하여 동의하는 자에 한 합니다.

1. 개요
   1. 목적
   2. 범위
   3. 정의 및 약어
   4. 관련문서
   5. 개발자 스펙
2. 설계 고려사항
   1. 소프트웨어 설계 목표
   2. 제약사항
      1. H/W 제약사항
      2. S/W 제약사항
      3. N/W 제약사항
      4. 외부 제약사항
   3. 기타
3. 설계
   1. 구성
      1. 소프트웨어 배치
      2. 소프트웨어 아키텍처
      3. Component Design
         1. Engine Test Manager ( Component )
            1. - Epack Downloader ( Service )
            2. - Sample Downloader ( Service )
            3. - Engine Test Executor ( Service )
         2. Update Test Manager ( Component )
            1. Patchset Downloader ( Service )
            2. SFX Downloader ( Service )
            3. Update Test Executor ( Service )
      4. 운영체제와의 관계도
   2. Data Design
      1. Interface Design
         1. Component Interface
         2. Module Interface
         3. Process Interface
         4. N/W Interface
         5. H/W & Device Interface
      2. Internal Software Data Structure
      3. Global Data Structure
      4. Temporary Data Structure
      5. Permanent Data Structure
      6. Database Description
      7. Database Description
         1. ERD(Entity Relationship Diagram)
         2. Data Dictionary
   3. Architectural Design → 이런 단어는 소프트웨어 공학에 없습니다.
      1. Use Case Diagram
      2. Class Diagram
      3. Flow Chart
      4. DFD(Data Flow Diagram) - SRS 작성시 요구사항 발제자와 커뮤니케이션 도구
      5. Sequence Diagram
   4. User Interface Design
      1. Screen images
      2. Screen Object and Actions
   5. Component Design (& Module Design)
   6. 기타
      1. 소스트리 구성
      2. 빌드 설계
      3. 설치 설계
      4. 배포 설계
         1. 마스터 구성
         2. 배포 및 업데이트 방법
         3. 업데이트 방법
      5. 테스트 설계
         1. Classes of Test
         2. Expected Software Response
         3. Perfomance Bounds
         4. Identification of Critical Components

1. 개요

# 1.1. 목적

RAS시스템의 ETEST서비스를 더 빠르게, 더 정확하게 검사하고, 테스트를 추가하고 확장 가능하게 고도화하는 ETS( Enging Test System )를 설계한 다.

1.2. 범위 본 설계의 범위는 아래의 기능에 대한 설계이다.

구분 기능

Engine Test 샘플망에 구성된 Windows 운영체제에서 빌드된 Engine 과 샘플 서버를 활용한 정의된 샘플을 기준으로 Detect Test 를 수행한다.

Update Test Agent 에 설치된 V3 ES 9.0 제품을 기반으로 API 테스트를 통해 무결성 검사와 업데이트 테스트를 동시에 수행한다.

Monitoring InfluxDB 를 활용하여 시계열 데이터를 기준으로 모니터링하는 ASD 서비스와 동일한 수준의 모니터링을 수행한다.

1.3. 정의 및 약어

N/A

# 1.4. 관련문서

분류 참조 URL

SRS <https://docs.ahnlab.com/x/7R8ODQ>원페이저 <https://docs.ahnlab.com/x/nwoADg>비지니스 아키텍쳐 <https://docs.ahnlab.com/x/2HnXDQ>어플리케이션 아키텍쳐 <https://docs.ahnlab.com/x/43nXDQ>데이터 아키텍쳐 <https://docs.ahnlab.com/x/5nnXDQ>

# 1.5. 개발자 스펙

구분

요구 스펙

WEB

Apache Server Configuration

WAS

Tomcat Server Configuration and Operation

Java

Spring Base Hibernate Framework 이해 및 활용

Oracle SQL Query 활용

Jenkins C/S 구조의 Remote Agent 를 활용한 명령 실행 이해 및 활용

2. 설계 고려사항

# 2.1. 소프트웨어 설계 목표

* 기존 ETEST를 개선하여 엔진을 활용한 악성 샘플 진단 테스트를 고도화한다. ( 확장 가능한 테스트 플랫폼 활용 방식을 적용한다 ) - 기존 ETEST를 개선하여 제품을 기준으로 업데이트 테스트 및 무결성 검사 테스트를 고도화 한다.
* 새로운 ETS ( Engine Test System ) 프로젝트는 ALM ( ROSE )을 활용한 관리 범주에 입각하여 프로젝트를 진행 할 수 있도록 한다.

# 2.2. 제약사항

2.2.1. H/W 제약사항 모든 샘플은 물리 머쉰이 아닌 가상 머쉰에서 테스트를 수행한다. 즉, 물리 머쉰에서 직접 테스트를 수행하지 않는다.

2.2.2. S/W 제약사항 테스트 OS는 모든 OS를 테스트 할 수 없기 때문에, Public 하게 사용중인 Windows 10 Version 을 기준하여 테스트를 수행한다.

## 2.2.3. N/W 제약사항

샘플을 대상으로 테스트를 수행하기 때문에 샘플 서버망에서 Engine Test를 수행한다.

단, 패치셋과 SFX 파일을 대상으로하는 업데이트 테스트는 RND, Server 망에서 Update Test를 수행한다.

## 2.2.4. 외부 제약사항

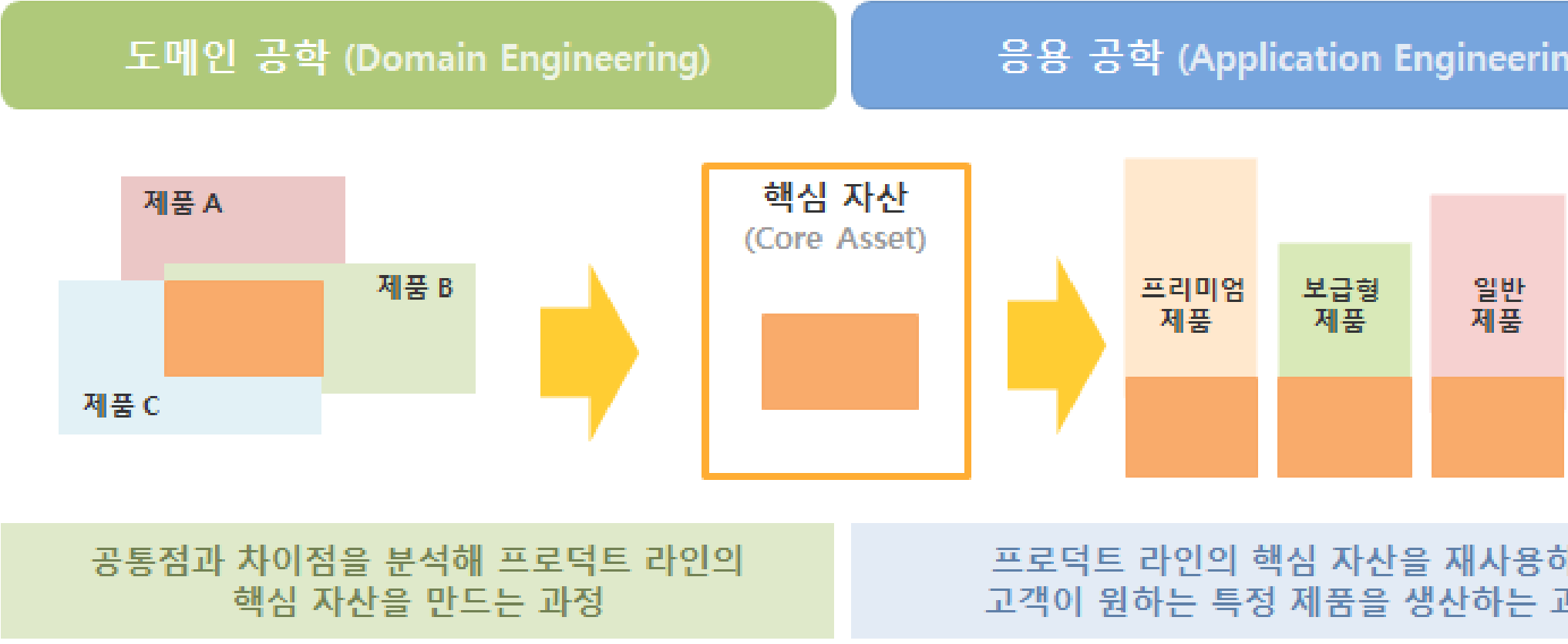
샘플 테스트의 경우, 샘플망에서 (Engine 바이너리를 다운로드 받기 위하여) RND에 접속하는 행위를 방지하기 위하여,

Agent 에 Jenkins Master가 명령을 내릴때 Engine 바이너리를 전송(transfer)한 후 JOB을 실행하도록 한다.

# 2.3. 기타

* PLE ( Product Line Engineering ) 기법을 활용하여, 재사용을 극대화한 프로젝트 구조를 적용한다.

PLE 에 대한 고찰 → PLE를 적용한 프로젝트 구조 차용 ( 3.6.1 절 )



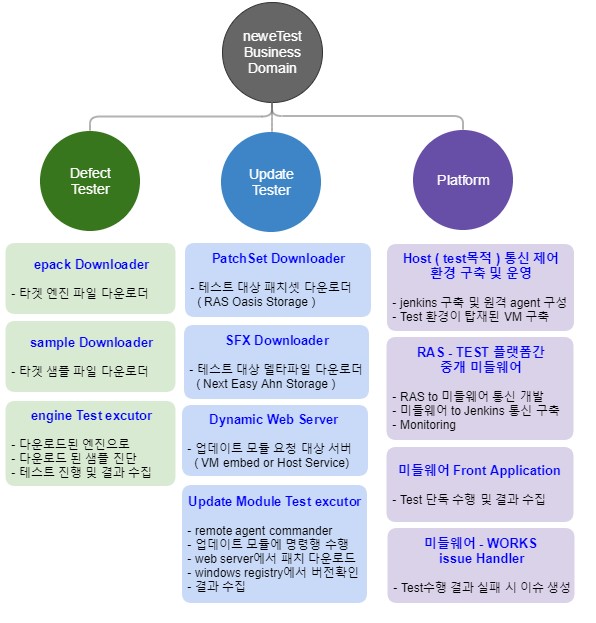
* 새로운 ETS ( Engine Test System )은 자체적인 Static Code Analysis ( SonarQube )를 적용하여 코드 품질을 유지하도록 한다.
* 새로운 ETS ( Engine Test System )은 Bitbucket ( [repos.ahnlab.com](http://repos.ahnlab.com/) )을 활용하여 GIT 으로 형상관리를 적용하는 ASD 서비스 개발팀의 첫 프로젝 트이다.

3. 설계

# 3.1. 구성

## 3.1.1. 소프트웨어 배치

반드시 작성되어야 하는 내용입니다. 소프트웨어를 개발 완료 후 타제품과의 관계 또는 타제품에의 적용 모습 등을 기술한다. 소프트웨어 배치는 다음의 비지니스 아키텍쳐로 갈음하도록 한다. 비지니스 아키텍쳐를 기반으로 달성하고자 하는 목적 시스템 구성과 배치를 확인 할 수 있기 때문이다.



ETS가 수행하는 비지니스는 크게 다음과 같이 3가지로 구분된다. ASD개발팀 packing 파트는 각 비지니스를 자동화한다.

Defect Tester

epack system에 적재된 엔진파일과 샘플파일을 다운로드 하고, 테스트 host에서 다운로드한 엔진파일로 샘플 진단 여부를 확인한다.

Update Tester

엔진 패치셋을 host 웹서버에 적재하여 mupdate 모듈에서 업데이트기능 수행 여부를 확인한다.

Platform

Jenkins - Remote Agent ( TEST HOST Controller ) 구성을 활용하여, common knowledge 로 확장할 수 있도록 한다.

Jenkins 결과 수집 및 RAS 통신, ETS 단독 수행 등의 기능을 담당할 middle ware Java Application을 배치한다.

middle ware Java Application 에 명령 및 결과 데이터를 표시할 frontend web application 을 독립 배치한다.

하위 업무는 독립적으로 분리가능하거나, 다른 업무와의 관계가 작게 유지되는 방향으로 식별하였다.

각 업무 간의 주요 흐름을 개념적으로 표현하면 다음과 같다.)

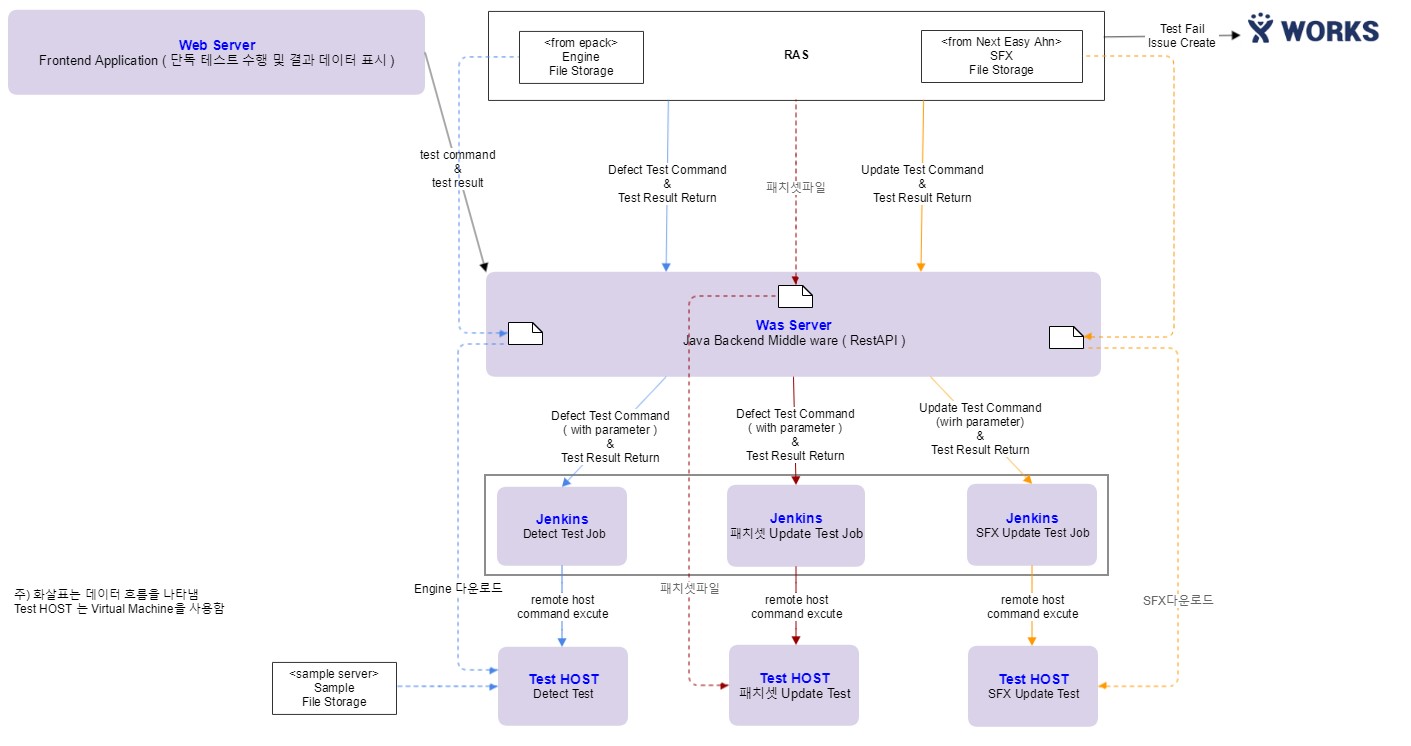
(

)

참고 : Defect, Update Test Module 은 Application Architect layer 에 해당하여 다음 페이지에서 확인 가능하다. :

[https://docs.ahnlab.com/x/43nXD](https://docs.ahnlab.com/x/43nXDQ)

[Q](https://docs.ahnlab.com/x/43nXDQ)



3.1.2

.

소프트웨어 아키텍처

소프트웨어의 Component의 개략적인 조망을 보여주는 아키텍처를 표시하고 설명합니다.

소프트웨어 아키텍쳐는 다음과 같이 구성한다.

비지니스 아키텍쳐를 기반으로 구체화된 어플리케이션 아키텍쳐로 본 장을 갈음한다. 어플리케이션 아키텍쳐로 각 시스템의 세부 컴포넌트와 모듈을 표시하고 상호 작용하는 목적이 기술되어 있기 때문이다.

1

.

2

.

3

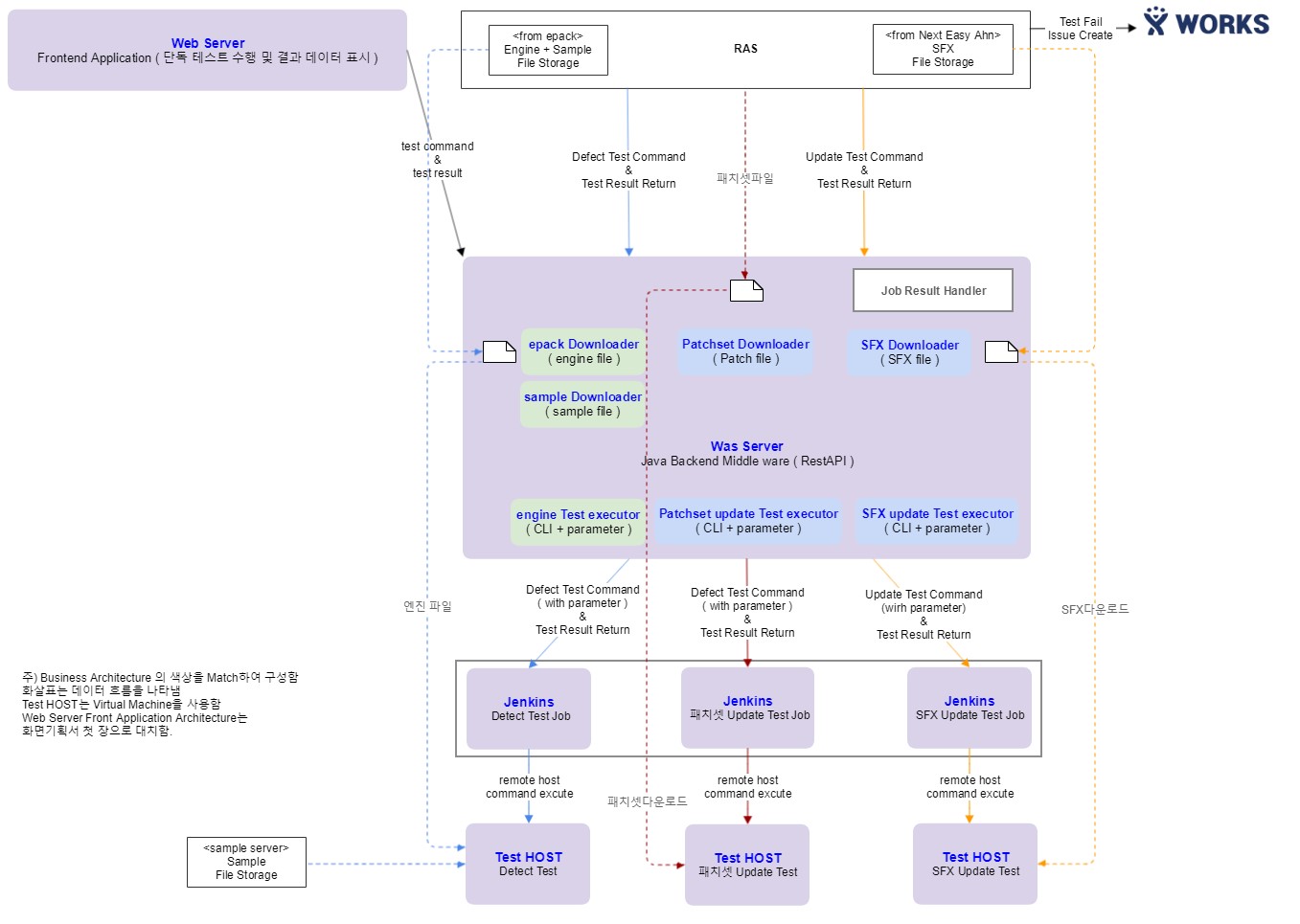
.

4

.

5

.



WAS Server의 비기능 관련 내용은 하기와 같습니다.

JVM Profile 관련 monitoring 기능 ( ELK - APM 활용 예정 및 opensource Java Profile monitor 활용 예정 )

Application log 관리 기능 ( ELK - File Beat 활용 예정 )

Application Status 관리 기능 ( ELK - Heart Beat 활용 예정 )

Application Request 관리 기능 ( ELK - Packet Beat 활용 예정 )

Application HOST 관리 기능 ( ELK - Metric Beat 활용 예정 )

## 3.1.3. Component Design

반드시 작성이 되어야 하는 내용입니다. 소프트웨어 구성요소인 각 Component와 그 하위 module을 정의하고, 각각의 기능 및 의존관계 등을 기술합니다. 각 Component 또는 Module 의 Public 함수들에 대한 Function Prototype 은 "3.4 Interface Design" 또는 "3.6 Component Design" 에 반드 시 기술되어야 한다.

### 3.1.3.1. Engine Test Manager ( Component )

EngineTestManager Component는 Epack에 적제되어 RAS에서 Oasis system에 적제된 엔진을 대상으로 정의된 샘플을 샘플 서버에서 받아와서 Test 할 Engine으로 Sample을 Detect 할수 있는지를 확인하는 컴포넌트이다.

3.1.3.1.1. - Epack Downloader ( Service )

EpackDownloader Service는 Oasis 에 적제되어있는 Engine을 Copy 해 오는 Service Component 이다.

#### 3.1.3.1.2. - Sample Downloader ( Service )

SampleDownloder Service는 Sample Sever에 적제되어있는 악성 샘플을 TEST 직전에 Copy해 오기위한 정보를 관리하는 컴포 넌트이다.

이 정보는 EngineTest 를 수행하는 HOST의 Agent에서 Job으로 TEST 직전 수행하여 악성 샘플을 복사해 오도록하는 정보관리 컴포넌트이다.

#### 3.1.3.1.3. - Engine Test Executor ( Service )

EngineTestExcutor Service는 EpackDownloader Service와 SampleDownloader Service로 부터 데이터를 취합하여, Jenkins Master Server의 Engine Test Job 수행을 하기위한 Parameter를 전달하고, Job을 수행시키기 위한, Event를 발생시키는 컴포넌트이다.

### 3.1.3.2. Update Test Manager ( Component )

UpdateTestManager Component는 NEA ( Next Easy Ahn ) 시스템에 의해 생성된 Patchset 과 SFX 파일을 대상으로 실제 안랩 제품( V3 ES 9.0 )이 상기 정의된 업데이트 파일을 Load하여 Update가 되는지를 확인하는 컴포넌트이다.

#### 3.1.3.2.1. Patchset Downloader ( Service )

PatchsetDownloader Service는 NEA에서 구성된 파일을 RAS에서 패킹하여, Oasis에 적제된 Patchset 파일을 Middleware WAS에 로드하여 적제하는 컴포넌트이다.

#### 3.1.3.2.2. SFX Downloader ( Service )

SFXDownloader Service는 NEA에서 구성된 파일을 RAS에서 패킹하여, Oasis에 적제된 SFX 파일을 Middleware WAS에 로드하여 적제하는 컴포넌트이다.

#### 3.1.3.2.3. Update Test Executor ( Service )

WAS에 로드된 업데이트 파일은 Update Test 수행할 때 Load 대상이 되며,

TEST HOST Agent에서 안랩 제품의 업데이트 서버 설정을 API로 변경한다. ( 업데이트 서버를 Middleware WAS로 변경 ) 따라서,

UpdateTestExecutor Service는 Remote Host Agent에 전달할 업데이트 서버변경 정보를 가지고 Jenkins Master Server의 JOB을 수행시키는 컴포넌트이다.

3.1.4. ~~운영체제와의 관계도~~

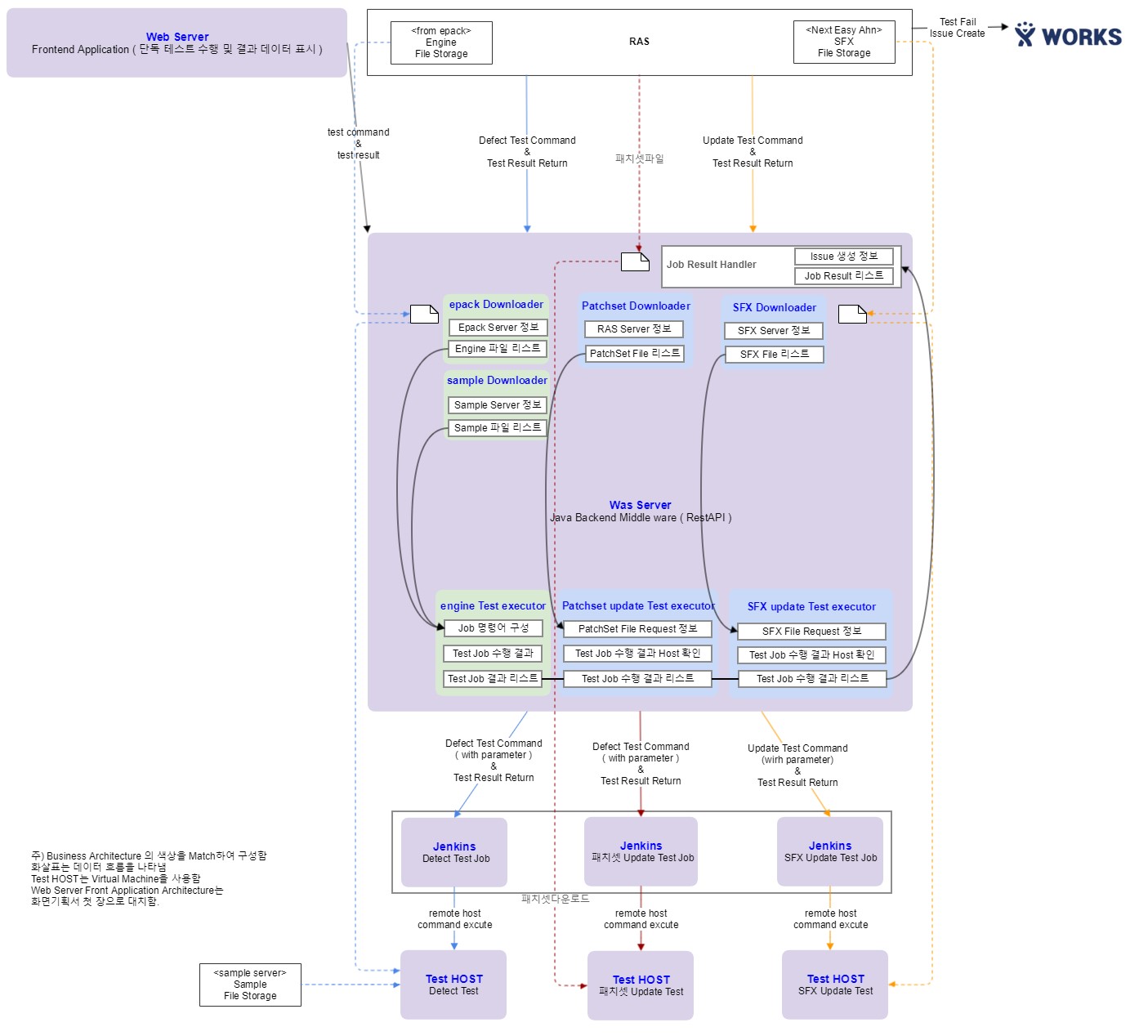
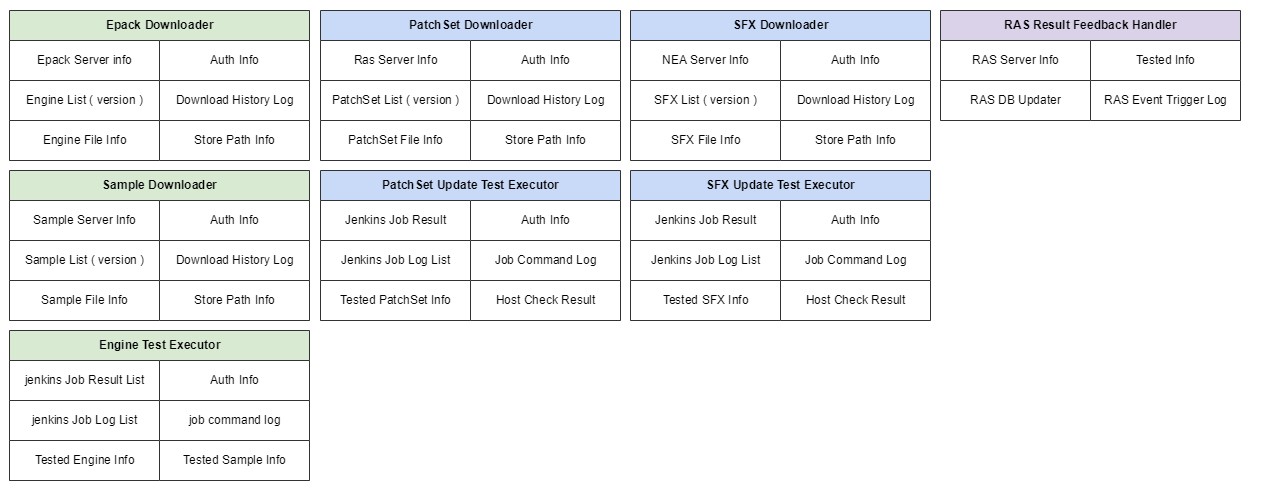
운영체제와는 Dependency가 없음.

3.2. Data Design 반드시 작성되어야 하는 내용입니다.

소프트웨어 Component 들 간에 전달되는 자료구조를 설명한다.

설계 중 사용되는 모든 Data에 대해 기술한다. 데이터 디자인은 다음과 같이 구성한다. 어플리케이션 아키텍쳐를 기반으로

구체화된 컴포넌트와 모듈이 사용하는 데이터를 설계하는 데이터 아키텍쳐로 갈음한다. ( 사유 : 데이터의 범주와 전달하는 데이터를 표기하고 있 기 때문이다. )



## 3.2.1. Interface Design

반드시 작성이 되어야 하는 내용입니다.

Component 간 발생할 수 있는 모든 Interface Line에 대한 정의 호출하는 방향으로 Line을 연결한다. Return에 대한 Line은 생략할 수 있다.

인터페이스 디자인의 전제는 시스템 정의에서 상세화된 컴포넌트의 데이터 아키텍쳐가 Input 값이며, ( 어떤 데이터를 input하고 어떤 데이 터를 return 할 것인지 정의되어야 한다 ) 컴포넌트 기준으로 사용되는 인터페이스가 상세화되는것이 인터페이스 디자인 Output이다.

이 장은 원래 3.4 장인 문서 하단에 갑자기 등장하여, 어쩔수 없이 문서의 흐름을 원활하게 하기 위하여 변경하였다.

### 3.2.1.1. Component Interface

EpackDownloader Interface

public interface EpackDownloader{

public EpackServerInfo getEpackServerInfo(ConfigProperties) throws Exception;

public TestEngineInfo getEngineInfo(RestAPI URL Value Object) throws Exception;

public List<TestedEngineFile> getStoredEngineList(SearchConfdition Object) throws

Exception;

}

SampleDownloader Interface

public interface SampleDownloader{

public SampleServerInfo getSampleServerInfo(ConfigProperties) throws Exception;

public List<TestedSampleFile> getSampleInfoList(RestAPI URL Value Object) throws

Exception;

}

EngineTestExecutor Interface

public interface EngineTestExecutor{

public Boolean isEngineTestJobRunning() throws Exception;

public List<TestResult> getJobResultList(RestAPI URL Value Object) throws Exception;

public void setJobInfo(SearchCondition Object) throws Exception;

public JobInfo getJobInfo(SearchCondition Object) throws Exception;

public JenkinsMasterInfo getJenkinsMaster(SearchCondition Object) throws Exception;

public void executeJob() throws Exception;

}

#### 3.2.1.2. ~~Module Interface~~

Component 하단의 Module 간 발생할 수 있는 모든 Interface Line에 대한 정의

모듈은 가장 상위에 위치하는 구현의 단위, 컴포넌트는 런타임 엔티티를 참조하는 단위라고 생각하면 금방 그 차이를 이해 할 수 있을거라고 생각된다. 따라서 모듈과 컴포넌트는 상위와 하위관계를 명확히 구분짓기 어렵고 서로 다른 개념으로 바라보아야 한다. 그렇기 때문에 모듈 1000개가 모여 하나의 컴포넌트가 될 수도있고, 컴포넌트 1000개가 모여서 하나의 모듈을 구성 지을 수도있다.

쉽게 설명해서, 모듈이란 실질적으로 구현이 된 단위를 의미한다. 반면, 컴포넌트는 실제적으로 동작하고있는 엔티티로써 활동중인 독립적인 단위를 중점적으로 보고 있다.

예를들어 1개의 서버에게 서비스를 제공받는 100개의 클라이언트가 존재한다고 가정하자.

위에 설명한 내용으로 모듈, 컴포넌트의 개수를 각각 세어보면 서버가 구현된 모듈 1개, 클라이언트가 구현된 모듈 1개이므로 이 시스템 인프라의 총 모듈 개수는 2개이다. 컴포넌트의 경우 실제 동작하고 있는 엔티티를 의미하므로 총 101개가 된다. [출처]

Clements, Paul, et al. Documenting software architectures: views and beyond. Pearson Education, 2002.

<

?xml version="1.0" encoding="UTF-8"?

>

<

project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema

-

instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-

v4\_0\_0.xsd">

<!-- pom Setting -->

<modelVersion>4.0.0</modelVersion>

<groupId>com.ahnlab.asd</groupId>

<artifactId>ETS</artifactId>

<version>19.08.02</version>

<packaging>pom</packaging>

<name>standard-project</name>

<description>ETS ( Engine Test System ) with TSF. by AhnLab</description>

<url>http://ras.ahnlab.com</url>

<inceptionYear>2011</inceptionYear>

<organization>

<name>AhnLab</name>

<url>http://ras.ahnlab.com</url>

</organization>

<!-- -->

<modules>

<module>web-module</module>

</modules>

따라서, Module Interface Design은 상기 용어 설명으로 갈음한다.

3.2.1.3. ~~Process Interface~~

프로세스의 호출과 통신 관점에서 발생할 수 있는 모든 Interface Line에 대한 정의

3.2.1.4. ~~N/W Interface~~

Network 의 연결 관점에서 발생할 수 있는 모든 Interface Line에 대한 정의

#### 3.2.1.5. ~~H/W & Device Interface~~

Hardware Device 의 연결 관점에서 발생할 수 있는 모든 Interface Line에 대한 정의

3.2.2. ~~Internal Software Data Structure~~

3.2.3. ~~Global Data Structure~~

3.2.4. ~~Temporary Data Structure~~ 3.2.5. ~~Permanent Data Structure~~

내부 Component에서 사용되는 Data를 기술한다.

구조체(Structure)

상수 기타

객체지향 프로그래밍 언어를 사용하는 프로젝트에서는, 컴포넌트란, 잘 동작하는 기능단위를 말하며, 컴포넌트에서 사용하는 데이터의 단위는 VO ( Value Object ) 또는 DTO ( Data Transfer Object )를 의미한다. 이 데이터 단위 표시는 데이터베이스 ERD와 연관관계를 가지게 된다.

ERD와 Mapping 관계를 가지지 않는 VO혹은 DTO는 Class 명세 또는 데이터딕셔너리에 표시되기 때문에

3.2.2, 3.2.3, 3.2.4, 3.2.5 절은 적시하지 않기로 한다.

3.2.6. ~~Database Description~~ 영구적으로 사용되는 Data에 대해 기술한다.

영구적으로 사용되는 데이터는 데이터를 관리하는 시스템인 Database system에 해당한다. 또한 데이터베이스 스키마의 데이터를 Mapping 하는 VO, DTO로 보통 표기되기에 ERD와 VO관계를 파악하는 것으로 갈음할 수 있다. 따라서, ERD, 데이터딕셔너리와 Class 다이어그램으로 갈음한다.

뿐만아니라, 3.2.7 절과 구분명이 동일하다. ( 적시할 이유가 없는 2번째 이유이다. )

## 3.2.7. Database Description

### 3.2.7.1. ERD(Entity Relationship Diagram)

ERD는 클래스 상세화를 통하여 어떤 데이터를 데이터베이스에 적제해야 할지 결정 한 후 작성될 수 있다. 데이터 설계단계는 각 컴포넌트의 메소드 인터페이스에서 활요할 데이터객체 ( VO, DTO )가 정의된 후 상세화 및 명세가 가능하다. 따라서, 클래스 다이어그램이 작성된 후 프로젝트 종료시점에 적시하도록 한다.

### 3.2.7.2. Data Dictionary

데이터 딕셔너리란 대부분 읽기 전용으로 제공되는 테이블 및 뷰들의 집합으로 데이터베이스 전반에 대한 정보 제공을 의미한다. 예를 들면 오라클의 사용자정보, 권한과 롤 정보, 스키마 객체 정보와 무결성 제약조건에 대한 정보이다. 따라서, 이 정보는 프로젝트 종료시점에 적시하도록 한다.

3.3. ~~Architectural Design → 이런 단어는 소프트웨어 공학에 없습니다.~~

설계자의 판단에 따라서 작성하는 내용입니다. 중요 Function 이나 중요 상태에 대해 기술합니다.

3.3.1. ~~Use Case Diagram~~ 3.3.2. ~~Class Diagram~~ 프로젝트 종료 시점에 export 하여 작성하도록 한다.

3.3.3. ~~Flow Chart~~

3.3.4. ~~DFD(Data Flow Diagram) - SRS 작성시 요구사항 발제자와 커뮤니케이션 도구~~

Data Architecture 와 ERD로 대치한다.

3.3.5. ~~Sequence Diagram~~

3.4. ~~User Interface Design~~

3.4.1. ~~Screen images~~

3.4.2. ~~Screen Object and Actions~~

ETS 본 프로젝트는 UI 가 제공되지 않는 시스템이므로 본 장을 작성하지 아니한다.

단, 추후 고도화가 필요할 경우 wireframe을 활용하여 작성하기로 한다.

3.5. ~~Component Design (& Module Design)~~

반드시 작성되어야 하는 내용입니다. 각 Component 또는 Module 별로 상세한 내용을 설계하며, 각 모듈에 대해 다음과 같은 상세 항목을 설명합니다.

Module ID

Module Name

Input parameters and the order passed

Returns(output parameters) and the order returned

Modifies(list of variables, local and global, this module modifies)

Restrictions/limitations

Is-calles-by(list of modules that call it)

Calls(list of modules called by this module)

Algorithmic model(pseudo code)

Local data structures

3.1.4 장의 Component 정의 설명을 보면, 다음과 같다.

소프트웨어 구성요소인 각 Component와 그 하위 module을 정의하고, 각각의 기능 및 의존관계 등을 기술합니다.

각 Component 또는 Module 의 Public 함수들에 대한 Function Prototype 은 "3.4 Interface Design" 또는 "3.6 Component Design" 에 반드시 기술 되어야 한다.

이는 모듈 단위까지 정의가 되며, 사용하는 메소드 정의까지 망라되 있다. 각 메소드의 인터페이스 디자인을 작성하기 위한 Input에 해당한다. 마땅히 잘 작성되어야 하는 절이다.

3.5 장까지 작성되고나면 컴포넌트 정의부터 모듈 정의 ~ 메소드 정의를 위한 인터페이스 디자인까지 되어있는 Skeleton 프로그래밍이 가능한 명세상황 에서

3.6 장에서 다시 컴포넌트 디자인을 하라는것은 무엇을 적시해야 하는지 명확하지 않다. ( 설명도 동일하다 ) 따라서, 본 3.6 장은 작성하지 않기로 한다.

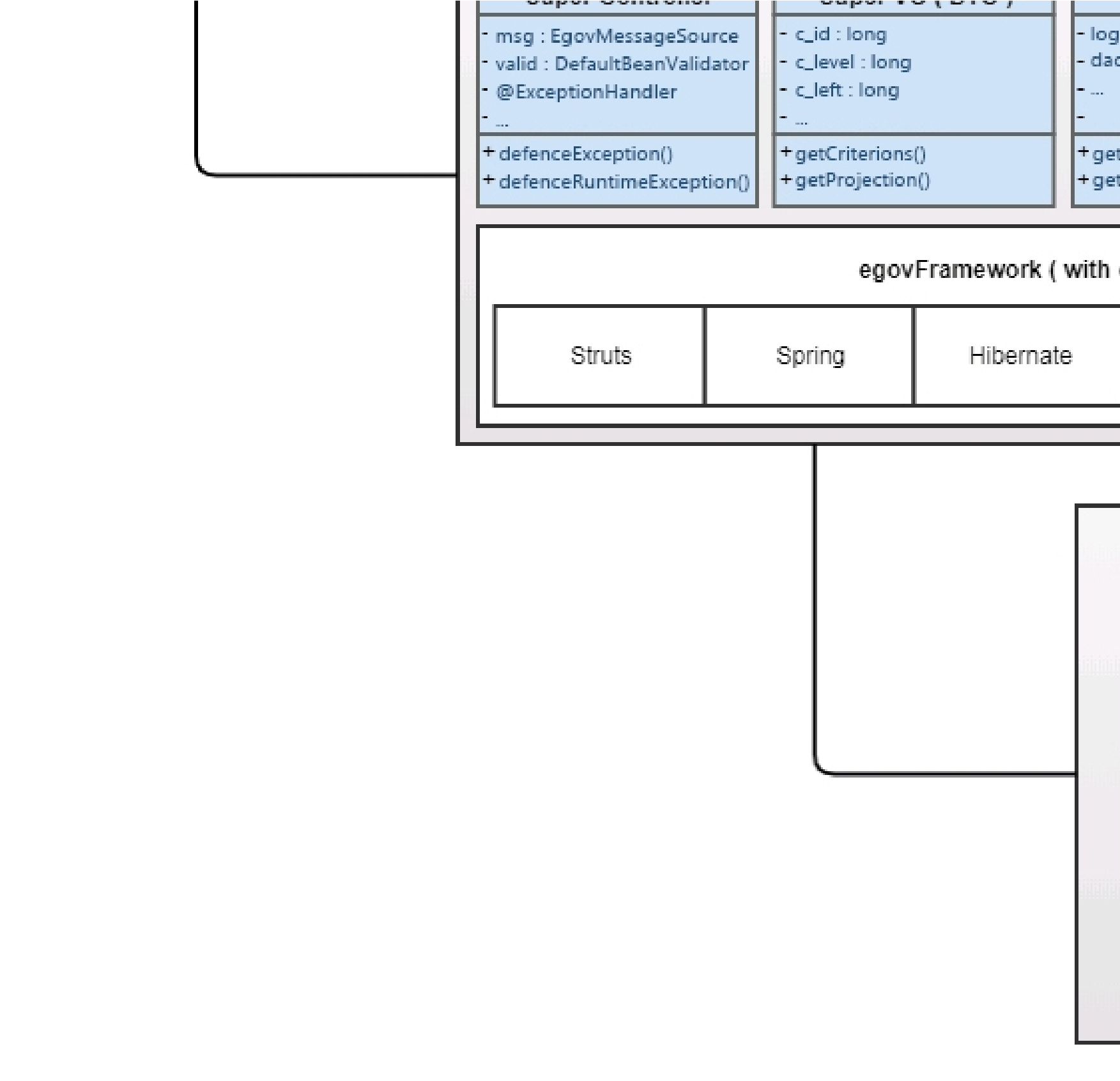
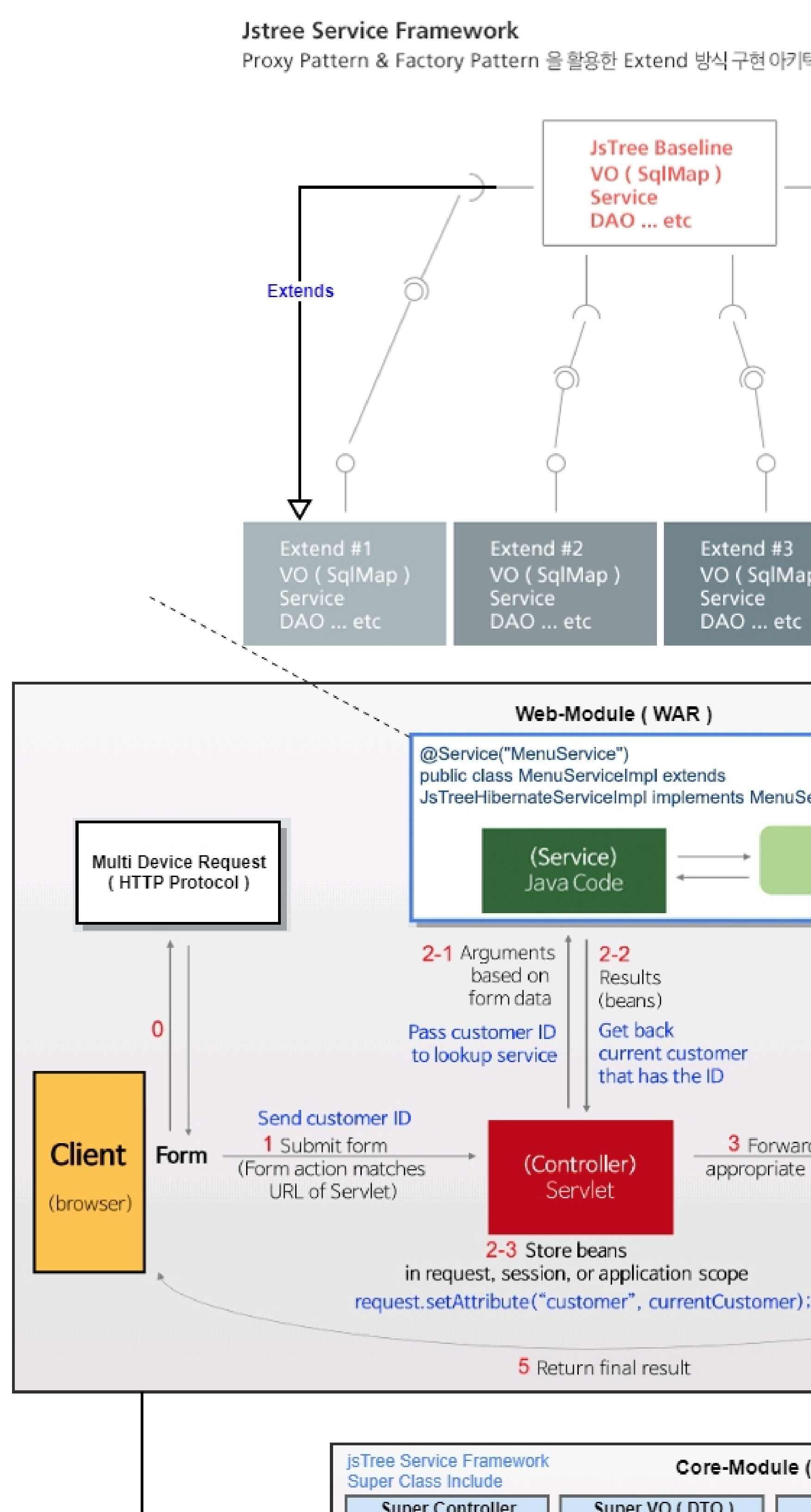
# 3.6. 기타

## 3.6.1. 소스트리 구성

반드시 작성이 되어야 하는 내용입니다. 소스트리 위치 및 디렉토리를 어떻게 구성할 것인지에 대해 기술한다.

ETS ( Engine Test System )은 시스템의 엔트로피를 줄이고 재사용성을 극대화하기 위하여 PLE ( Product Line Engineering ) 기법을 사용한 다. 기본 구조는 Maven 프로젝트 관례를 따르며 다음과 같은 기본 구조를 가지고 있다.

하기는 ETS의 프로젝트 기본 구조를 표시한다.



소스 트리 구성

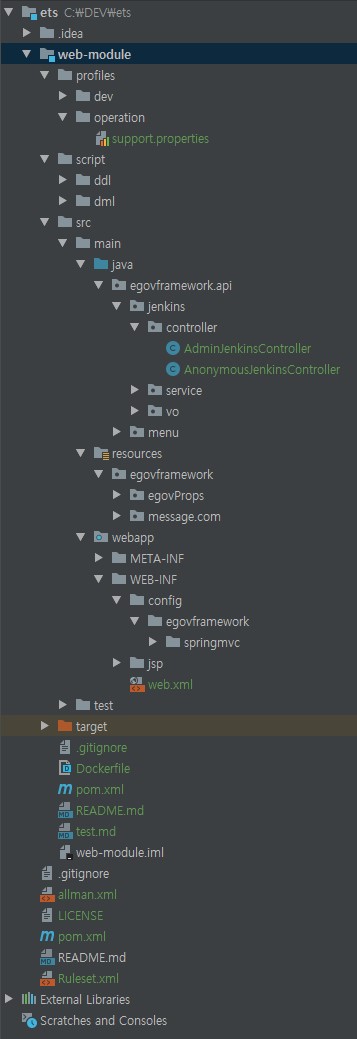
구분

설명

본 프로젝트는 Maven 프로젝트 구조를 사용하고 있으

며, 이에 준하는 관례를 활용한다.

분의 최하위 ( egov.api . com.



Profiles

dev : 개발 모드로 프로젝트를 빌드 할수도 있

으며, Profile properties Load 폴더

operation : 운영 환경에 배포하기 위한

Profile Properties Load 폴더

Script

ddl : 클래스 패키지 단위로 Data Definition

Language를 작성할 폴더

dml : ddl에 의해 스키마 생성후 기본 데이터

삽입이 필요한 경우의 폴더

src ( 소스 트리 최상위 root )

main ( TDD를 활용할 경우 test code를 분기

하기 위한 기본 폴더 )

Java ( Java 와 Scala 등 여러 언어를 구

분하기 위한 분기 폴더 )

egovframework ( 첫번째 패키지 )

com ( 보통 사용하는 그룹 네임 :

com.ahnlab ~ )

api 목적에 맞도록 패키지 구

ahnlab.api )

Resources

다 )

을 로드

webapp

록 구성

test : TDD 를 활용하여 서비스 구성을 할 첫

번째 분기 폴더 )

Dockerfile : pom.xml 에 의하여 docker로 전환할 수 있

도록 구성

pom.xml : application 을 개발 및 배포까지의 관리용

파일

본 프로젝트는 eclipse 및 intelij IDE tool load test 를

완료한 상태이며,

openjdk 및 oraclejdk 1.8 기준 compile 및 project

setting 을 모두 호환 합니다.

IDE load 시 : eclipse:eclipse 명령으로 load

개별 빌드 시 : mvn clean install

빌드 배포 시 : mvn clean deploy

를 사용합니다.

본 페이지는 PLE에 따라서 문서를 좀더 정교하게 구현

하여 배포하는것을 목표로 합니다.

component 대표명

controller : mvc 패 턴의 컨트롤러 구분

service : 실제 비지

니스 로직이 구현

vo : model 코드

egovframework ( 설정 정보는 전자 정부표준프레임워크를 기반으로 한 com 혹은 ahnlab 으로 지정하여도 로드 된다. 보통 사용되는 리소스

properties 및 message 파일

META-INF : 메타파일을 관리 WEB-INF : web.xml 을 사용하여 웹

어플리케이션 구조를 관리

config : springmvc 를 활용하

는 파일 로드

( 이후 추가 설정파

일을 차례로 로드 )

jsp : 기본 리턴 api server 모 드 이나, 필요에 따른 jsp needs에 따라 활용 가능하도

## 3.6.2. 빌드 설계

반드시 작성이 되어야 하는 내용입니다.

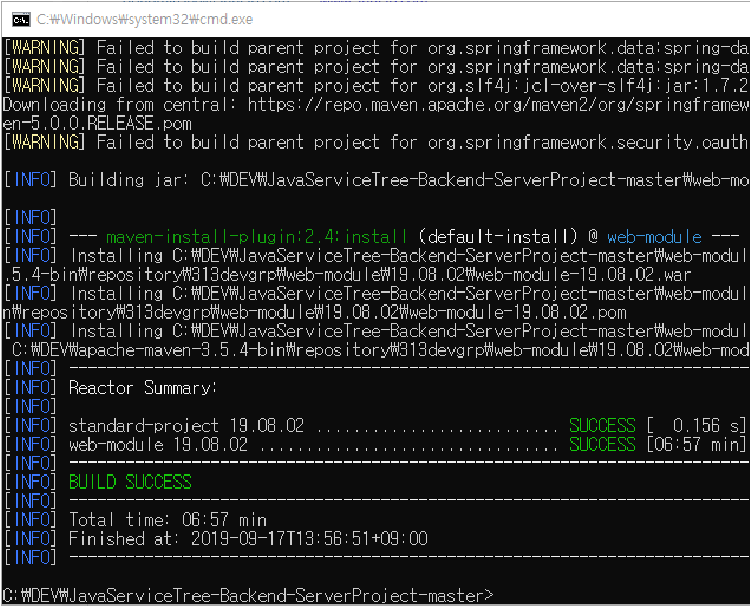
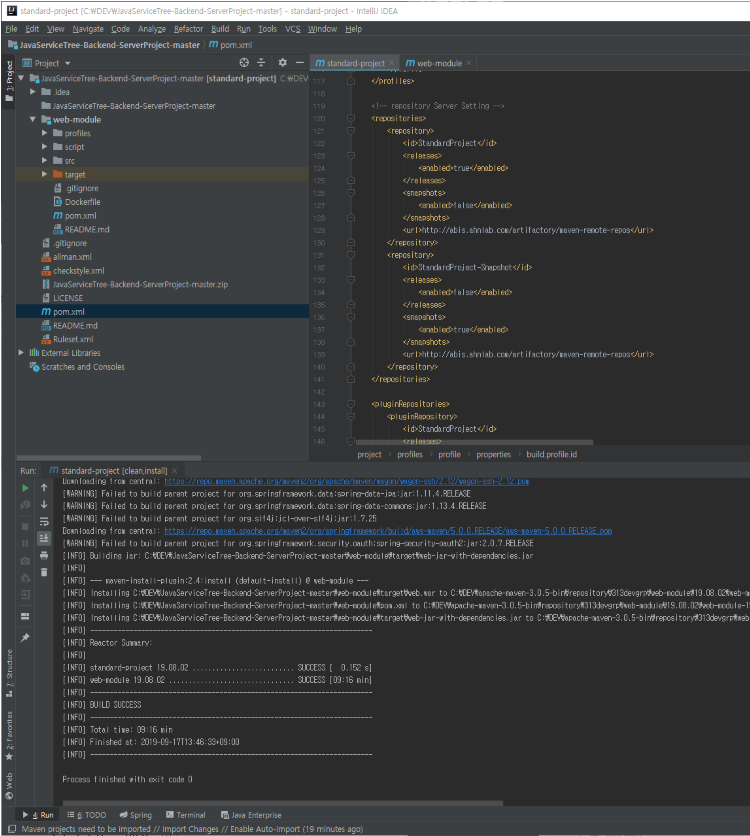
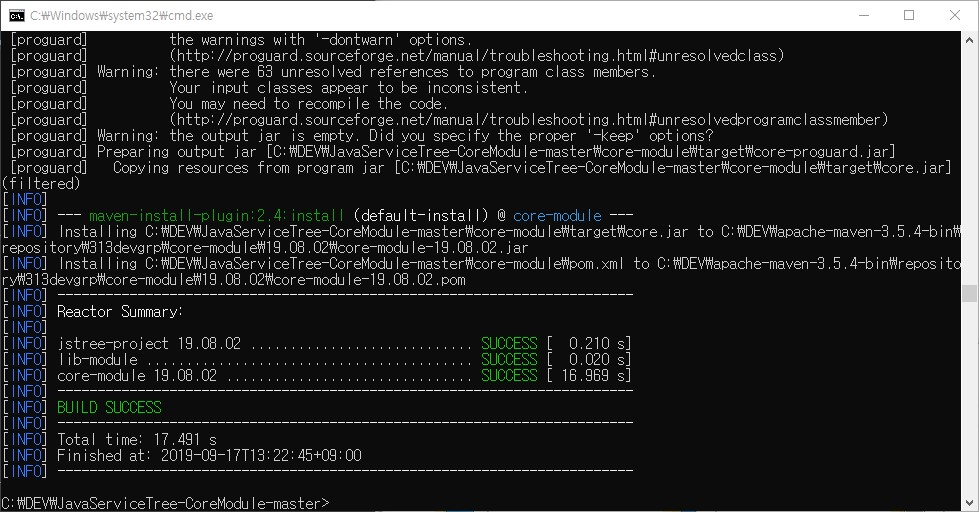
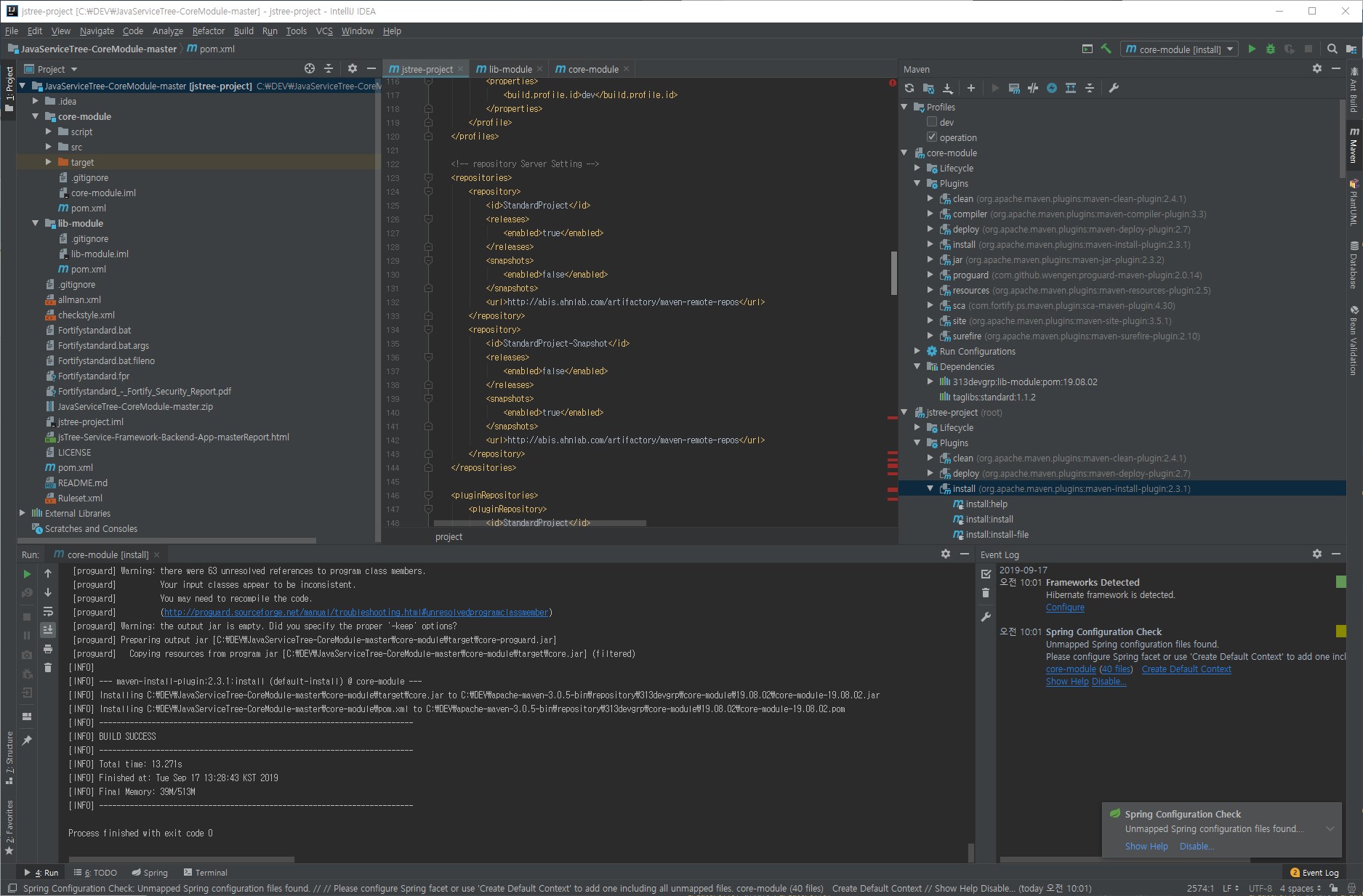
Build를 위해 사전에 설치되어야 할 라이브러리 종류, 버전 등을 포함한 Build에 필요한 다양한 instructions 을 기술한다.

Core-Module IDE 로컬 빌드 및

Console 빌드 ( 빌드 머쉰 및 운영환경 빌드 동일 증명 )

Web-Module IDE 로컬 빌드 및

Console 빌드 ( 빌드 머쉰 및 운영환경 빌드 동일 증명



## 3.6.3. ~~설치 설계~~

반드시 작성이 되어야 하는 내용입니다. 설치할 파일 리스트 및 설치 위치, 설치 시 필요한 정보, 설치 User Interface 등에 대해 설명한다. 본 프로젝트는 설치를 염두해 둔 클라이언트 프로젝트가 아니다.

정형화된 서버에 설치하는 서버사이드 아티팩트를 산출하는 프로젝트이므로 본 장은 작성하지 않는다.

## 3.6.4. 배포 설계

3.6.4.1. ~~마스터 구성~~ 마스터의 내/외부 구성형태를 구체적으로 기술한다.

3.6.4.2. 배포 및 업데이트 방법 최초 배포 및 설치 방법을 기술한다.

ABIS 및 WAS 배포 및 업데이트 방안 설명

Maven의 다음설정을 통하여, 어플리케이션 Artficat인 war 파일을 ABIS에 Deploy 한다.

어플리케이션 배포 방안

<distributionManagement>

<repository>

<id>ETS</id>

<url>http://abis.ahnlab.com/artifactory/maven-remote-repos</url>

</repository>

<snapshotRepository>

<id>ETS-Snapshot</id>

<url>http://abis.ahnlab.com/artifactory/maven-remote-repos</url>

</snapshotRepository>

<site>

<id>ETS-Site-Public</id>

<url>ftp</url>

</site>

</distributionManagement>

Maven의 다음 플러그인 설정을 활용하여, WAS에 war 파일을 HotDeploy 합니다.

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<executions>

<execution>

<phase>deploy</phase>

<goals>

<goal>redeploy</goal>

</goals>

</execution>

</executions>

<configuration>

<charset>UTF-8</charset>

<uriEncoding>UTF-8</uriEncoding>

<url>http://tomcat/manager/text</url>

<server>ETS-Snapshot</server>

<path>/</path>

<protocol>org.apache.coyote.http11.Http11NioProtocol</protocol>

</configuration>

</plugin>

3.6.4.3. ~~업데이트 방법~~ 배포 이후, 패치 및 업데이트 시에 필요한 파일 및 정보, 방법 등을 구체적으로 기술한다.

## 3.6.5. 테스트 설계

개발자 입장에서 필요한 테스트 방법 및 예상결과 등에 대해서 기술한다.

본 프로젝트는 TDD 개발 방법론을 사용하며,

Database 관점의 unit test는 DBUNIT 을 활용하여, TSF ( jsTree Service Framework) layer의 TEST Coverage를 확장하고, 비지니스 로직 부분을 구현한 web-module의 경우 특정한 분기로직은 100% Coverage를 가지도록 한다.

3.6.5.1. ~~Classes of Test~~

블랙박스 테스트, 화이트박스 테스트를 비롯한 요구되는 테스트 종류 및 단계적인 테스트 방법을 기술한다.

3.6.5.2. ~~Expected Software Response~~

테스트케이스를 도출하고, 단계적인 테스트 방법을 기술한다. 이에 대한 나와야 하는 테스트 결과를 예상하여 기술한다.

3.6.5.3. ~~Perfomance Bounds~~

테스트되어야 하는 성능측면의 요구사항을 기술한다.

### 3.6.5.4. ~~Identification of Critical Components~~

테스트에서 특별한 주의가 요구되거나 Critical 한 사항에 대해서 기술한다.

SRS의 6.1.1 엔진테스트 시연 방법을 통한 시나리오 테스트를 수행한다.

SRS의 6.1.2 엔진업데이트 시연 방법을 통한 시나리오 테스트를 수행한다.