Документ за софтуерната архитектура

Version <1.0>

История на промените

|  |  |  |  |
| --- | --- | --- | --- |
| **Дата** | **Версия** | **Описание** | **Автор** |
| 23/01/2016 | 1.0 | Създаване на документа | Адриан Данаилов, Мартин Абрашев, Калоян гецов |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Съдържание

1. Въведение 4

1.1 Цел 4

1.2 Обхват 4

1.3 Дефиниции, Акроними и Абревиатури. 4

1.4 Препратки 4

1.5 Кратък преглед 4

2. Представяне на архитектурата 4

2.1 Бюджет 4

2.2 Наличие на “legacy” код 4

2.3 Географско разположение на системата 5

2.4 Обем на данните 5

2.5 Връзка с чужди системи 5

2.6 Сигурност 5

3. Архитектурни цели и ограничения 5

3.1 Използвани технологии 5

3.1.1 SVN = GIT 5

3.1.2 Maven – dobavqme 5

3.1.3 FUCK JAVA BEANS 5

3.1.4 Entity beans – ORM – trqbva ni 5

3.1.5 JBoss = Apache (трябва да се провери точно) 5

3.1.6 SOAP services – трябват ни, ние ще използваме Java Spring – и с по-малко обяснения 5

3.2 Устойчивост на данните 5

3.3 Сигурност 6

4. Use-Case View 6

5. Logical View 6

5.1 Overview 6

5.2 Architecturally Significant Design Packages 6

5.3 Use-Case Realizations 6

6. Process View 6

7. Deployment View 6

8. Имплементационен поглед 7

8.1 Overview 7

8.2 Layers 7

9. Data View (optional) 7

10. Size and Performance 7

11. Качество 7

Документ за софтуерната архитектура

# Въведение

## Цел

Целта на този документ е да опише софтуерната архитектура на ABM. Той покрива както хардуерa, който е нужен за пълното и функциониране, така и софтуера който ще се използва за реализацията на различните под системи. Архитектурата дава ясна представа на екипа разработващ системата какви стъпки трябва да поеме той, както и предотвратява възможността от възникването на по – нататъшни проблеми за системата.

## Обхват

Документът включва

## Дефиниции, Акроними и Абревиатури.

Всички акроними, дефиниции и абревиатури може да се видят в документа:

ABM-I1-4-Glossary.doc

## Препратки

В този документ ще има препратки към следните документи:

* ABM-I1-5-SoftwareRequirementSpecifications.doc

## Кратък преглед

В следващите точки се разглеждат основните фактори, които се взимат в предвид при изграждането на една система и решенията на проблемите които повдигат те. Технологиите планирани за използване при реализацията и.

Във втората част от документа ще се разгледат различните погледи над системата съпътствани с диаграми и кратки обяснения.

# Представяне на архитектурата

За изготвянето на архитектурата на тези система са взети под предвид следните фактори:

* Софтуерните изисквания, описани в документа за спецификация на допълнителните изисквания, зададени от възложителя на проекта;
* Нефункционални изисквания, описани в гореспоменатия документ;
* Географското разположение на системата;
* Обемът на данните с който системата работи;
* Нуждата от връзки с чужди системи;
* Бюджета нужен за реализация на проекта;
* Живот на системата;
* Наличие на “Legacy” код;
* Подобни системи.

## Бюджет

Взимайки в предвид възможностите на фирмата разработчик и договора сключен с фирмата възложител технологиите и хардуерът нужен за пълното осъществяване на проекта е комбинация от платени и безплатни услуги. Поради силната зависимост от хардуерни технологии на системата тук място за компромис не се допуска. В точки 2.2 до точка 4 са описани всички технологии нужни за реализация на системата.

## Наличие на “legacy” код

Създаването на тази система ще се осъществи без наличен код на”legacy” код от предходни системи, който програмистите да надграждат. Това означава, че фирмата изпълнител поема пълна отговорност за жизнения цикъл на системата и поведението и. Въпреки липсата на “legacy” код голяма част от функционалностите на системата стъпват на чужди разработки и технологии. Банкоматните системи са добър пример за чужда разработка, тук единствено трябва да се контролира диалога между двете системи за да се гарантира тяхната ефективност.

## Географско разположение на системата

Взимайки в предвид големината на проекта и разположението на системата в цяла Европа. Топологията на мрежата, която отговаря най – добре на нашите цели е “Mesh network”. Тя предлага най – надеждна връзка до всяка една точка като потенциално може да загуби скорост за сметка на успешно доставени данни.

Осъществяването на вътрешните мрежи ще се осъществява с топология от тип шина, която предлага бързина, надеждност и ниски разходи. Централният клон ще играе ролята на главен управленчески орган, при настъпване на промени по системата той ще изпраща информация към по – малките клонове.

## Обем на данните

Според количеството на данните който ще постъпват и ще се обработват се налага те да се съхраняват с “Data center” технологии. Тази технология позволява съхранението на данните на един сървър, който предлага високо ниво сигурност спрямо пробив на данните, както и създаване на архив на файловете. Една от най – важните му характеристики е способността да се надгражда което позволява обемът на съхранените да надвишава далеч над нормалното.

## Връзка с чужди системи

Поради спецификата на работата на тази система се налага осъществяването на връзки с други системи. Най – наложилият се принцип на осъществяване на връзки с други системи е чрез сервизна шина(Enterprise Service Buss). Тя не само позволява общуването между различни приложения и системи, но и надеждно ниво на защита. Чрез нея ще може да се осъществят връзки като с държавни системи така и с други банки или “smart phone” приложения. Въпреки завишените разходи за поддръжката и на фона на конкурентността на други фирми използването и е задължително.

## Сигурност

Компютрите в различните клонове на банката ще се достъпват чрез VPN. Това ще добави допълнително ниво на сигурност към системата, като позволява ползването на ресурси по мрежата единствено чрез попълване на акаунт и парола и преминаване на проверка за MAC адрес.

# Архитектурни цели и ограничения

## Използвани технологии

### GIT

GIT е безплатна система, с отворен код, за контрол на версийте. GIT е предназначен да се справя бързо и ефективно, както с малки проекти, така и с проекти с огромни размери. Системата е лесна за научаване, изисква много малко ресурси и е светкавично бърза. Надминава съперниците си в този клас – Subversion, CVS, Perforce и други, с функционалности като евтина поддръжка на клони (cheap local branching), удобна зона за зареждане на файлове преди запис (convenient staging areas) и други.

### Maven

Maven е инструмент за автоматизиране на билдовете при работа с Java проекти. Работи чрез използването на XML файл, в който е описана информацията вънпните зависимости на билда (external dependencies). Инстурментът позволява контрол над реда на билдване, директорийте и плъгините.   
Друга част от функционалностите на Maven е възможността да се свалят динамично външни API, библиотеки, модули, и други, като след сваляне те се съхраняват в локално хранилище.

В текушия проект, в средата за разработка, на сървъра е създадено хранилище, което да се използва за стабилните версии на подмодулите и за пускане на билдове.

### FUCK JAVA BEANS

### Entity beans – ORM – trqbva ni – 6TE POLZVAME HIBERNATE

### JBoss = Apache (трябва да се провери точно)

### SOAP services – трябват ни, ние ще използваме Java Spring – и с по-малко обяснения

## Устойчивост на данните

В проекта ABM информацията ще се съхранява под формата на база от данни и файлова система.

Основен приоритет е информацията да се запазва по надежден начин, гарантиращ целостта и устойчивостта й.

Съхранените данни трябва да могат да бъдат достъпвани, модифицирани и добавяни по определени от проекта критерии. Използвания DBMS ще е MySQL, за драйвер ще се ползва MySQL Connector/J driver.

## Сигурност

Основен компонент на системата е сигурността. За осигуряването на контрола на достъп ще бъде разработен модул, които да позволява подържането на информация за регистрираните (оторизирани) потребители. Модула ще е базиран на фраймуорка Spring Security. Достъпа до системата може да се осъществи само от оторизирани потребители, а конкретните права за достъп до ресурсите се контролират от ролите на потребителя.

Следните възможности на системата трябва да се разработят:

* Аутентификация – влизане в системата с потребителско име и парола
* Оторизация – достъп до ресурсите за различните потребители, задава се посредством роли.
* Конфиденциалност – възможност за съхраняване и пренасяне на данните в криптиран вид.

Целокупност на данните (data integrity) – поддържане на коректна и консистетна база от данни защитена от неоторизиран достъп и външни намеси както по време на предаването й по мрежата така и по време на нейното съхранение.

Използваните технологии разполагат с голям брои вградени защити срещу хакерски атаки и неоторизиран достъп.

# Use-Case View

[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]

# Logical View

[This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.]

## Overview

[This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.]

## Architecturally Significant Design Packages

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

## Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality.]

# Process View

[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]

# Deployment View

[This section describes one or more physical network (hardware) configurations on which the software is deployed and run. It is a view of the Deployment Model. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs) that execute the software and their interconnections (bus, LAN, point-to-point, and so on.) Also include a mapping of the processes of the **Process View** onto the physical nodes.]

# Имплементационен поглед

[This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.]

## Overview

[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers. ]

## Layers

[For each layer, include a subsection with its name, an enumeration of the subsystems located in the layer, and a component diagram.]

# Data View (optional)

[A description of the persistent data storage perspective of the system. This section is optional if there is little or no persistent data, or the translation between the Design Model and the Data Model is trivial.]

# Size and Performance

[A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.]

# Качество

Софтуерната архитектура ясно очертава възможностите на системата, както и позволява интегрирането на нови системи към нея.

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]