Упражнение по Софтуерни архитектури

***Архитектурни Стилове и Изгледи***

*Станислав Стоянов, 76 група, 471218066*

# Задача 1

Изберете два от архитектурните стилове които коментирахме на лекцията и потърсете в Интернет примери за архитектури, които се базират на тях. Според вас защо е избран този подход и какви предимства дава той?

***Model-View-Controller (MVC)***

*Източник:* [*https://visualstudio.microsoft.com/*](https://visualstudio.microsoft.com/)

*Предимства, които дава съответния архитектурен стил:*

*Конкретно тази платформа представлява Multi-Page Application и основните предимства са, че лесно може да се извършва поддръжка на системата и нейните компоненти. Също така търсещите машини и алгоритмите на Google може лесно да правят SEO. Защо е за предпочитане този архитектурен стил? Защото позволява на разработчиците да използват само консистентни езици и технологии, пример ASP.NET CORE за back-end частта и JavaScript за front-end частта. MVC е за предпочитане да бъде ползвано при средно големи проекти и не е подходящ за малки приложения, тъй като може да се стигне до писане на много код.*

***Microservices***

*Източник:* [*https://www.amazon.com/*](https://www.amazon.com/)

*Предимства, които дава съответния архитектурен стил:*

*За разлика от монолитните приложения и SOA (Service-Oriented Architectures), микросървисите са от скоро на международната сцена (началната им разработка започва през 2010г.). Amazon е компанията, която е една от първите мигрирали към този архитектурен стил. Тази архитектура е базирана на множество малки приложения, които си взаимодействат с HTTP, REST или дори Web API услуги. Основното предимство е, че всички тези малки приложения може да комуникират директно. От Amazon споделят, че преминавайки към този архитектурен стил, разработчиците лесно могли да локализират проблемите във всяко едно малко приложение и да предоставят бързи решения като намалят времето за изчакване от страна на потребителите. За пример тук най-удачната архитектура за ползване е именно тази, защото при използване на MVC в проект като Amazon поддръжката на цялата система би била невъзможна, а да не говорим и за производителността.*

# Задача 2

Банките са оправдано предпазливи по отношение на сигурността в системите им. Кой архитектурен стил (или комбинация от тях) бихте избрали за разработването на система за онлайн банкиране и защо? Покажете го и с една проста Component&Connector диаграма.

*Бих избрал да ползвам MVC + Microservices архитектура, защото от една страна MVC дава необходимата централизация и защитеност на данните, лесен е за поддръжка и позволява на разработчиците да правят бъдещи подобрения в системата. View-та могат да бъдат използвани в различна презентационна логика и системата да има както онлайн приложение така и платформа за Android/IOS. От друга страна при евентуално разрастване на системата може да се прибегне към използване на микросървиси с отделни база данни, които в комбинация с MVC да поемат по-тежък трафик от страна на потребителите и да осигурят по-бързо изпълнение на заявките. Естествено основен приоритет ще бъде сигурността на системата и защитата на потребителските данни.*

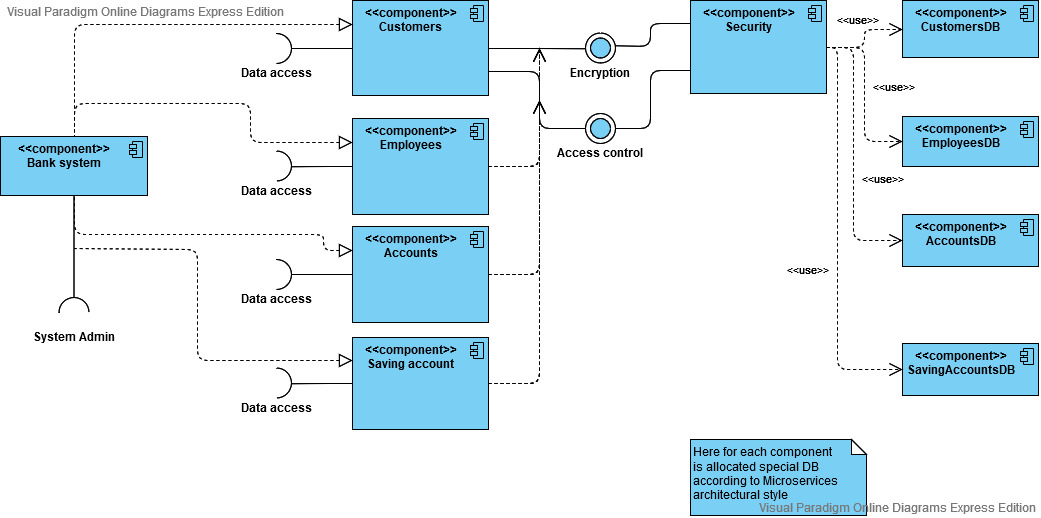
**

Figure 1 - component diagram

# Задача 3

Да предположим, че вашата компания е закупила друга компания и сте получили задача да обедините ваша система с подобна в другата компания. За системата знаем че използва комбинация от Client-Server, Shared-Data и Microservice архитектурни стилове. Вие отговаряте за това каква информация ще се поиска от другата компания относно системата им. Но различните заинтересовани лица (*stakeholder*) се интересуват за различни изгледи. Попълнете таблицата според това колко е важен всеки изглед за всеки от списъка. Може ли да се комбинират някои от колоните (след попълването)? При положение че другата компания няма готова такава документация с какъв приоритет бихте я поискали и защо?

*Може да се комбинират ролите на Performance Analyst, Availability Analyst и Security Analyst в една обща роля Analyst, защото различните типове анализатори споделят едни и същи изгледи. Ако другата компания няма достатъчна информация за C&C изгледите, то би могло да се получат проблеми при обединението на двете системи. Също така е съществено важно имплементацията да бъде правилно обединена, за да се получи един общ изглед т.е за правилно конфигуриране на двете системи е необходима документация както за back-end частта, така и front-end частта на системата. Относно приоритета на изгледите – най-важни според мен за системата са Module Views и C&C Views, защото цялата система стъпва на тяхната структура, докато при Allocation Views единствено е важно приложението да бъде правилно деплойнато, т.е достъпно за крайния потребител.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ***Module Views*** | | | ***C&C Views*** | | | ***Allocation Views*** | | |
| **Stakeholder** | **Decomposition** | **Uses** | **Class Diagram** | **Client-Server** | **Shared-Data** | **Microservice** | **Deployment** | **Implementation** | **Work Assignment** |
| *Developer* | ED | ED | ED | ED | ED | ED | SD | SD | SD |
| *Project Manager* | N | N | O | SD | SD | SD | N | N | ED |
| *Tester* | N | N | N | SD | SD | SD | SD | SD | O |
| *System Admin* | N | N | N | SD | SD | SD | D | SD | SD |
| *User* | N | N | N | N | N | N | N | SD | N |
| *Maintainer* | O | O | O | D | D | D | O | O | SD |
| *Performance Analyst* | O | O | N | O | O | O | D | D | D |
| *Availability Analyst* | O | O | N | O | O | O | D | D | D |
| *Security Analyst* | O | O | N | O | O | O | D | D | D |
| *Deployment team* | O | N | O | SD | SD | O | ED | D | D |
| **ED**: *extensive details*, **D**: *detailed* *information*, **SD**: *some* *details*, **O**: *overview*, **N**: *not needed* | | | | | | | | | |

# Бонус задача

Изберете една open-source система и посетете уебсайта. Там потърсете архитектурната документация и се опитайте да отговорите на следните въпроси. Кой архитектурен стил е използван според документацията? Относно предоставената документация за архитектурата какво смятате че липсва или не е достатъчно добре документирано и как би се отразило това върху способността ви да добавите нова функционалност?

*Система:* [*https://www.gimp.org/*](https://www.gimp.org/)

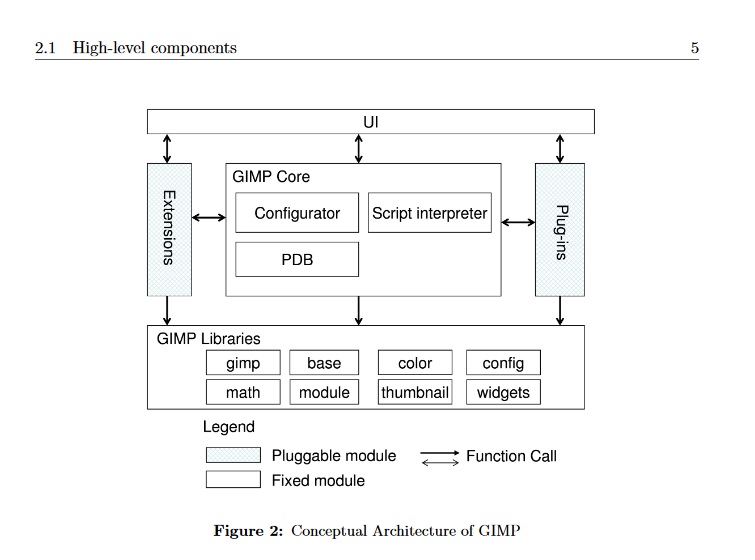
*Информация за системата от нейната документация:*

*“*The GIMP is a multi-platform raster graphics editor. GIMP is an acronym for GNU ImageManipulation Program. The GIMP is suitable for a variety of image manipulation tasks,including photo retouching, image composition, and image construction.”

Архитектурата на системата може да бъде декомпозирана на **5** архитектурни компонента:

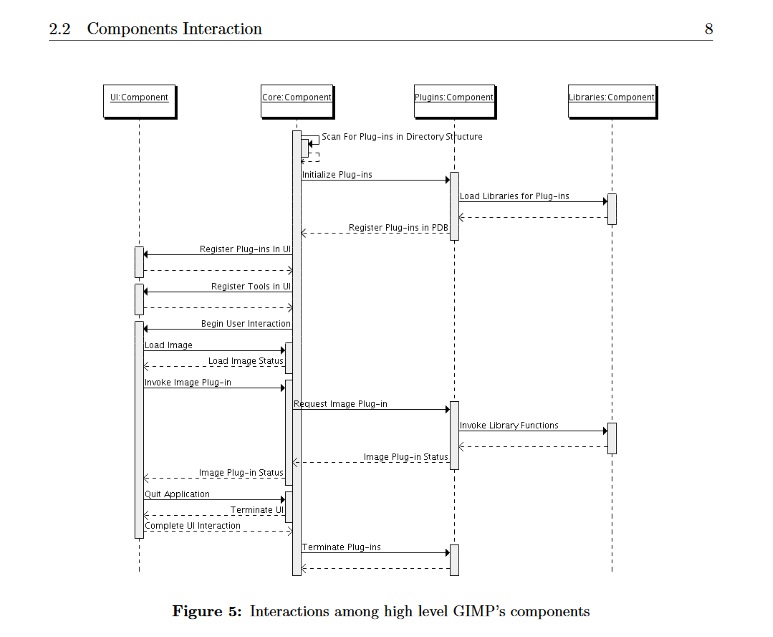
* Потребителски интерфейс
* Разширения (Extensions)
* GIMP ядро (Core)
* Плъгини
* GIMP библиотеки

На следната диаграма са показани взаимодействията между отделните компоненти на високо ниво.

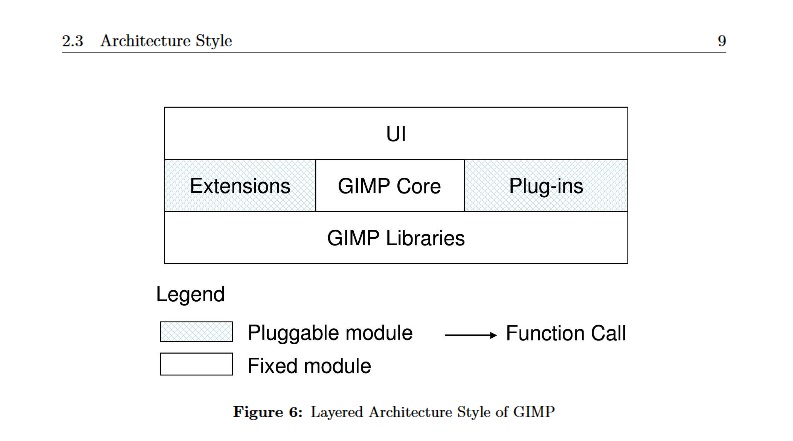


Обуславяйки се на документацията основната функционалност на системата е имплементирана чрез помощта на разширения (extensions) и плъгини.

На диаграмата на последователностите (sequence diagram) отново много добре може да се види взаимовръзката между отделните компоненти на системата.



В заключение може да се каже, че системата има Layered архитектурен стил със следните силно изразени слоя – Библиотеки (Libraries), Обработка на изображенията (Image Manipulation) и потребителски интерфейс (User Interface).



Като цяло ядрото на системата е написано на C и относно документацията няма какво да се каже – доста подредена, структурирана и лесна за четене. Напълно непознат човек може лесно да разбере основните концепции, които са имплементирани. Няколко думи и за опциите за разширяемост на системата от други разработчици според предоставената документация. Модулният дизайн позволява функционалността на системата да се променя линейно, тъй като броят на разработчиците, допринасящи за проекта се увеличава. Разработчиците създават нови функционалности за проекта, като паралелно разработват свои собствени модули. Тези модули обикновено се зареждат от софтуера динамично, при поискване, за да осигурят допълнителната функционалност. Тяхната интеграция е прозрачна за крайния потребител, т.е незабележима. Този архитектурен стил може да се наблюдава в много мащабни проекти с отворен код като:

* Apache
* Mozilla
* Sendmail

# Допълнителни материали, свързани с упражнението можете да откриете тук

Точка 1.2 от следния адрес <http://jz81.github.io/course/sa/Software%20Architecture%20in%20Practice%20(3rd).pdf#page=30>

Точка 3.5 от следния адрес <http://jz81.github.io/course/sa/Software%20Architecture%20in%20Practice%20(3rd).pdf#page=73>

Точки 13.1 и 13.2 от следния адрес <http://jz81.github.io/course/sa/Software%20Architecture%20in%20Practice%20(3rd).pdf#page=225>

<http://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf>