

**Технически Университет – София**

**Факултет Приложна Математика и Информатика**

**Катедра Информатика**

**АРХИТЕКТУРЕН ПРОЕКТ**

**За**

**Cooking Hub**

**Имена на студентите:** Станислав Бисеров Стоянов, Атанас Иванов Парасков, Димитър Рачев Пашов

**Проектна група:** 16

**Дата:** 26.10.2020г.

**Факултетни номера:** 471218066, 471218069, 471218034

**Съдържание:**

1. Въведение
2. Предназначение на избраната тема

2.1. Обхват на проекта

2.2. Избрани актьори

1. Основни цели на архитектурата - описание на няколко ключови изисквания и ограничения на разработвания софтуерен продукт, които оказват влияние на избраната архитектура
2. Архитектурен обзор

4.1. Описание на основни сценарии (use-case) и актьори (actors), които са важни за разработваната архитектура, чрез use-case диаграми

4.2. Описание на логическия изглед на архитектурата. Клас диаграми за илюстрация връзките между архитектурно значими класове, подсистеми, пакети и слоеве. Диаграми на състоянието за илюстрация конкретни процеси на определени роли.

4.3. Изглед на процесите. Описание на отделните изпълними процеси (подсистеми) и зависимостите между тях.

4.4. Изглед на внедряването (Deployment view)

4.5. Изглед за разработка (Implementation view)

1. Обосновка на това как избраната архитектура осигурява адекватна реализация на поставените нефункционални изисквания
2. Използвани термини и символи
3. Използвани източници

***1. Въведение (Станислав Стоянов и Димитър Пашов)***

С напредването на софтуерните технологии, в наши дни все по-лесно става намирането на информация за различни дейности свързани с бита на модерния човек. Обръщайки внимание на определени необходимости на отделните индивиди, даденият проект цели да представи пример за изготвяне на обществена поръчка за приложение свързано с храната.

Настоящият документ представя софтуерната архитектура служеща за изготвяне на уеб приложение свързано с кулинарията и здравословният начин на живот. Чрез представяне на съществено важните функционални и нефункционални изисквания, представената документация играе ключова роля за правилното разбиране и осмисляне сложността на зададената система.

За изготвянето на подобен проект ще бъде използван архитектурният шаблон за програмиране **MVC** ([Model-View-Controller](https://bg.wikipedia.org/wiki/Model-View-Controller)), като всеки един негов аспект ще бъде подробно описан в настоящия документ.

***2. Предназначение на избраната тема (Станислав Стоянов и Димитър Пашов)***

В дадения документ се представя основната част за изготвянето на проекта, а именно изготвянето и използването на подходяща архитектура за софтуерна разработка. Представяйки определената структура на софтуера, се предоставя началната точка, необходима на софтуерните разработчици да започнат изграждането на продукта. Тази начална точка е с особена тежест и характер, тъй като по определената документация софтуерните разработчици получават опорен план, който след започване на разработката трябва постепенно да бъде надграждан. Също така таргетираната група определя основното предназначение на избраната тема, а именно крайните потребители с интерес към кулинарството и приготвянето на здравословна храна. Освен предоставяне на разнообразни и вдъхновени рецепти от известни готвачи в сферата, **Cooking Hub** ще допринесе за развитието на подобен род системи, осигурявайки на своите потребители необходимата точна информация, лесна достъпност и най-вече интерактивност по отношение приготвянето на храна.

***2.1. Обхват на проекта*** ***(Станислав Стоянов)***

Проектът е насочен към всички крайни потребители с интереси в света на кулиранството. Поради уникалността и разнообразността си, в глобален мащаб проектите много трудно могат да бъдат формализирани. Въпреки това може да се каже, че всеки проект се вписва в един стандартен жизнен цикъл със следните фази:

* Иницииране
* Планиране
* Изпълнение
* Контролиране
* Закриване

Представената архитектура се вписва основно във фазата планиране и служи като опорен стълб за изпълнението на следващата фаза и реалното проектиране на проекта.

***2.2 Избрани актьори*** ***(Станислав Стоянов)***

Основните актьори, участващи в описаната архитектура ще бъдат подробно представени в т.4 “Архитектурен обзор” чрез необходимите use-case UML диаграми, клас диаграми и диаграми на последователността и състоянието, тъй като те неизменно участват в бизнес процесите на системата. За повече дейталност и конкретност се представят следните актьори, участващи активно във всички основни функционалности на **Cooking Hub**:

* Администратор
* Модератор
* Обикновен регистрирал се потребител
* Обикновен нерегистрирал се потребител
* Потребител с ограничен достъп
* Потребител с липса на пълен достъп (невъзможност за използване на системата под каквато и да е форма)

***3. Основни цели на архитектурата - описание на няколко ключови изисквания и ограничения на разработвания софтуерен продукт, които оказват влияние на избраната архитектура (Станислав Стоянов)***

Уеб системата за кулинария, за по-кратко навсякъде ще бъде използвана абревиатурата от т.6 “Използвани термини и символи” **CH – Cooking Hub** предоставя възможност за регистрация на крайните потребители, осигурявайки им редица полезни функционалности като:

* Създаване/редакция/изтриване на собствени рецепти
* Преглед на рецептите, написани от останалите потребители
* Създаване на ревю и даване на оценка на рецепта с цел обратна връзка
* Създаване на различни статии, групирани по категории
* Достъп до глобалния чат за обмен на всеобхватна информация по преценка на потребителите

Налице е също така подробна информация за всеки един потребител, изразяваща се под формата на личен профил, където се съхранява профилната снимка на съответния потребител, дата на присъединяване към CH, статистика относно създадените рецепти/ревюта/коментари/статии, информация за използвания провайдър за регистрация и логин било то CH или Facebook. Системата има уеб базиран потребителски интерфейс като избраният архитектурен стил ще бъде MVC ([Model-View-Controller](https://bg.wikipedia.org/wiki/Model-View-Controller)), което ще разреши евентуални бъдещи проблеми свързани с производителността, скалируемостта и ефективността на системата. Този архитектурен шаблон най-често се използва при създаването на потребителски интерфейс. Той „разделя“ приложението на три взаимосвързани части. MVC шаблонът разделя тези главни компоненти, което позволява на разработчиците да използват отново вече написан код по-ефективно. Традиционно MVC шаблонът се използва за графични потребителски интерфейси GUI, но вече той се използва както и за уеб приложения, така и за мобилни такива. Както личи от името “Model-View-Controller”, шаблонът се състои от три отделни компонента, които изпълняват определена роля и са свързани в определена последователност.

* „Моделът (Model)“ е централен компонент в шаблона. Той е динамичната структура от данни на приложението, независима от потребителския интерфейс. Моделът управлява данните, логиката и правилата на приложението. Конкретно в нашата софтуерна системата всички модели представляват таблиците в релационната база данни.
* „Изгледът (View)“ е изходящият поток от информация (това, което приложението изпраща като отговор до дисплея, респективно – до потребителя, в следствие на неговата заявка). Възможни са няколко различни изгледа на една и съща информация. Чрез изгледът се получава рендериране на HTML страницата и по този начин потребителят вижда резултата от изпълнението на своята заявка.
* „Контролерът (Controller)“ е третата част от този шаблон. Той приема потребителския вход (т.е. данните, които потребителя въвежда, неговите заявки и т.н.) и ги преобразува в команди към модела или изгледа. Контролерът е сърцето на една MVC архиктура и той е отговорен за редица неща – Request, Response, HttpContext, RouteData, TempData, ModelState, User, ViewBag/ViewData. Той си комуникира със сървисите, които му предоставят готова бизнес логика и съществено важно е един контролер да няма директна комуникация с моделите в базата данни. Контролерът отговаря за това дали потребителят ще “види” даден изглед, дали ще бъде препратен към друга уеб страница (друг контролер) и дали изобщо ще получи отговор от дадената заявка.

Предимства на MVC:

Тъй като MVC разделя основните компоненти на приложението, това ни позволява да работим паралелно по различни компоненти, без те да си оказват влияние или да си пречат един на друг. Например разработката на приложението може да се раздели на две части, като едната поема изграждането на back-end логиката, а другата – на front-end частта. Back-end разработчиците проектират структурата на данните и как потребителят взаимодейства с тях, без да се нуждаят от потребителски интерфейс (front-end), за да тестват и усъвършенстват тези аспекти на приложението. Обратно – front-end разработчиците могат да проектират и тестват изгледа на приложението, преди да имат готова структура от данни, с която той да борави. Създавайки компоненти, които са независими едни от други, ние може да използваме многократно един компонент в различни приложения. В случая бизнес логиката в сървисите може да бъде преизползвана в други проекти на готово. Изгледът на едно приложение може да се преработи бързо и лесно за друго сходно приложение, което борави с коренно различна структура от данни. Моделът, който обработва данните и който представлява цялостната структура в БД е отделен компонент спрямо изгледа и това ни дава свобода да използваме наново един и същи код, според нуждите си.

* Ключови изисквания

Основните ключови нефункционални изисквания, които трябва да има, са спазване на атрибутите за качество, много добре дефиниран дизайн, ориентиран към потребителя и спазване на различни методи за проектиране и използване на потребителския интерфейс (Usability Heuristics for User Interface Design[[1]](#footnote-1)). На първо място от съществено значение е да бъде взето под внимание времето за реакция на системата (responsiveness), защото това гарантира, че потребителят ще изживее максимално добри чувства ([Emotional design](https://www.interaction-design.org/literature/topics/emotional-design)) докато ползва системата. Дългите закъснения при обработка на заявките трябва да бъдат елиминирани. По-конкретно уеб приложението трябва да отваря бързо в рамките на милисекунди всяка една страница. На следващо място идва сигурността (security), която гарантира, че сензитивните потребителски данни ще бъдат обработвани и използвани само от CH. В това число влиза потребителската парола, пол на потребителя, евентуално предоставен мобилен номер и имейл за връзка с него. Цялата биологическа информация за потребителя не трябва да бъде публично достъпна и използвана без неговото разрешение. Също така не трябва да бъде пренебрегвана надеждността на системата (reliability). Този атрибут за качество гарантира, че средното време между евентуални откази (Mean Time Between Failure MTBF[[2]](#footnote-2)) на системата ще бъде максимално голямо и това може директно да бъде свързано с потребителското изживяване описано по горе.

Добре дефинираният потребителски ориентиран дизайн и спазването на различни методи за проектиране и използване на потребителския интерфейс като: видимост на състоянието на системата, тоест дали в момента тя работи или се извършва някаква поддръжка, съвпадение между системата и реалния свят, постоянство, гъвкавост и ефективност на използването и евентуално възстановяване след грешка, причинена от потребителя са следващите ключови изисквания, които CH трябва да реализира. За да може атрибутите за качество от бизнес гледна точка да бъдат спазени, то трябва много детайлно да бъдат решени въпроси като това дали системата изпраща необходимата обратна връзка на своите потребители под формата на различни известявания (email, контактна форма), дали има съвпадение между очакванията на потребителя за функционалността и реалната такава. На първо място, не е коректно, когато потребителят очаква да получи отговор на изпратено запитване чрез контактната форма и да се получава забавяне и липса на такъв отговор. Особено значим момент е и системата да информира достатъчно добре потребителя за новости по начина на нейното използване, нови статии, както и когато се напише ревю от друг потребител относно дадена рецепта. Постоянството от своя страна представлява потребителят бързо да се ориентира в използването на приложението и да не се чуди кой бутон да натисне или коя страница да отвори, а това се постига единствено със следване на определени конвенции от страна на разработчиците (спазване на КПК – Качество на Програмния Код, SOLID принципи и правилно наименуване на класове и методи). При настъпване на евентуални грешки във CH, потребителят трябва да има възможност за изпращане на контактна форма с конкретния проблем или за по-добро потребителско изживяване Live chat, реализиран с технологията SignalR, където да може конкретно да си зададе въпроса, а от другата страна човек по поддръжка на системата да съдейства максимално бързо. Заложените функционалности при разработката на системата имат за цел да осигурят изброените до тук изисквания.

* Ограничения

Основното ограничение за реализация на системата е времевият фактор. Преди създаването и имплементацията на всички изброени ключови изисквания е необходимо да се разпредели максимално добре времето за имплементация на отделните модули на системата. От атрибутите за качество, тук ясно изразен фактор е съвместимостта на системата (compatiblity), т.е до колко добре системата позволява оперативна съвместимост с по-стара наследена система. Предвид заявения архитектурен стил, системата ще се използва главно в уеб средите и няма да зависи от конкретната операционна система на потребителя. Този атрибут може да бъде пренебрегнат с цел намаляване на разходите и увеличаване на предоставеното време за реализация, като в това число влиза и дизайнът на CH. По-добре е да се отделят повече средства за производителност и надеждност, както и хардуерен ъпгрейд, отколкото за уникално красиво изглеждащо приложение. Друго ограничение е и тестваемостта на системата и до колко добре тя ще бъде покрита с необходимите стрес/load[[3]](#footnote-3) и regression[[4]](#footnote-4) тестове. Трябва да бъдат взети под внимание само тестовете свързани с производителността и надеждността, като целта е да бъде изтествана основната функционалност (при MVC влизат сървисите, отговарящи за бизнес логиката). Освен изброените тестове, ще бъдат реализирани и основните тестове на по-ниско ниво, а именно unit[[5]](#footnote-5) тестове чрез използване на готови външни библиотеки като xUnit/nUnit. И последното съществено ограничение е времето за проектиране и кодиране на цялата система, защото има определени срокове за реализация. Представете си, че определен поръчител има необходимост от създаване на подобна система и се налага бърза реализация на приложение, което да бъде достъпно от всякакви устройства. Ако системата ни бъде бързо проектирана и създадена, то шансовете ни за спечелване овациите и уважението на поръчителя при евентуална продажба на CH са много по-големи.

***4. Архитектурен обзор***

За начало на обзора на съответната архитектура се обръща внимание на нейните изгледи. Водейки се по избрания архитектурен шаблон (MVC), в даденият документ се представят някои от основните изгледи необходими на разработчиците да изградят по-ясна представа за начина на функциониране на системата, както и на участниците, играещи определена роля в нея.

***4.1. Описание на основни сценарии (use-case) и актьори (actors), които са важни за разработваната архитектура, чрез use-case диаграми***

Чрез следните use-case диаграми представяме съществено важната функционалност на основните роли в системата.

* Администратор

*TODO*

* Модератор

*TODO*

* Обикновен регистрирал се потребител

*TODO*

* Обикновен нерегистрирал се потребител

*TODO*

* Потребител с ограничен достъп

*TODO*

* Потребител с липса на пълен достъп (невъзможност за използване на системата под каквато и да е форма)

*TODO*

***4.2. Описание на логическия изглед на архитектурата. Клас диаграми за илюстрация връзките между архитектурно значими класове, подсистеми, пакети и слоеве. Диаграми на състоянието за илюстрация конкретни процеси на определени роли.***

* Клас диаграми

*TODO*

* State diagram (диаграма на състоянието) за конкретен процес

*TODO*

***4.3. Изглед на процесите. Описание на отделните изпълними процеси (подсистеми) и зависимостите между тях.***

Представяме изгледа чрез две възможности на реализация на UML диаграмите за последователност (Sequence diagrams) и активност (Activity diagrams).

* Диаграми за последователност

*TODO*

* Диаграми за активност

*TODO*

***4.4. Изглед на внедряването (Deployment view)***

Този изглед е представен чрез две възможности на реализация на UML Deployment диаграми, което ще подпомогне инженерът да конфигурира по-бързо необходимите сървърни среди и средата, където ще се инсталира системата. Също така е описано разположението на отделните модули на CH, тоест уеб сайтът за уеб достъп до системата и общата бизнес логика на приложения сървър заедно с базата данни.

* Диаграма 1

*TODO*

* Диаграма 2

*TODO*

***4.5. Изглед за разработка (Implementation view)***

Изгледът на разработка описва основните класове на отделните модули на системата. За представянето му сме използвали два варианта на UML Package диаграми.

* Диаграма 1

*TODO*

* Диаграма 2

*TODO*

***5. Обосновка на това как избраната архитектура осигурява адекватна реализация на поставените нефункционални изисквания (Станислав Стоянов)***

Първото нефункционално изискване, което нашата софтуерна система трябва да покрива, е производителността, или по – точно системата ни да извършва обработка на потребителски заявки в нормално възприетото за това време (2 секунди), дори и при многократно нарастване на броя едновременно работещи с нея потребители. Това значи, че ние трябва да осигурим мощен сървър за нашата система, който да може да се справя с огромен наплив от потребителски заявки, без отговорите му да закъсняват с повече от 2-3 секунди. Конкретно когато дойде време за деплоймънт и **CI/CD** можем да използваме безплатните услуги на Microsoft Azure. Всъщност производителността е пряко свързана с друго нефункционално изискване, което системата трябва да покрие, а именно скалируемостта – нашата система трябва да позволява лесно преконфигуриране на отделните модули при нарастване на обема на потребителски заявки с цел запазване на определената производителност. За да се постигне такава скалируемост, е необходимо да се ръководим по небеизвестните **OOP** принципи, **SOLID** концепции, **КПК**, които ще позволят както лесното евентуално фактуриране кода на отделните модули в **MVC**, така и лесното добавяне на нова функционалност към вече съществуващата. В тази връзка навременното писане на тестове ще подсигури евентуалната сигурност, когато стане време за промяна или ъпгрейд на съществуващата функционалност. Друго нефункционално изискване, което системата ни трябва да покрие е сигурността – CH трябва да гарантира, че чувствителни данни няма да бъдат достъпни за трети лица. Това ще може да се постигне чрез строго определените роли в системата, които могат да се дават само от администратор. Достъп до потребителските данни ще имат само администраторите. Възможно решение, което ще подсигури сигурността на системата, е да се добави така наречената Two-Factor Authentication. Това може да се постигне, като изградим софтуера така, че при всяко влизане на потребител, независимо с каква роля, да се изпраща СМС или имейл с код (цифри), който да бъде нужен за успешното влизане. Ще бъдат възможни двете опции, тъй като незадължително условие е потребителят да предоставя своя мобилен номер при регистрация в системата. Последното нефункционално изискване, което системата ни трябва да покрива, е отказоустойчивостта, тоест системата ни да продължи да работи коректно, дори и при отпадане поради някаква авария на някои от основните сървъри. Това може да се постигне чрез закупуването на така наречения backup сървър, който да се включва само при възникването на дадена повреда в основния сървър, за да може да не се наруши работата на системата за неопределен период от време. За реализация на тази отказоустойчивост, може да се използват предоставените услуги на Microsoft Azure. За целите на съответния проект няма да бъде реализирана такава опция, но е важно да се вземе предвид при евентуално инвестиране или отстъпване правата на софтуера на дадена компания.

***6. Използвани термини и символи (Станислав Стоянов)***

* CH – Cooking Hub
* OOP – Object-Oriented Programming
* SOLID – Single responsibility principle, Open/closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle
* КПК – Качествен програмен код
* MVC – Model-View-Controller
* GUI - Graphical User Interface
* CI – Continuous Integration
* CD – Continuous Delivery

***7. Използвани източници (Станислав Стоянов)***

* лекции и презентации за UML диаграмите, архитектурни стилове и атрибути за качество от курса “Софтуерни архитектури”
* Даниел Митев, Дизайн, ориентиран към потребителя (User Centered Design). Основни етапи при проектирането на ДОКП
* Даниел Митев, Методи за проектиране и използване на потребителски интерфейс (Usability Heuristics for User Interface Design). Схематичен дизайн
* знания от предишни курсове (ООП, СУБД, СА)
* за информацията под чертите са използвани препратки от Wikipedia

1. Основните методи за проектиране и използване на потребителски интерфейс са създадени през 1990г. от Джакопо Нилсен в колаборация с Ролф Молих. [↑](#footnote-ref-1)
2. MTBF = MTTF (Mean Time To Fail) + MTTR (Mean Time To Repair) [↑](#footnote-ref-2)
3. Стрес/Load тестовете представляват симулация на голям поток от потребителски заявки и гарантират надеждност и липса абнормалност на системата [↑](#footnote-ref-3)
4. Regression тестовете осигуряват това, че поправен бъг в системата няма да се повтори отново в друг компонент [↑](#footnote-ref-4)
5. **Unit testing**-ът е метод за тестване, който проверява дали индивидуалните единици от сорс код работят правилно. Единица или един юнит е най-малката част от едно приложение, която може да се изтества. [↑](#footnote-ref-5)