*Архитектурен проект*

*Тема: „EduWatch“*

Изготвили:

Проектна група № 13:

Тодор Пъков, 78гр., фак. №: 471218003

Александрина Михова, 78гр., фак. №:

471218032

Доротея Пикова, 78гр., фак. №: 471218043

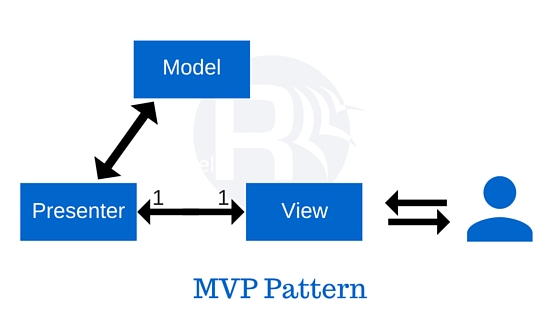
Галина Георгиева, 77гр., фак. №: 471218072

27.10.2020г.

1. **Основни цели на архитектурата**

За приложението “EduWatch” е избран Model–view–presenter модел на архитектура. Той е разновидност на добре познатия Model–View -Controller (MVC) модел. MVP е удачен за използване при създаването на потребителски интерфейси. Архитектурният стил разделя приложението на три градивни основни блока: Model, View, Presenter. Важна част от изграждането им е те да бъдат максимално разделени един от друг, за да не се нарушават принципите на архитектурата.

Всеки един от структурните блокове има строго определени функции. Програмата ще работи с потребителски интерфейс, който ще бъде свързан с другите градивни части на архитектурата.



1.View :

Изгледът е отговорен за взаимодействането с потребителите. Той не съдържа особена логика. Предназначението му е да насочва потребителските входове и команди към Presenter. Препратката на View към Presenter е важна и ключова част от изработването на архитектурата.

2.Presenter :

Presenter е междинната част на модела. Той е връзката между View и Model. В него е внедрена цялата бизнес логика като например : логиката на данните, валидиране на въведени данни, получаване на събития от View, преобразуване на данни от View в Model и обратно.

3.Model :

В Model се съхраняват обекти и услуги заедно с техните данни. Обектите съдържат данни, които се съпоставят с таблица в база данни. Информацията в този случай за преподаватели, ученици, предмети оценки и други свързани с тях сведения ще бъдат съхранявани в релационна база данни, от която ще се извличат и преобразуват в конкретни обекти чрез Object-Relational Mapping (ORM). Тези обекти ще са част от “Model” частта на програмата и ще бъдат достъпвани само по предварително определен похват.

Отделните части на модела са слабо свързани /loose coupling/. По този начин отделните компоненти имат частична независимост един от друг, което позволява по-лесно тестване, добавяне на нови данни, преконфигуриране на вече съществуващите.

Presenter играе ролята на връзката между View и Model. Обработва заявките на отделните потребители като основната му цел е достъп единствено до информацията , което е нужна на даденото лице.

**Особености на архитектурата:**

View и Model не си взаимодействат директно. Това подобрява имплементацията на View.

Архитектурата дава възможност за промяна на потребителския интерфейс от Web към Windows Forms или Mobile.

Висок риск крие Presenter. Той може да се разшири до огромен всезнаещ клас, ако се наруши Single Responsibility Principle. Нарушаването на този принцип допълнително води до огромно завишение на сложността.

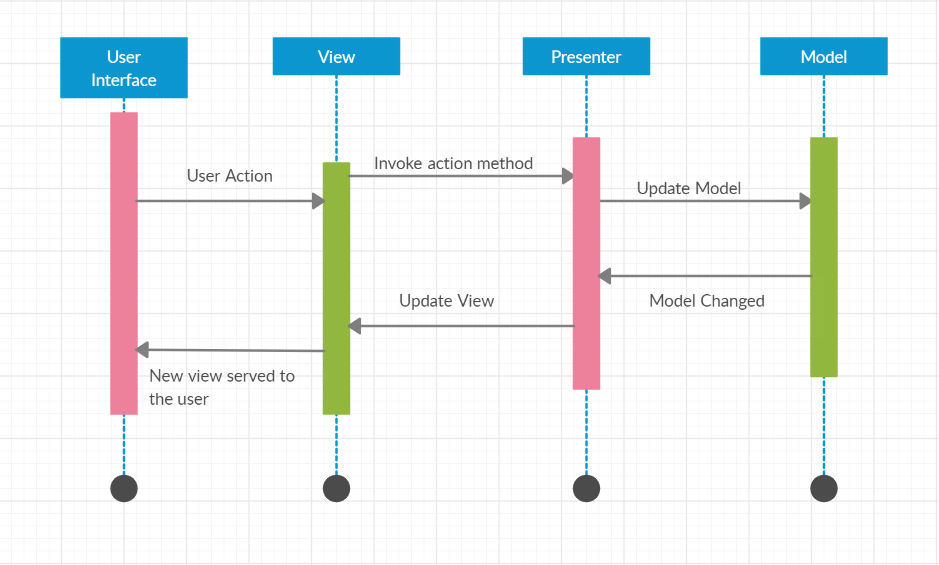
Важно е да се следва последователността при изготвянето на софтуерния продукт, за да се избегнат възможните проблеми и за да може крайният продукт да е висококачествен и максимално оптимизиран.

1. **MVP и Windows Forms**

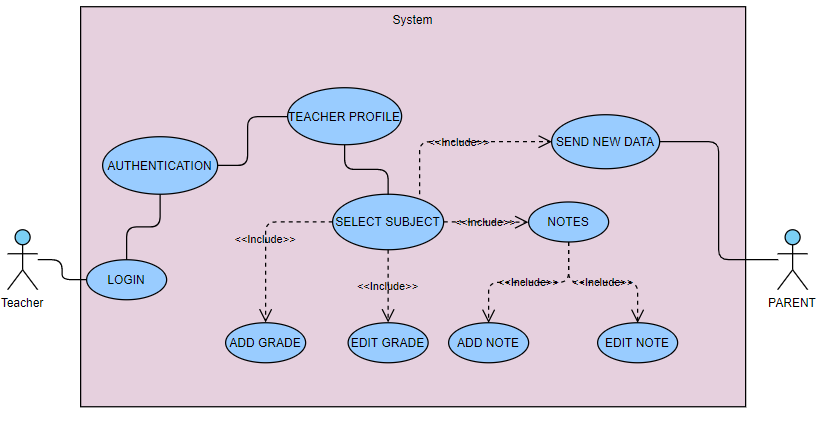
Основен проблем при използването на МVC-базирани модели е използването на потребителския интерфейс. Често се наблюдава, че той извършва повече функционалности отколкото би трябвало да са му възложени. Липсата на ясно разпределение между модулите довежда до това, че потребителският интерфейс изземва голяма част от логика принадлежаща на останалите слоеве.

MVP решава тази често допускана грешка. Неговата цел е да остави на потребителския интерфейс само кода и логиката, които са му необходими. По този начин се постига изграждането на рационализиран и защитен код за многократна употреба , който е лесен за тестване.

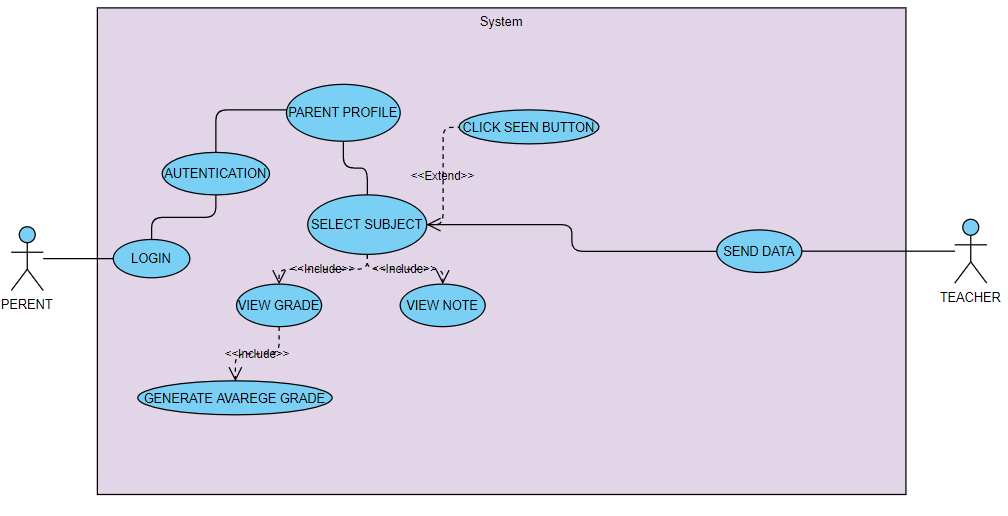
1. **Логически изглед**



Следната диаграма показва работата между отделните компоненти на приложението. Потребителския интерфейс праща заявка към View. View показва, че е извикан метод за действие от Presenter. Работата на Presenter е да се извлече нужната информация от Model, като това става чрез извикване на метод за бизнес логиката. Model получава заявката за извличане на данни от Presenter и му връща нужната информация, която Presenter препраща към View. View актуализира новия изглед към потребителския интерфейс. Диаграмата строго демонстрира работата и отношенията на всеки един модул от MVP архитектурата, без излишни стъпки и с точно разпределение на всеки един случващ се процес. Важно е да се отбележи, че последователното изпълнение в случая води до пълно улеснение при използване от масов потребител. Основните три модула са скрити от обикновения ползвател, така че достъпът да му до данни, несъществени за неговата работа, да не бъде осъществим. Това е основна цел за приложенията свързани с масово или неспециализирано ползване –потребителя получава само услугата, от която има нужда.

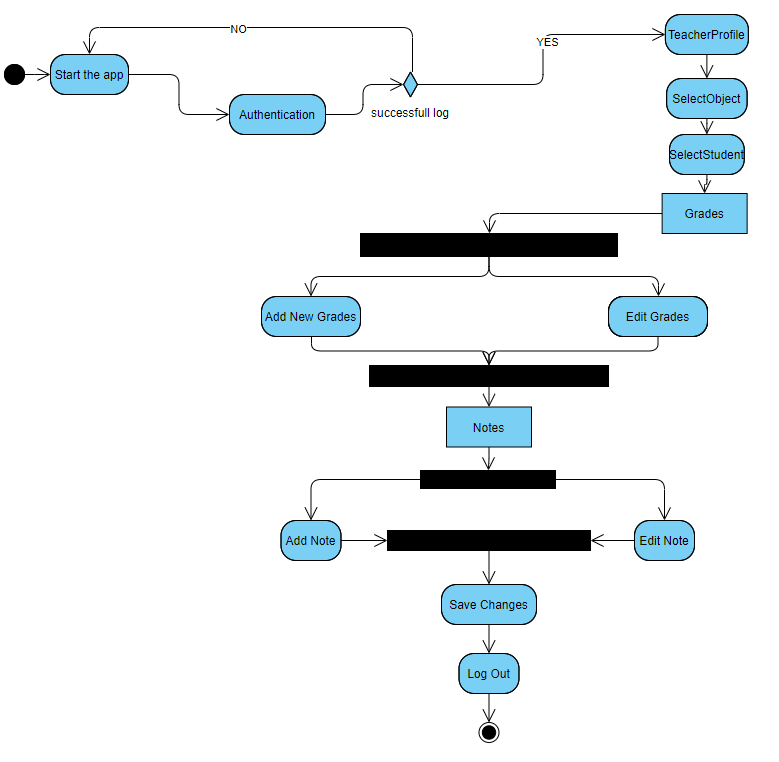


Следната Use - Case диаграма показва сценарий от гледна точка на учителя. Той влиза в приложението, като попълва потребителско име и парола. След автентикация, достъпва своя персонализиран преподавателски профил. Там има възможност да избере предмета, по който обучава. Позволено му е да добавя и редактира оценки. Може да влиза и в рецензиите по съответния предмет, където в задълженията му влиза да опише спрямо поставеното оценяване своите становища, препоръки и обобщения за ученика. Преподавателя изпраща данните към родителя, който следи актуалната информация за своето дете чрез собствен персонализиран профил.



Следната Use-Case диаграма показва сценарии от гледна точка на родителя. Той влиза в системата като въвежда на потребителско име и парола. След успешна автентикация родителят влиза в персонализирания си профил, който засяга само информацията за неговото дете. Избира предмета , който го интересува след , което има възможността да се запознае с оценките и персонализираните рецензии. Индивидуалните забележки са задължителен елемент, чиято цел е да позволява на всеки един родител да получава обективна критика относно състоянието на знанията на ученика. Така той винаги е запознат с прогреса и пропуските на детето си, което е важна част от контрола над учебния процес. Родителя може да генерира среден успех по избраната дисциплина. Чрез бутона „Видяно“ учителят ще знае кога потребителят-родител е проследил данните в системата, което е от изключително значение за адекватната обмяна на информация между образователната система и отговорния настойник.

Логическият модел на приложението се представя и чрез Activity UML диаграма. За по-голяма яснота дейността на потребителя е разделена на две основни части-гледна точка на преподавател и гледна точка на родител.



Първата част демонстрира последователността на процесите от гледна точка на обучаващото лице. Учителят стартира приложението, след което трябва да въведе потребителско име и парола .

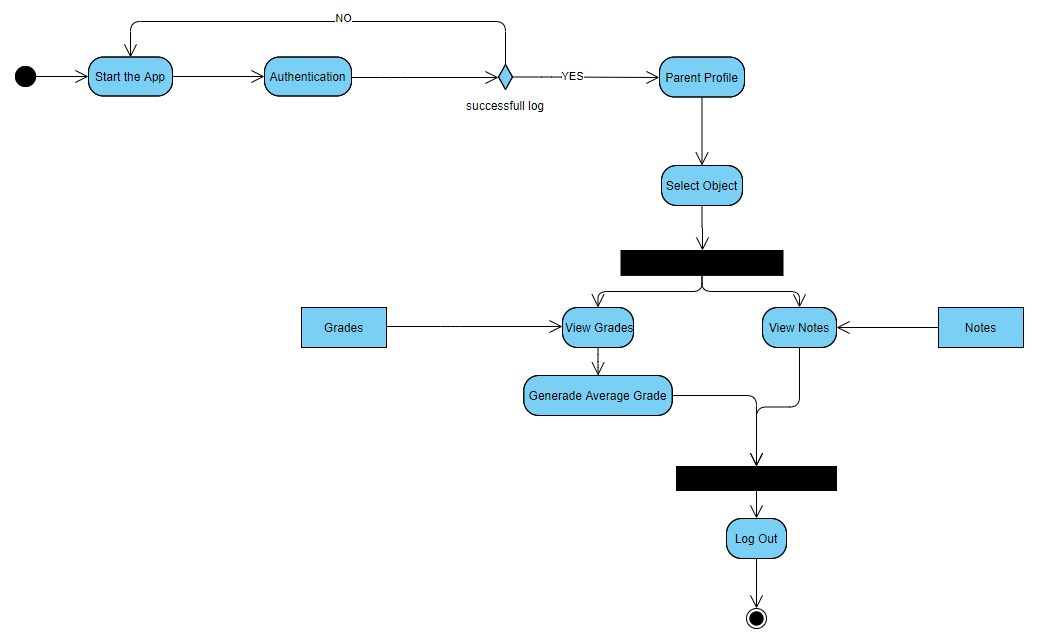
При успешна автентикация, системата отвежда потребителя в персонализирания му профил (в случая учителски). В противен случай не му позволява достъп до системата.

Преподавателят избира предмета, по който преподава, след което бива допуснат до списък с ученици. При избор на конкретно лице от дадения списък, се достъпват наличните негови оценки. Учителят има право да добави нови или да редактира стари такива.

След извършване на действията , преподавателят трябва да напише персонализирана рецензия, свързана с причините, поради които е поставил съответното оценка. По този начин родителя, за когото са предназначени тези данни, винаги е информиран относно прогреса и пропуските на своето дете.

Учителят може да добавя нови рецензии и да редактира стари при необходимост. След това текущите данните се запазват.

При извършване на гореописаните процеси учителят излиза от системата.



Втората Activity диаграма демонстрира последователността от действия, от гледна точка на родителя.

Родителят стартира приложението, като може да достъпи персонализирания си профил след успешна автентикация чрез правилно въведени потребителско име и парола.

След влизането в профила си родителското тяло може да избере учебния предмет, който се интересува. Родителят има достъп единствено до оценките и рецензиите на своето дете. По този начин той може обективно да проследява учебния процес на своя ученик и да разбира на време прогреса му.

Родителят вижда и персонализираните рецензии на учителя , които му дават възможност да е запознат с пропуските на ученика. Заедно с цялата подробна информация, в родителския профил се генерира и среден успех на детето по отделните предмети.

След получаване на необходимата информация родителят излиза от системата.

1. **Нефункционални изисквания**

Нефункционалните изисквания се използват за оценка на цялостната работа на електронния дневник. Приложението трябва да бъде лесно за използване и да притежава сигурна защита на данните.

В зависимост от реализацията, могат да се разгледат различни атрибути на качеството като:

* Достъпност: Както при всяко друго приложение, така и тук, възможностите за отказ могат да се появят при различни условия като неточности в кода, слаба интернет връзка, неспособност на системата да реагира адекватно в случаи на multitask, бъгове в устройството, изпълняващо приложението в конкретен момент и т.н. Поради ясната отчетливост на слоевете и свойството loose coupling на избраната Model-View-Presenter архитектура, тези обичайни ситуации биха били лесни за овладяване в максимално кратки срокове. Доброто разделение спомага за бързо откриване и поправяне на грешките и неточностите, причинили отказа.
* Сигурност: Съхраняването на важни лични данни, в комбинация със строгото разграничаване на двата вида роли (различни по дейност роля „Преподавател“ и роля „Родител“) в електронния дневник обуславят нуждата от оторизиран достъп и правят различна реакцията на системата в зависимост от това кой я използва. Тези условия са предпоставка за хеширане на паролите, мярка, минимизираща риска от изтичане на конфиденциална информация и повишаваща доверието в приложението. Слоевете на MVP архитектурата, както и многото код, нужен за поддръжката и осигуряват добър набор от тактики, които да предотвратят злоупотреби с данните.
* Възможност за тестване: Архитектурата Model-View-Presenter позволява лесно поетапно тестване на отделните компоненти поради слабото им свързване. Слабата връзка между слоевете осигурява независимото разгръщане, поддръжка и актуализация на различните компоненти на приложението, независимо от конкретния сценарий. Благодарение на дизайна на MVP архитектурата, преизползването на кода е напълно възможно, дори наложително в някои сценарии. Преизползването на код би улеснило тестването и би намалило цената за провеждане на тестовете.
* Използваемост: Тъй като за потребителите на електронния дневник може да се предполага, но не и да се очаква да имат добро ниво на компютърна грамотност, идеята на приложението е да бъде с максимално опростен интерфейс за комуникация между софтуера и ползвателя. Тези условия, в комбинация с добрата подреденост и ясното разграничаване на слоевете на Model-View-Presenter архитектурата, целят лесното взаимодействие на потребителя със софтуера на електронния дневник.
* Разширяемост: В същината си, разширяемостта е принцип, който осигурява бъдещо развитие и растеж. Предвидените от принципа подобрения, ненарушаващи съществуващите системни функции, могат да се осъществят безпроблемно, благодарение на “loose coupling” качеството на Model -View-Presenter архитектурата. Слабата връзка между слоевете прави лесна подмяната и иновацията в кода. Добрата капсулация способства за лесна интеграция на нововъведенията. Многото код, обаче, прави трудно овладяването на евентуално възникнали грешки при иницииране на ъпдейти/ъпгрейди в кода.

Използвани източници:

<https://fpmi.bg/moodle/mod/page/view.php?id=4593&forceview=1>

<https://syndicode.com/blog/12-software-architecture-quality-attributes/>

<https://docs.microsoft.com/en-us/archive/msdn-magazine/2010/september/msdn-magazine-cutting-edge-better-web-forms-with-the-mvp-pattern>

<https://docs.microsoft.com/en-us/archive/msdn-magazine/2011/december/mvpvm-design-pattern-the-model-view-presenter-viewmodel-design-pattern-for-wpf>

<https://www.quora.com/What-are-the-pros-and-cons-of-MVC-MVP-architecture-Do-I-need-to-use-these-architectures>

<https://rvtechnologies.com/mvp-vs-mvvm-architecture-patterns-for-android-app-development/>

<https://online.visual-paradigm.com/>

<https://proandroiddev.com/why-mvp-is-outdated-in-android-187083eeb4e6>