**СОФИЙСКА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ЕЛЕКТРОНИКА**

**„ДЖОН АТАНАСОВ“**

**Д И П Л О М Е Н П Р О Е К Т**

**Тема: “**Проектиране на уеб базирана платформа за участниците на капиталовите пазари“

**Практическа част:** „Реализация на уеб базирана платформа за участниците на капиталовите пазари“

Дипломант: Александър Андреев Иванов 12В клас

*/име, презиме, фамилия, клас/*

Професия: 481030 „ Приложен програмист“

Специалност: 4810301 „Приложно програмиране“

Ръководител на дипломен проект: инж. Любица Димитрова

Дипломант: .....................

*/подпис/*

Ръководител на Дипломен проект: .................

*/подпис/*

София

2023

**СЪДЪРЖАНИЕ**

Съдържание............................................................................................................2

**Теоретична част**...................................................................................................4

Глава I. Увод...........................................................................................................4

Глава II. Изложение................................................................................................6

1. Съществуващи решения.........................................................................6
2. Предпроектно проучване.......................................................................7
3. Архитектури, концепции, парадигми и дизайн шаблони................9
   1. „Клиент – сървър“ мрежова архитектура............................8
   2. Трислойна архитектура........................................................10
   3. Уеб API..................................................................................10
   4. Едностранични приложения...............................................10
   5. Обектно ориентирано програмиране..................................10
   6. SOLID принципи..................................................................10
   7. Общи принципи при проектиране......................................10
   8. Обектно релационно картографиране................................10
   9. Дизайн шаблони...................................................................10
   10. Асинхронно програмиране..................................................10
4. Платформи, езици, библиотеки и протоколи.....................................10
   1. C#, .NET и ASP.NET CORE.................................................10
   2. Релационни бази данни........................................................10
   3. Entity Framework Core..........................................................10
   4. HTTP и WebSocket протоколи............................................10
   5. SignalR и уеб функционалности в реално време...............17
   6. Допълнителни библиотеки на С#.......................................17
   7. HTML, CSS и JS...................................................................18
   8. React - JavaScript библиотека..............................................18
5. Структура на имплементацията.........................................................18
   1. Структура на базата данни..................................................18
   2. Структура на сървърното приложение..............................18
   3. Структура на клиентското приложение.............................18

Глава III. Заключение..........................................................................................28

**Практическа част**...............................................................................................29

Тема, реализация, изводи.....................................................................................30

Използвана литература.........................................................................................31

Приложения...........................................................................................................32

**I. Увод**

Ние живеем в свят, който ни предоставя информация, на където и да насочим погледа и слуха си. Ежедневието ни е изпълнено с рутинни действия, но и важни решения, които трябва да направим. За съжаление обаче нашето време за работа и за възприемане на нова информация е ограничено. Днес човек израства в кариерен план и увеличава вероятността си за повишение само ако е способен да бъде високопродуктивен. Продуктивността се измерва чрез намирането на отношението между броя и нивото на трудност на успешно завършените задачи спрямо времето, за което те са изпълнени. Затова човек трябва да намери начин да автоматизира повтарящите се задачи в ежедневието си, да филтрира излишната информация и да организира работната си среда възможно най-оптимално.

Преди да дефинирам съществуващите всекидневни технически проблеми във финансовия свят, аз бих искал първо да дам отговор на следните фундаментални въпроси във финансите: какво са капиталовите пазари, защо са важни, кои са участниците в тях и какво сa финансови инструменти. Капиталовите пазари са интернационална система от борси, която позволява свободното движение на капитали в целия свят. Тази система улеснява компаниите и правителствата в процеса им на привличане на финансиране от обществото. Участниците на капиталовите пазари могат да бъдат инвестиционни банки, брокери, пенсионни фондове, взаимни фондове, застрахователни компании и хедж фондове. Част от пазара могат да бъдат и физически лица, които обаче трябва да използват брокер като посредник, за да получат достъп до борса, а съответно и до пазара. Участниците на капиталовите пазари се наричат търговци или борсови спекуланти. Финансовите инструменти са индивидуални активи, които се търгуват на капиталовите пазари, тоест те могат да се купуват и продават на борсата от участниците в нея. Всички видове участници са потенциални потребители на платформата, която аз ще проектирам и реализирам.

След като отговорих на тези основополагащи въпроси от сферата на финансите, е време да се запознаем с техническите проблеми в нея, които трябва да бъдат решени. Участниците на капиталовите пазари днес срещат проблеми с огромния обем от информация и ограничено време за работа, състояща се от създаване на анализи, наблюдаване на търговския процес на капиталовите пазари и изпълнение на търговски поръчки. Те също споделят желанието за висока продуктивност с хората от останалите професии. Затова основните цели на софтуера, който ще разработя, са автоматизиране на повтарящите се задачи в ежедневието на участниците на капиталовите пазари и филтриране излишната информация за тях. Тези цели ще бъдат постигнати, ако платформата, включва следните функционалности: лесно достъпване на източник на достоверни новини; осигуряване на затворена и конфиденциална социална мрежа от спекуланти, работещи заедно, чрез която да обменят ефикасно знания, прозрения и мнения; водене на счетоводна книга за търговските поръчки и на записки, свързани с психическото състояние на спекуланта по време на търговия. Софтуерът също трябва да осигурява достъп до котировки на финансови инструменти, икономически данни, календар с важните актуални събития и калкулатор за изчисляване на възвръщаемостта на търговска позиция. Чрез тези функционалности аз ще достигна основната си цел – да създам платформа, която спестява ценно време, предоставя удобни инструменти за работа и качествена информация на участниците на капиталовите пазари, за да могат те да вземат ежедневно правилни решения. Софтуерът също ще удовлетворява желанието на търговците да достъпват всичките си инструменти за работа и източници на информация през една платформа.

Задачите за създаване на гореописаните функционалности са следните: да се проучи начин за събирането на данни като новини, календарни събития, икономически показатели котировки на финансови инструменти, за да се създаде и тяхна визуализация. Във връзка с воденето на счетоводство трябва да се анализира кои елементи от търговския процес са най-значими и да се имплементират. За воденето на дневник за емоционалното състояние по време на работа на борсовия спекулант, трябва да се създаде личен електронен дневник за записки. По отношение на комуникацията между тесен кръг от заедно работещи спекуланти, ще създам чат стаи, в които те ще обменят писмено информация в реално време. За разработването на калкулатора за изчисляване на възвръщаемостта на търговски позиции, ще трябва да се имплементират стандартизирани математически формули. Всеки потребител трябва да има акаунт в платформата, за да я използва пълноценно. Това означава, че трябва да се разработят функционалности за управление на потребители като регистрация, автентикация и опции за редакция на потребителски профил.

**II. Изложение**

1. **Съществуващи решения**

В днешно време съществуват платформи за новини като Newsquawk и Financial Times, калкулатори за изчисляване на възвръщаемостта на търговска сделка като тези на StoneX Financial, уебсайтове като Trading Economics, които да визуализират икономически и финансови показатели, и социални мрежи като Microsoft Teams и Facebook, чрез които хората могат да комуникират. Много източници на образователно съдържание за това как се спекулира и инвестира на капиталовите пазари могат да бъдат открити в интернет. Създадени са вече и софтуерни разработки като Ексел на Майкрософт, които се използват за счетоводна дейност. Но всички тези приложения се поддържат от различни компании с различни цели, което води до невъзможността те да работят заедно върху желанията на общ за всички тях тесен кръг от потребители. Най-често предназначението на тези платформи не е фокусирано върху улесняването на работната дейност на търговците. Тези две причини водят до реалността, в която борсовите спекуланти трябва да се съобразяват и адаптират със софтуерните приложения, които съществуват, без да имат възможността да поискат промени по функционалностите или визуализацията. Това поражда в мен идеята да създам платформа, която обединява всички необходими на търговците инструменти за работа и източници на информация. Тази платформа ще бъде предназначена само за анализиране на капиталовите пазари, за информиране на техните участници и за следене на производителността и психологическото състояние на търговците по време на работа.

Благодарение на това, че платформата, която ще създам, е тясно ориентирана, потребителското мнение под формата на предложения и оценки за функционалностите на софтуера ще бъдат разглеждани и обсъждани с финансисти и софтуерни инженери. На база финансовото и инженерното експертно мнение ще се правят решения дали потребителските желания, изразени като мнения, да се имплементират в платформата или не.

Този процес е невъзможен при повечето съществуващи платформи, които търговците използват днес, тъй като голяма част тях не са предназначени само за работата на борсовите спекуланти.

1. **Предпроектно проучване**

В този раздел ще представя накратко предпроектното проучване и изводите от него, които направих преди да започна създаването на софтуерния дизайн и имплементацията на софтуерния продукт – моята дипломна работа.

Преди началото на изграждането на софтуерен дизайн и реализацията на платформата за участниците на капиталовите пазари, трябва да направя анализ на функционалните изисквания към платформата. Те са декларирани в уводната част на този научен труд на страница 4, където описвам какви функционалности са необходими, за да се постигне целта на платформата. Анализът върху функционалните изисквания е необходим, за ​​да се провери осъществимостта на разработването на дадения софтуерен продукт. След като се определи дали проектът е технически и финансово осъществим, се създава документ за спецификация на софтуерните изисквания (Software Requirement Specification - SRS). В него се описват функционални и нефункционални изисквания към софтуера, както и неговия обхват. Документът е необходим, за да се вземат правилни решения при създаването на софтуерния дизайн и за да бъдат дефинирани задачи, които да се изпълнят по време на имплементацията на платформата. Дипломният ми проект може да бъде частично разглеждан като документ за спецификация на софтуерните изисквания (Software Requirement Specification – SRS).

Най-важната стъпка от цялото предпроектно проучване е да се подберат качествените източници на икономически и финансови данни, новини, актуални събития, параметри на финансови инструменти и котировки на търгувани активи. Без качествени данни, платформата е лишена от причина да съществува, тъй като никой участник на капиталовите пазари няма да изложи средствата си на риск, ако знае, че той работи с некачествена информация. Именно затова въпросът за данните е първи, защото отговорът ще определи дали реализацията на софтуерния продукт е осъществима.

Алтернативи за източници за новини има много: Yahoo Finance, Investing.com, Newsquawk и Benzinga PRO. Това са едно от най-легитимните официални платформи за новини. От изброените доставчици на новини Newsquawk и Benzinga PRO са най-бързи и най-качествени. Ползването на техните услуги обаче изисква средства, които надхвърлят бюджета на проекта. Поради ограничените финансови условия, изборът на доставчик на данни е между Yahoo Finance, Investing.com. Според моите наблюдения Imvesting.com са по-бързия доставчик на новини и затова ще използвам тях при реализацията на платформата.

Източникът на икономически и финансови данни ще бъде базата данни на системата от централни банки на Съединените американски щати. Името на базата данни е „Federal Reserve Economic Data(FRED)“. Изборът ми се базира на легитимността на тази институция.

Източникът на декларирани отворени позиции на пазарите на деривати, които са необходими за направата на качествени анализи от професионалните търговци, ще бъде Комисията по търговски деривати на суровини - Commodity Futures Trading Commission(CFTC). Изборът ми също се базира на легитимността на тази институция.

Източникът на параметри на деривативните финансови инструменти ще бъде Чикагската стокова борса - Chicago Mercantile Exchange(CME)“. Изборът ми се базира на легитимността на тази институция.

Източникът на актуални събития и на котировките на финансови инструменти е платформата TradingView, защото предлага качествена и рентабилна информация.

Тъй като проектът изисква използването на няколко, вече съществуващи и достъпни, източници на информация, то събирането на икономически и финансови данни, новини, актуални събития, параметри по финансови инструменти и текущите котировки на водещи активи е възможно.

**Таблица 1. Използвани външни източници на данни**

|  |  |
| --- | --- |
| **Източник** | **Какви данни предоставя?** |
| TradingView | Актуални котировки на множество от активи; Календарни събития |
| Federal Reserve Economic Data | Икономически и финансови показатели |
| Commodity Futures Trading Commission | Доклади за декларираните отворени позиции на деривативните пазари |
| Investing.com | Актуални новини |
| Chicago Mercantile Exchange | Спецификации по деривативните договори; математически формули за изчисляване на възвръщаемостта на търговска позиция |

След като източниците на качествени данни са подбрани, то следва да се проучи дали технически е възможно да се разработи уеб платформа, която, освен да събира и да визуализира тези данни от външни източници, може да предлага услуги като частни чат стаи, калкулатори, персонални акаунти и дигитализирани опростени счетоводни книги. Чрез създаването на база от данни и използването на обектно-ориентиран език, скриптов език, уеб протоколи за комуникация, допълнителни библиотеки и технологична рамка за разработка на уеб приложения аз мога да създам софтуерен продукт, улесняващ ежедневната дейност на професионалните участници на капиталовите пазари.

Тъй като успешно съм подбрал достъпни източници на качествена информация и съм способен да имплементирам функционалности по зададените в уводната част функционални и нефункционални изисквания, то мога да определя софтуерния продукт – уеб платформа, улесняваща работата на участниците на капиталовите пазари – за технически и финансово осъществима.

1. **Архитектури, концепции, парадигми и дизайн шаблони**

Жизненият цикъл за софтуерната разработка гласи, че след предпроектното проучване трябва да се направи дизайн на софтуерния продукт. Той включва решения за това каква мрежова архитектура да се приложи, какви парадигми и дизайн шаблони да се използват и какви принципи да се следват. На този етап също се създават архитектура на релационната база данни и архитектура както на сървърното, така и на клиентското уеб приложение.

* 1. **„Клиент – сървър“ мрежова архитектура**

**М**режова архитектура, върху която ще се базира платформата, е „клиент-сървър“. Тя дефинира две основни роли: клиент и сървър. Тяхната комуникация се изпълнява посредством заявки и отговори. Тази архитектура е отлична за целта на платформата, защото сървърът ще бъде ангажиран с изпълнение на конкретна услуга само след получаване на заявка от клиент, а не постоянно. Редът на обмен на информация между „клиент-сървър“ е:

1. Клиентът подават заявка за определена услуга към съответния порт на сървъра
2. Сървърът изпълнява конкретната заявена услуга
3. Сървърът връща евентуален отговор (възможно е да не изпрати такъв в следствие на прекъсване на връзката)
4. Клиентът получава отговора

Този модел на работа позволява използването на различни протоколи за комуникация като HTTP и WebSocket.

* 1. **Трислойна архитектура**

В софтуерното инженерство многослойната архитектура (позната още като N-слойна архитектура) е архитектура от типа клиент-сървър, в която потребителският интерфейс, бизнес логиката на приложението и съхранението на данните са логически разделени на модули. Най-разпространената форма на многослойна архитектура е трислойната архитектура. На нея ще се базира платформата, която ще разработвам.

Причината да използвам многослойната архитектура е в това, че тя позволява създаване на гъвкави приложения. При разделянето на едно приложение на слоеве, аз ще имам възможността да добавям или променям отделен слой, без да имам необходимостта да преработвам цялото приложение. Начинът на работа на отделните модули е независим.

Трислойната архитектура е изградена от презентационен слой (наричан още потребителски интерфейс), който служи за прякото взаимодействие с потребителя и изпращането на заявки към бизнес слоя. Не трябва да има директна връзка между презентационния слой и слоят за бази данни. Бизнес слоят контролира функционалността на приложението като извършва различни процеси по обработката на данните. Той комуникира и с презентационния слой, и с базите данни. Слоят за данни е третият слой. Той служи за съхранение данни и комуникира само с бизнес логиката.

Diagram

Description automatically generated

**Фиг. 1 Примерна постъпкова диаграма на трислойната архитектура**

Платформата, която ще разработвам, ще спазва принципите на Традиционния трислоен модел. Използвайки този подход, потребителите правят заявки през графичен интерфейс(презентационен слой), който взаимодейства само със слоя с бизнес логиката. Слоят с бизнес логиката може да достъпва слоя за данни. Един недостатък на този традиционен подход е, че презентационният слой зависи от слоя с бизнес логиката, който пък зависи от слоя за данни. Това означава, че слоя с бизнес логиката, който обикновено съдържа най-важната логика в приложението, зависи от подробностите за изпълнение на достъпа до данни.

Diagram, timeline

Description automatically generated

**Фиг. 2 Традиционен трислоен модел**

Алтернатива на традиционния трислоен модел е Изчистената архитектура (Clean architecture). Изчистената архитектура поставя бизнес логиката в центъра на приложението. Вместо бизнес логиката да зависи от достъпа до данни или други инфраструктурни проблеми, тази зависимост е обърната: детайлите на инфраструктурата и изпълнението зависят от ядрото на приложението (бизнес логиката). Тази функционалност се постига чрез дефиниране на абстракции или интерфейси в ядрото на приложението, които след това се имплементират от типове, дефинирани в инфраструктурния слой. Казано синтезирано, слоя с бизнес логиката е толкова независима, че при драстични промени като промяна на база данни или цялостна промяна на технологичната рамка на приложението тя ще продължава да функционира правилно.

Diagram

Description automatically generated

**Фиг. 3** **Изчистената архитектура (Clean architecture)**

Причините да имплементирам платформата по Традиционния трислоен модел са две на брой:

1. Чистата архитектура изисква много предварително програмиране, преди действително да се приложи софтуерно решение. Липсата на достатъчно време за имплементация подтиква към по-бързата за реализация архитектура.
2. Не предполагам промяна на системата за управление на базата данни (възможността за промяна на системата за управление на базата данни е един от плюсовете на Изчистената архитектура)
   1. **Уеб API**

Преди да започна с разяснението какво е уеб API и защо е необходим, бих искал да споделя как конкретно приложението ми ще бъде разделено спрямо принципите на трислойната архитектура. Платформата, която ще реализирам, ще бъде разделена на две отделни приложения. Едното ще сървърно, другото – клиентско. Разликата между тях е, че сървърното приложение ще приема заявки без значение дали източникът е уеб клиент, мобилно приложение или дори друг уеб сървър. Сървърното приложение поема отговорността на слоя с бизнес логиката и отговорността на слоя за данни. Клиентското приложение ще бъде зареждано и изпълнявано в рамките на уеб клиента на потребителя (ще се стартира на браузъра). Сървърното приложение приема ролята на уеб API. Но какво е уеб API?

Първо ще представя общото определение за API: Това е интерфейс за програмиране на приложения (application programming interface - API), който представлява набор от дефиниции на подпрограми, протоколи и инструменти за изграждане на софтуерни приложения. Казано с прости думи, API е интерфейс, който има множество от функции, които позволяват на програмистите да имат достъп до специфични функционалности или данни на приложение, операционна система или други услуги. Уеб API, както подсказва името, е API в мрежата, който може да бъде достъпен чрез HTTP протокол. Важно е да се отбележи, че това е концепция, а не технология. Можем да изградим Web API, използвайки различни технологии като Java, .NET и др.

Причината да разделя платформата на две отделни приложения и съответно да имплементирам сървърното приложение като уеб API е, че целя независимост на сървърното приложение (сървъра от „клиент-сървър“ мрежовата архитектура) от типа клиент (браузър, мобилно приложение или друг уеб сървър). Това е необходим подход на имплементация, защото в бъдеще потребителите могат да поискат не само имплементация на уебсайт, но и на мобилно приложение или ще заявят желание просто да достъпват данните, които се доставят от сървърното приложение. Тази директна комуникация със сървърното приложение за достъп до данни и функционалности, реализираща се без графичен интерфейс, е позната като „Машина към машина“ („machine-to-machine“). Тази комуникация е силно разпространена в днешно време и това дава стимул на решението ми да имплементирам уеб API.

Незаивсимостта на сървърното приложение ще бъде постигната, като го имплементирам да връща ресурси във формат JSON, а не във формат HTML или XAML. Сървърното приложение не трябва да отговаря за графичния интерфейс. Това е отговорност вече на клиентското приложение. Сървърното приложение трябва да доставя нужните данни и услуги за създаване, актуализиране и изтриване на данни и не трябва да отговаря за презентационната логика. Това е корена на причината да разделя платформата на сървърно и клиентско приложение. Този подход ще ми позволи и да създам модерни едностранични приложения с добро потребитлско преживяване, но този аспект ще го разясня подробно в раздел „Едностранични приложения“.

Diagram

Description automatically generated  
**Фиг. 4 Уеб API, „Клиент-сървър“ мрежова архитектура и трислойна архитектура, обединени в една диаграма**

* 1. **Едностранични приложения**

Едностраничното приложение (Single-page application - SPA) е уеб приложение, което работи изцяло в уеб браузъра и зарежда само един документ. То не изисква опресняване на страницата от страна на браузъра по време на употреба, а по-голямата част от DOM(Document Object Model) дървото остава непроменено, тъй като само малка част от него трябва да бъде актуализирана при определено събитие. Когато съдържанието на страницата трябва да бъде променено, едностраничното приложение използва API на JavaScript. Потребителите могат да получат достъп до уебсайта, без да се налага постоянно да теглят цяла нова страница при промяна на URL адреса. В резултат на това производителността и потребителското преживяване се подобряват.

Едностраничните приложения работят по следния начин: Когато въведем URL адрес на уеб страница в браузъра, за да поискаме достъп, браузърът прави заявката до определения в URL адреса сървър, който отговаря с HTML документ и други статични файлове като Cascading Style Sheets (CSS) и JavaScript. Сървърът доставя HTML съдържание и други статични файлове само с първата заявка. Всяка следваща заявка до сървъра обаче връща данни в JSON(или XML) формат, но не и HTML или други статични файлове, свързани с презентационната логика на уебсайта. Тези последващи заявки се приемат от уеб API, който представих в раздел „Уеб API”. Отговорите на уеб API пораждат промени по страницата на едностраничното приложение. Тези промени не изискват опресняване на страницата от уеб браузъра.

Тяхната алтернатива – многостаничните приложения – получават от уеб сървъра HTML файл, който те трябва да рендерират в браузъра. Този нов HTML файл налага опресняване на страницата от уеб браузъра.

Клиентското приложение за уеб браузъри на платформата за участниците на капиталовите пазари ще бъде едностранично. Поради това ми решение, ще представя ползите и недостатъците на едностраничните приложения (SPA).

**Таблица 2. Ползите и недостатъците на едностраничните приложения**

|  |  |
| --- | --- |
| **Ползи** | **Недостатъци** |
| По-добро потребителско преживяване | Потенциално по-дълго време за първоначално зареждане |
| Подобрена скорост | Неефективно SEO  (Индексация, уникални връзки и метаданни липсват) |
| Използване на по-малко интернет ресурс |  |
| Ефективно кеширане |  |

Недостатъка „Потенциално по-дълго време за първоначално зареждане“ е пренебрежимо малък при правилното използване на технологичните рамки за създаване на едностранични приложения. А недостатъка „Неефективно SEO (Индексация, уникални връзки и метаданни липсват)“ е решим с SEO техники за оптимизиране.

Поради съществуването на ползи и на решения на недостатъците на едностраничните приложения аз смятам, че е добро решение да създам клиенското приложение, базиращо се на SPA, за платформата.

* 1. **Обектно ориентирано програмиране**

Време е да изляза от темата за архитектурите и абстрактните концепции и да навляза в темата за програмните парадигми и принципи. В този раздел ще представя накратко трите основни по популярност парадигми в програмирането, а след това ще се обоснова защо съм избрал обектно ориентираното програмиране за моя проект.

Процедурното програмиране директно инструктира хардуерното устройство какви логически стъпки трябва да се извършат. Тази парадигма използва линеен подход, като третира данните и процедурите като две различни свои структурни единици. Процедурното програмиране разделя програмата на процедури (подпрограми или функции), съдържащи поредица от стъпки, които трябва да бъдат извършени. Казано синтезирано, процедурното програмиране е създаване на списък с инструкции, които се изпълняват стъпка по стъпка, за да се свърши дадена логическа задачата.

Обектно-ориентираното програмиране (ООП) е наследник на процедурното програмиране. То разглежда компютърната програма като реалния живот, изпълнен с разнообразни обекти, които имат различни характеристики. Обектно-ориентираното програмиране се базира на множество от обекти и тяхното взаимодействие помежду им. Всеки обект е самоустойчив, защото съхранява в себе си всички нужни променливи, които го характериират и методи, представляващи действия, които обекта може да извършва. Най-важните принципи в ООП са наследяването, капсулирането, полиморфизмът и абстракцията. Накратко казано, програмата се разделя на обекти, които взаимодействат помежду си.

Функционалното програмиране се различава от процедурното програмиране и обектно-ориентираното програмиране, тъй като използва математически функции. Чрез него резултатите от операциите се базират само на въведените входни данни на математическата функция(разглеждата като чиста функция). Операциите(чистите функции) не разчитат на външни, временни или скрити, променливи (познати като състояние на програмата).

Diagram

Description automatically generated

**Фиг. 5 Типове програмни парадигми**

Избрах обектно ориентираното програмиране за моя проект, защото:

1. Капсулирането предоставя модулиране на приложението, което води до по-лесно менажиране и дебъгване на кода
2. Бъдещи добавяния към проекта не оказват влияние върху вече съществуващия код (стига да се спазват SOLID принципите, които ще представя в раздел „SOLID принципи“)
3. Повтаряемостта на кода се намалява чрез преизползване и наследяване на обекти
4. Разделянето на отговорности (като основно правило в програмирането) между обектите е по-лесно осъществимо
5. Имплементацията на решения, които ще се използват в реалния свят, се реализира по-лесно чрез ООП, защото то се базира на реалния живот

Чрез обекти аз мога да представя различните финасови активи, акаунти на потребители, новини, актуални събития, книги (или друг вид обучително съдържание за секция образование в пларформата), съобщения по чат стаите, самите чат стаи и записите в счетоводните книги. Погледнато на по-високо ниво, всички клиентски заявки, услуги, които предоставя уеб API-а на платформата, и техните резултати могат да бъдат разглеждани като обекти.   
 Това са причините да използвам обектно ориентираното програмиране за моя проект.

* 1. **SOLID принципи**

Чисто философски погледнато, принципите същестуват, за да ни помагат, а не просто за да създават неудобство.

1. **Платформи, езици, библиотеки и протоколи**

**---------**Реферирай раздел „Архитектури, парадигми и дизайн шаблони“ **---------**

Изборът какви инструменти за разработка да използвам, се базира на отговорите на следните въпроси: „Език от високо или от ниско ниво ще бъде необходим?“, „Би ли било полезно този език да бъде обектно-ориентиран?“, „Ще бъде ли необходима база данни и какъв тип трябва да бъде тя?“, „Ако приложението е уеб базирано, какви протоколи за комуникация трябва да се спазват?“ и „Какви допълнителни библиотеки и технологички рамки(„Фреймуърс“, от англ. “Frameworks”) ще бъдат необходими?“.

* 1. **C#, .NET и ASP.NET CORE**

Платформата, която ще разработвам, ще бъде уеб базирана, достъпвана през уеб клиент – браузър, ще се базира на парадигмата в компютърното програмиране „Обектно ориентираното програмиране“ и няма проблем да бъде реализирана на език от високо ниво.

Езикът С# е избраният от мен език, чрез който да имплементирам платформа за участниците на капиталовите пазари. Той е обектно ориентиран, строго типизиран