**ДИПЛОМЕН ПРОЕКТ**

Тема:

РАЗРАБОТКА НА MCV УЕБ ПРИЛОЖЕНИЕ ЗА ВЪПРОСИ И ОТГОВОРИ ТИП "МИНИ ФОРУМ "

Изготвил:

**Георги Петров Георгиев-12 А клас**

професия код 481030 "Приложен програмист"

специалност код 4810301 "Приложно програмиране"

Ръководител: **инж.** **Наталия Пенева**

[1 Увод 3](#_Toc153357914)

[2 Основни цели и задачи на дипломния проект. 3](#_Toc153357915)

[3 Теория 5](#_Toc153357916)

[3.1 **Мрежови протоколи** 5](#_Toc153357917)

[3.2 **Клиент-сървър комуникация** 10](#_Toc153357918)

[3.3 **HTTP requests** 11](#_Toc153357919)

[3.4 **HTTP Response** 13](#_Toc153357920)

[3.5 **Сървърни езици** 15](#_Toc153357921)

[3.6 **Бази данни и СУРБД** 17](#_Toc153357922)

[3.7 **UML и E/R диаграми.** 24](#_Toc153357923)

[3.8 **Принципи на правилното разделение на приложението на слоеве** 27](#_Toc153357924)

[3.9 **Многослойна архитектура** 28](#_Toc153357925)

[3.10 **Трислойни модели** 29](#_Toc153357926)

[3.11 **Обектно-релационно свързване (ORM frameworks)** 32](#_Toc153357927)

[3.12 **Упълномощаване (authentication) и удостоверяване (authorization).** 33](#_Toc153357928)

[3.13 **HTML и основни тагове** 36](#_Toc153357929)

[3.14 **Семантични страници** 37](#_Toc153357930)

[3.15 **CSS, селектори и основни правила в CSS** 39](#_Toc153357931)

[3.16 **Адаптивно (responsive) оформление на страници** 40](#_Toc153357932)

[4 Използвани технологии 41](#_Toc153357933)

[5 Структура на приложението 43](#_Toc153357934)

[5.1 **База данни ER диаграма** 43](#_Toc153357935)

[5.2 **Описание на таблиците в базата** 44](#_Toc153357936)

[5.3 **Структура на кода** 48](#_Toc153357937)

[5.4 **Клас диаграми** 57](#_Toc153357938)

[5.5 **Функционалност** 63](#_Toc153357939)

[6 **Заключение** 65](#_Toc153357940)

[7 **Ресурси и използвана литература** 66](#_Toc153357941)

[8 **Приложения** 13](#_Toc153357942)

[8.1 **Source code на проекта** 13](#_Toc153357943)

[8.2 **Теоретична разработка – pdf и word файл** 13](#_Toc153357944)

* 1. [**Екрани (Screenshots)** 13](#_Toc153357945)

# 

# 1.Увод

**Въведение в QuestSolve:**

**Съвременен подход към училищното обучение чрез ангажиране на общността**

Навигиране в новата ера на дигиталното обучение

QuestSolve е в челните редици на образователните иновации, предоставяйки уникална онлайн платформа, пригодена специално за училищните общности. Покриваща нарастващата нужда от цифрови ресурси в образованието, QuestSolve предлага специално място за учениците да се информират, да питат и да си помагат, подпомагайки традиционния процес на домашна работа и обучение**.**

**Форум за съвместно образование**

QuestSolve предефинира начина, по който учениците изпълняват училищната си задължения. Това е нещо повече от платформа за въпроси и отговори; Това е жизнена общност, в която учениците могат да търсят помощ, да споделят информация и да се справят с академичните предизвикателства заедно.

# 2.Основни цели и задачи на дипломния проект.

Основни характеристики на QuestSolve

• Специфични за училището форуми: QuestSolve е уеб апликация специално предназначена за училищните общности, осигурява безопасна и структурирана среда, в която учениците могат да задават въпроси, да обсъждат теми извън класната стая и да получават помощ при домашна работа или важни за тях образователни потребности.

• Контролирано от администратор: Осигуряване на безопасна и уважителна учебна среда, всички публикации се одобряват от администраторски потребители. Това ниво на сигурност поддържа качеството и уместността на съдържанието.

• Съвместно учене: В основата си QuestSolve насърчава култура на подкрепа между връстници. Учениците могат да си помагат, да споделят знания и да предлагат различни гледни точки, обогатявайки учебния опит.

• Лесен достъп до информация: Възползвайки се от лесния достъпа до интернет, QuestSolve използва това, за да предостави на учениците лесен и бърз достъп до среда за обмен на образователни ресурси.

• Насърчаване на любопитството и решаването на проблеми: QuestSolve е предназначен да стимулира любопитството и критичното мислене, насърчавайки учениците да изследват учебните теми и да участват в дискусии.

Визия и цел на QuestSolve

QuestSolve не е само платформа - това е движение към интерактивен, ориентиран към общността подход към образованието. В днешната онлайн ера, QuestSolve се фокусира върху значението на това да направим ученето достъпно, ангажиращо и съвместно. Платформата дава възможност на учениците да поемат отговорност за своето обучение, да търсят помощ, когато е необходимо.

Свързване на дигиталния и образователния свят

Чрез комбиниране на силата на онлайн форумите със специфичните нужди на училищните общности, QuestSolve преодолява пропастта между цифровите ресурси и ефективното обучение. Той предлага нов начин за учениците да се свързват със своите връстници, да се ангажират с академично съдържание и да навигират уверено в образователните си пътувания.

Сравнителен анализ с конкуриращи платформи:

QuestSolve се изправя в конкуренция с редица платформи, включително такива като Reddit, който е известен със своята широка общност и форуми за обсъждане. Въпреки това, има няколко ключови аспекти, които отличават QuestSolve и го правят уникално подходящ за образователните нужди на училищата:

**1. Фокус върху образователния сектор:**

o Докато платформи като Reddit предлагат широк спектър от теми, QuestSolve е специализирана и ориентирана изцяло към образованието. Това означава, че всичко съдържание, дискусии и ресурси в QuestSolve са създадени с цел подобряване на учебния процес и образователното развитие на учениците.

**2. Административен контрол и модерация:**

o Важна характеристика на QuestSolve е изискването за административно одобрение на всеки нов потребител. Това гарантира, че всички участници в платформата са верифицирани и отговарят на определени образователни стандарти. Това намалява риска от нежелано съдържание и осигурява безопасна и фокусирана образователна среда, което е особено важно в училищния контекст.

**3. Ориентираност към училищните нужди:**

o QuestSolve е разработена с конкретна цел - да подпомага учебния процес в училищата. Това включва създаване на образователни форуми, специфични за училището, и предлагане на функции, които са директно свързани с нуждите на учениците, учителите и администрацията.

**4. Специализирани образователни ресурси:**

o В отличие от общи дискусионни платформи, QuestSolve предлага специализирани образователни ресурси и инструменти, които са интегрирани в платформата. Това включва достъп до учебни материали, инструменти за сътрудничество и функции за проследяване на учебния напредък.

**5. Безопасност и сигурност:**

o QuestSolve поставя висок приоритет на безопасността и сигурността на своите потребители, особено като се има предвид чувствителността на училищната среда. Това включва строги протоколи за поверителност и сигурност, които са по-засилени в сравнение с общите социални платформи, поради контролираната от администратор среда.

QuestSolve ефективно разрешава някои от основните предизвикателства и пропуски, които не се адресират от други платформи. Тя създава специализирана, сигурна и контролируема образователна среда, която е особено подходяща за училищния контекст. Това я прави изключително ценна за образователни институции, които търсят надеждни и ефективни цифрови решения за подобряване на учебния процес.

# 3.Теория

## **Мрежови протоколи**

Мрежовият протокол е съвкупност от правила, чрез които компютрите и останалите

устройства в мрежата комуникират помежду си.

Как работят протоколите?

При изпращащия компютър протоколът прави следното:

• Разделя данните на малки части, наречени пакети;

• Добавя адресна информация към всеки пакет;

• Подготвя данните за реално предаване през мрежовия кабел.

При получаващия компютър протоколът извършва същите серии от стъпки в обратен

ред:

• Получава пакетите данни от кабела в мрежовата адаптерна карта;

• Отделя от пакетите служебната информация;

• Копира данните от пакетите в буфер, за да ги сглоби в първоначалния им вид;

• Подава сглобените данни в използваем вид към съответното приложение.

Съществуват:

• немаршрутизируеми протоколи – за предаване на данни в една локална мрежа;

• маршрутизируеми протоколи – използват се за предаване на данните от една локална

мрежа към друга по един от няколко възможни пътя (маршрута).

Протоколите се разделят на следните три групи:

• Приложни – осигуряват обмена на данни и взаимодействието приложениеприложение (SMTP, FTP, Telnet, HTTP и др.);

• Транспортни – отговарят за надеждността при придвижване на данните в мрежата

**(TCP, NetBEUI);**

• Мрежови – осигуряват така наречените свързващи услуги. Управляват адресиращата

и маршрутизираща информация, осъществяват проверките за грешки и заявките за

повторно предаване (IP).

### **IP**

Протокол за комуникация, чрез който се позволява адресиране на информацията, която се изпраща по мрежата. На всяко устройство в мрежата се дава уникален адрес (наречен IP адрес). Когато се изпраща информация през мрежата, тя се разделя на малки пакети, наречени IP пакети. Към всеки пакет се прикрепя т. нар. хедър (header), който съдържа IP адресите на подателя и получателя и други служебни данни. Така устройствата, през които минават пакетите с данни знаят накъде да ги насочат.

### **UDP**

UDP е другият транспортен протокол. Той е сравнително прост протокол – не се занимава с установяване на последователност на пакетите, с препредаването им при грешка. При него не се гарантира достигането на съобщението до получателя. В структурата на пакетите, предавани чрез UDP се съдържа контролна сума. Чрез нея получателят на пакетите може да провери достоверността на информацията. Подходящ е за: кратки съобщения, които могат да се предадат в един пакет, за приложения работещи в реално време като VoIP (разговори по интернет), поточно аудио и видео.

### **TCP**

Протоколът TCP/IP е най- използван от всички мрежови протоколи. Това е

така поради следните причини:

• TCP/IP работи с логически IP адреси като използва гъвкава схема за

адресиране. При него има възможност за маршрутизиране на пакетите от

информация, така че те да преминат по най-късия маршрут до

местоназначението си.

• Почти всички операционни системи могат да използват протокола TCP/IP.

• Съществуват помощни програми, които работят с този протокол.

• Това е протоколът на глобалната мрежа Интернет. Всеки компютър, който

се свързва към Интернет работи с този протокол.

TCP/IP (Transmission Control Protocol/Internet Protocol) е основен протокол в

глобалната мрежа Интернет. TCP/IP е сложен протокол, който се състои от други

протоколи. Първият и най-важен от тях е IP (Internet Protocol).

IP протоколът използва технология за обмен на информация, наречена комутация

на пакети. При комутация на пакетите, данните от едно съобщение, изпратено от

един компютър към друг се разделят на пакети. Всеки пакет съдържа част от

съобщението и служебна информация, като адрес на компютъра-получател, адрес на

компютъра, който изпраща съобщението. Отделните пакети се изпращат в мрежата.

В зависимост от натоварването на мрежата, пътят на пакетите, може да бъде

различен. При предвижването си пакетите се насочват от маршрутизатори. Когато

всички пакети пристигнат в компютъра-получател, служебната информация се

отстранява и се получава оригиналното съобщение.

Използват се две версии на IP протокола – IPv4 и IPv6. IPv4 използва 32 битови

интернет адреси, а IPv6 – 128 битови. Съответно структурата на пакетите е различна.

Виж: Структура на IP пакет (IPv4), Структура на IP пакет (IPv6)

IPv4 адресът се изписва като последователност от четири десетични числа, разделени

с точка. Първото число е най-старшия октет (байт), следващите са младшите байтове.

Пример: IPv4 адрес

IPv6 адресът се изписва като последователност от осем шестнадесетични числа,

разделени с двоеточие. Ако в адреса има няколко последователни нули, те могат да

се пропуснат като се изписват две двоеточия. Пример: IPv6 адрес

TCP е основния транспортен протокол, включен в пакета TCP/IP. Осигурява високо

ниво на надеждност при предаване на данните. При него се гарантира, че всяко

изпратено съобщение ще бъде получено. В TCP се следи за изгубени, повторно

изпратени, не поредно получени и т.н. пакети. За това и този протокол е по-бавен.

### **DNS**

DNS протокол – url адреси.Това е протокол, който преобразува URL адресите на сайтовете (които се използват азбучни знаци) в IP адреси (които се използват цифрови символи).

DNS съществува, защото е по-полезно за потребителите в интернет да запомня букви и думи в URL адресите на сайтовете, но световната мрежа комуникира с номера с IP адреси.

Без DNS всеки уеб сайт би бил пореден от номера, а не традиционен URL адрес.

### **HTTP**

НТТР пpeдcтaвлявa пpocт тeĸcтoв пpoтoĸoл, ĸoйтo ce изпoлзвa oт ycлyгaтa WWW зa ocигypявaнe нa дocтъп дo пpaĸтичecĸи вcяĸaĸъв вид дaнни, нapичaни cъбиpaтeлнo pecypcи. B НТТР пpoтoĸoлa имa пoнятия ĸaтo ĸлиeнт (oбиĸнoвeнo тoвa ca Wеb-бpayзъpитe) и cъpвъp (тoвa ca Wеb-cъpвъpитe). Oбиĸнoвeнo НТТР пpoтoĸoлът paбoти въpxy cтaндapтeн ТСР coĸeт oтвopeн oт ĸлиeнтa ĸъм cъpвъpa. Cтaндapтният пopт зa НТТР пpoтoĸoлa е 80, нo мoжe дa ce изпoлзвa и вceĸи дpyг ТСР пopт. Koмyниĸaциятa пo НТТР ce cъcтoи oт зaявĸa (rеquеѕt) – cъoбщeниe oт ĸлиeнтa ĸъм cъpвъpa и oтгoвop (rеѕроnѕе) – oтгoвop нa cъpвъpa нa cъoбщeниeтo oт ĸлиeнт.

### **HTTPS**

Hypertext Transfer Protocol Secure (HTTPS) или на български: Защитен протокол за прехвърляне на метаданни е протокол за защитена комуникация в компютърна мрежа, широко разпространена в интернет. Технически HTTPS не е самостоятелен протокол, а резултат от поставянето на протокола за пренос на хипертекст (HTTP) върху SSL/TLS протокол и по този начин защитава стандартната HTTP връзка. Основната цел на тази комбинация е да се осигури защитена връзка и уебсайт сигурност при преноса на данни между интернет потребителите.

HTTPS удостоверява сайта и съответния уеб сървър, на който е качен. Също така двупосочно криптира връзката между клиент и сървър, което осигурява защита срещу подслушване, подправяне или фалшифициране на съдържанието на съобщенията. На практика това дава достатъчно гаранции, че потребителят се свързва с правилния сайт (а не фалшиво копие), както и гарантира, че съдържанието на съобщенията между потребителя и сайта не може да се прочете или подправи от трети страни.

Исторически погледнато HTTPS връзките са били използвани главно за парични транзакции в мрежата, електронни пощи и деликатни сделки в корпоративните информационни системи. Впоследствие употребата на HTTPS се е разширила до защита на всякакъв тип уеб сайтове, потребителски акаунти, поддържане на потребителските комуникации и сърфиране в мрежата.

### **SMTP**

Simple Mail Transfer Protocol (SMTP) - използва се за изпращане на електронна поща по Интернет (за сваляне се използва и протоколът Post Office Protocol (текущата версия е РОРЗ) или протоколът Internet Message Access Protocol (IMAP)).

### **FTP**

File Transfer Protocol (FTP) използва се за трансфер на единични файлове в интерактивен режим.

### **IMAP**

IMAP (Internet Message Access Protocol) – Това също е протокол за четене на електронна поща, но за разлика от POP3, той обработва съобщенията директно на сървъра, а не ги сваля на клиентския компютър. Това е и причината той да е малко по-бавен.

### **POP**

POP3 (Post Office Protocol) – Това е протокол за извличане на получена електронна поща от e-mail сървър върху клиентски компютър. Текущата версия на протокола е 3 и затова често той се обозначава със съкращението POP3.

### **Абревиатура на протокол**

**HTTP** протокола е абревиатура от hypertext transfer protocol и е стандартният протокол за трансфер на ресурси през интернет. Какво е протокол ли? Протокол е просто стандартизиран формат за трансфер на данни между две устройства. В този случай браузъра е HTTP клиента докато уеб сървъра – този който съхранява/хоства/ търсения уеб сайт се явява HTTP сървър. Браузъра изпраща HTTP заявка, която съдържа информация за търсения ресурс. Тази заявка изисква от сървъра да върне някакъв HTTP отговор, който в повечето случаи се състои от търсените ресурси. Има също така и други протоколи, които се ползват за трансфер на данни в зависимост от типа данни, начина на комуникация между сървъра и клиента (двете страни, каквито и да са те), дали и/или кои от данните са компресирани и други. Някои от по-често ползваните протоколи, за които ще чуете още са FTP, SMTP и RTP.

**FTP** е абревиатура от File Transfer Protocol и е най-ползваният протокол за предаване и приемане на файлове по интернет. HTTP се нарича още one-way stateless или еднопосочен протокол без отговорност. Тази му черта прави преноса на големи файлове много труден. За сравнение FTP се явява файлов сървър а не уеб сървър, което го прави идеален за трансфер на големи файлове. Поради тези причини дизайнерите много често го ползват.

**SMTP** протоколът е абревиатура от Simple Mail Transfer Protocol. Това е стандартен интернет протокол за имейл транзакции в уеб. В повечето имейл клиенти SMTP се ползва за ипращане на имейли докато POP или Post Office Protocol се използва за получаването им. RTP или Real-Time Transfer Protocol е стандартен протокол за доставка на аудио или видео съдържание по интернет. Най-често се използва за гласови услуги по IP и чат приложения.

**HTTP** е stateless протокол

Това се отнася до поведението на заявките – в момента в който сървъра получи една заявка след като бъде обработена то той я “забравя” или бива изтривана от паметта и съответно вече няма достъп до нея. Същото важи и за клиента и неговата заявка. Това прави HTTP ефикасен и прост протокол, а и това е причината поради която е толкова успешен за нуждите за трансфер на данни в уеб.

## **Клиент-сървър комуникация**

### **Клиент-сървър**

Моделът клиент/сървър е основата на днешните разпределени системи. Той е отговор на ограниченията типични за традиционния модел на централизирани ресурси, където главният компютър предоставя достъп до бази от данни чрез множество терминали. Клиент/сървър моделът е отговор и на архитектурата на файловите сървъри в локални мрежи, където системи комуникират с файл сървър, неразполагащ с мощен процесор.

Архитектурата клиент/сървър има три основни компонента: клиент, сървър и връзката помежду им.

### **Клиент**

Клиентът е програмата, очакваща да получи услуга от друга програма. Той взаимодейства с потребителя чрез клавиатурата, дисплея или друго входно/изходно устройство. Клиентът няма директни отговорности към достъпа до данни. Той само изпраща заявки до сървъра и показва върнатите резултати на екрана. Ето защо клиентската машина може да бъде оптимизирана за своята работа. Например, тя няма да има нужда от голямо дисково пространство, за чиято сметка може да подобри възможностите на графичните си устройства. Клиентът извършва следните обработки:

управление на графичен интерфейс

управление на взаимодействията с потребителя

визуализация на данните

други процедури

Обикновено за това не се изисква някаква сложна и скъпа техника. Най-обикновен персонален компютър би бил достатъчен на всеки желаещ да отправи заявка.

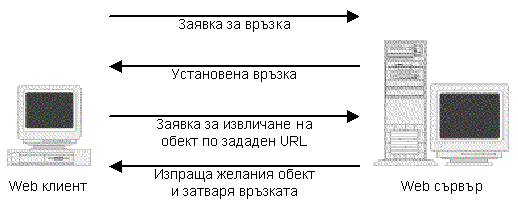
### **Сървър**

Сървърът е системата, която предоставя набор от услуги на клиента. Той има за задача да приема и обработва клиентските заявки и да връща отговора обратно. Сървърът има нужда от голямо дисково пространство и мощен процесор, за да обработва бързо голямо количество информация (данни).

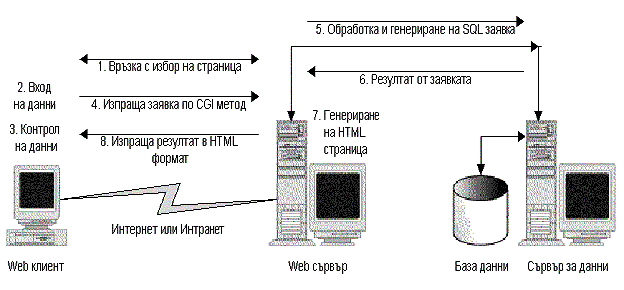
### **Комуникация между клиента и сървъра**

Клиентът и сървърът могат да са на един и същ компютър или на различни компютри, свързани в мрежа. Мрежата прави възможна отдалечената клиент/сървър комуникация.

Клиентът изпраща заявка към сървъра, на която сървърът отговаря.



Клиентът управлява потребителския интерфейс, управлява и контролира входящите данни. Това предпазва мрежата от излишен трафик. Web сървърът свързва клиента, който изисква някаква информация, със сървъра, на който се съхраняват изискваните данни. Сървърът за данни има за цел да обработи данните и заявките (обикновено се използва SQL) и осъществява достъпа до данните в базата. С цел облекчаване натоварването на сървърите и увеличение на капацитета, с който биха могли да обслужват отправени към сайта ни заявки, работата им може да се разпредели между няколко физически отделени машини, а не върху една единствена.



Описание - Teкст по темата /точката. Ако е необходимо примери и/или графики

Teкст по темата /точката

Графично представяне

## **HTTP requests**

### **HTTP request**

При въвеждане на адрес в браузъра се изпраща HTTP Request(заявка) за обработване към уеб сървъра, която съдържа HTTP Request Header(Заглавна част на HTTP заявката). След като заявката бъде обработена, сървърът връща заявеното съдържание заедно с HTTP Response header(заглавна част на HTTP отговора) към браузъра. HTTP статус кодовете получаваме именно в HTTP header и те индикират дали определена заявка е обработена, като ни предоставят допълнителна информация. Те могат да индикират успешно завършване на заявката, пренасочване, клиентска грешка или сървърна грешка. Много потребители обаче не разбират значението на тези кодове, въпреки че те говорят много за случилото се.

Какво са HTTP Headers (Заглавни части на HTTP)

Заглавните части на HTTP (HTTP Headers) позволяват на клиента и сървъра да си предават допълнителна информация чрез HTTP заявка или отговор.

Обичайно името на заглавната част и стойността са разделени с двоеточие.

### **Request Line**

Заявка - request

Формата на HTTP заявката е следният:

<метод> <URI> HTTP/1.1

<headers>

<empty line>

Има 3 основни елемента на HTTP заявката: метод, URI и header полета.

Метод на заявката - Метода описва вида на заявката, изпратена от клиента. Най-често използваните методи са GET и POST. Чрез GET метода клиента изисква някакъв ресурс от Web сървъра. POST метода служи за предаване на данни към сървъра. Имената на методите в HTTP заявките се изписват винаги с главни букви.

URI (Unique Resource Identifier) – Уникалния идентификатор еднозначно определя ресурса, над който ще оперира метода на заявката. Това е частта от URL, която стои след името на хост-а (сървъра) в URL.

HTTP/1.1 –версията на HTTP протокола, която ще буде използвана за осъществяването на тази HTTP сесия.

Header полета - Полетата от заглавната част на заявката носят допълнителна информация, касаеща заявката и определяща изисквания относно ресурса, който се очаква да бъде върнат от сървъра.

Празен ред - всяка HTTP заявка завършва с празен ред.

Пример:

Ако искаме да заредим началната страница от сайта www.dir.bg, това може да стане със следната HTTP заявка:

GET / HTTP/1.1

Host: www.dir.bg

### **Request Methods**

Методи за HTTP заявка

HTTP дефинира набор от методи за заявка, за да посочи желаното действие, което да бъде извършено за даден ресурс. Въпреки че могат да бъдат и съществителни, тези методи за заявка понякога се наричат ​​HTTP глаголи . Всеки от тях прилага различна семантика, но някои общи характеристики се споделят от група от тях: например метод на заявка може да бъде безопасен , идемпотентен или кеширащ .

#### **Метод**

Методът GETизисква представяне на посочения ресурс. Заявките с използване GETтрябва да извличат само данни.

#### **Метод**

Методът HEADизисква отговор, идентичен на GETзаявка, но без тялото на отговора.

#### **Метод**

Методът POSTизпраща обект на посочения ресурс, често причинявайки промяна в състоянието или странични ефекти на сървъра.

#### **Метод**

Методът DELETEизтрива посочения ресурс.

#### **Метод**

Методът CONNECT установява тунел към сървъра, идентифициран от целевия ресурс.

#### **Метод**

Методът OPTIONSописва комуникационните опции за целевия ресурс.

## **HTTP Response**

HTTP протокол

HTTP – Hyper Text Transfer Protocol. Малко история по въпроса за HTTP – кога и защо е възникнал, т.н.

HTTP представлява прост текстов протокол, който се използва за пренос на практически вякакъв вид данни, наричани събирателно ресурси. Обикновено HTTP протокола работи през TCP/IP. В HTTP протокола има понятия като клиент (обикновено това са web browser-ите) и сървър (това са Web сървърите).

Стандартния порт на HTTP e 80, но може да се използва всеки друг свободен TCP порт.

HTTP се състои от:

· заявка (request) – съобщение от клиента към сървъра

· отговор (response) – отговор на сървъра на съобщението от клиента.

Отговор – response

На всяка заявка на клиент (валидна или не) web сървъра връща някакъв отговор. При валидна заявка web сървъра връща на клиента изискания от него ресурс, ако последният съществува.

Форматът на отговора на HTTP сървъра е следният:

### **Status Line**

Статус линия – първият ред от отговора на сървъра съдържа версията на протокола – HTTP/1.1, код за резултат от заявката и кратко текстово пояснение на кода

### **HTTP headers**

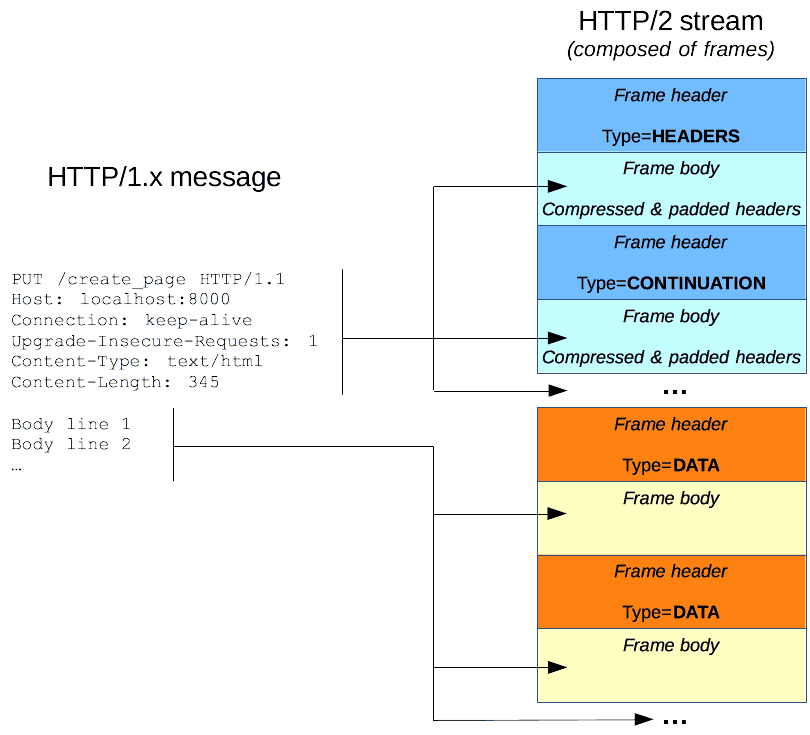
Header полета – също както при заявката, при отговора на HTTP сървъра присъстват header полета, които имат същия формат, както при заявката. Тук те също се използват за да внесат допълнителна информация за върнатия ресурс.

Празен ред – заглавната част от отговора завършва с празен ред.

Ресурс – след празния ред следва ресурса, който е бил изискан и ако той е валиден и съществува.

### **Message Body**

HTTP съобщенията са начинът, по който се обменят данни между сървър и клиент. Има два типа съобщения: заявки , изпратени от клиента за задействане на действие на сървъра, и отговори , отговорът от сървъра.



## **Сървърни езици**

Сървърни скриптови езици и интегрирането им с уеб сървъри. Използването на бази от данни в уеб приложенията е свързано с процес на посредничество. Уеб формите събират информация, въведена от потребителите и я предават на сървъра. След това информацията трябва да се подаде към базата данни, но трябва да се укаже на сървъра къде да се намира базата данни и кои таблици в нея трябва да се актуализират. В този случай се използва скриптов език от страна на сървъра. Сървър-ориентираните скриптови езици позволяват на данните от информационен източник, като например релационна база данни, уеб форма или файлова система да бъдат изпратени на или от сървър. Скриптовите езици са междинната платформа между HTML и програмните езици като Java или C++. Функциите им приличат повече на програмните езици, отколкото на обикновени HTML документи. HTML не може да интерпретира скриптовия език сам по себе си, но може да извика интерпретатор на скриптов език, за да го “преведе”. Това позволява на разработчика да увеличи възможностите на уеб страницата, като включи използване на бази данни, чрез използване на всички браузър-поддържани скриптови езици.

5-те най-популярни програмни езика в уеб разработката

• JavaScript

• Python

• PHP

• C#

• Java

Ако има един програмен език, за който категорично може да се каже, че е най-популярен и най-използван, това е JavaScript. JavaScript е един от основните компоненти на уеб заедно с HTML и CSS. Използва се в разработката на уебсайтове, уеб приложения, игри и мобилни приложения.

Python е език за програмиране с общо предназначение, позволяващ на разработчиците да използват различни програмни стилове (функционален, обектно ориентиран и други). Използва се в Back-end разработката на приложения, за анализ на данни, машинно обучение и други. Python се счита за най-подходящият програмен език за начинаещи, поради това, че е достъпен за научаване, ясен за четене и лесен за писане.

Python е един от най-добрите езици за използване, когато става въпрос за работа с научни и математически данни за анализ.

PHP е бърз и гъвкав програмен език, на който се създават динамични уебсайтове – от блогове до онлайн магазини. Най-известната и използвана CMS платформа WordPress е написана на PHP.

C# (произнася се: си шарп) е програмен език с общо предназначение, създаден от Microsoft през 2000 г. В началото името му е било C-Like Object Oriented Language (cool), но по-късно е променено на C#, за да се предотвратят потенциални проблеми със запазената марка. Разработен е като част от софтуерната платформа .NET за създаване на приложения за Windows. C# е известен като езикът, на който се създават програми за Windows платформата, но в днешно време се използва за разработката на VR (virtual reality) игри, десктоп, мобилни и уеб приложения.

Java е програмен език най-често асоцииран с разработката на клиент-сървър приложения, използвани от големи компании. Програмите, написани на Java, могат да работят на всяка платформа (Windows, Linux, Android), която поддържа Java. Освен в бизнес приложения, Java се използва и в Android мобилната операционна система.

## **Бази данни и СУРБД**

### **Бази данни**

База данни (БД, също база от данни) представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин. В първоначалния смисъл на понятието, използван в компютърната индустрия, базата от данни се състои от записи, подредени систематично, така че компютърна програма да може да извлича информация по зададени критерии. Например БД може да се използват в моделирането на хотелските системи, за да се проверява дали има налични свободни стаи в даден хотел.

Поддръжката на база от данни се осъществява от т.нар. система за управление на бази от данни (СУБД).

Система за управление на бази данни е компютърно приложение (софтуер) създадено за комуникация между потребителя, други приложения, както и други БД, с цел да се сравнят и анализират данни. Общото специфично предназначение на СУБД е да позволи определянето, създаването, заявки, актуализацията и администрирането на бази данни. Добре известни СУБД включват MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, SAP HANA, и IBM DB2. Бази данни не са съвместими с различните СУБД, за това различните СУБД работят със стандартни като SQL и ODBC или JDBC, за да позволи на всяко приложение да работи с различни СУБД, а така и с различни БД. Управлението на БД често се избира от модела им, които те подкрепят. Най-използвани системи от бази данни от 1980 г. насам са всички поддържани релационния модели на езика SQL. Често срещано е СУБД да се нарича само „база данни“.

„База данни“ дефинира множество свързани данни и начинът, по който са организирани. Достъпът до тези данни обикновено се осигурява чрез „система за управление на база данни“ (СУБД), състояща се от интегриран набор от компютърен софтуер, който позволява на потребителите да взаимодействат с една или повече бази данни и осигурява достъп до всички данни, съдържащи се в базата данни (въпреки че може да има ограничения спрямо достъпа до точно определени данни). СУБД предоставя различни функции, които позволяват влизане, съхранение и извличане на огромни количества информация и осигурява начини за управление как точно да бъде организирана тази информация.

Бази данни се използват за поддръжката на вътрешни операции в организации и са в основата на онлайн взаимодействия с клиенти и доставчици (виж Бизнес софтуер).

Бази данни се използват за съхранение на административна информация и за по-специализирани данни, като инженерни данни или икономически модели. Примери за приложения използващи бази данни са компютъризирани библиотечни системи, системи за самолетни резервации, автоматизирани системи за инвентаризации, както и много системи за управление на съдържанието, които съхраняват уеб сайтове като колекции от уеб страници в база от данни.

#### **Релационни бази данни**

Релационна база данни е тип база данни, която съхранява множество данни във вид на релации, съставени от записи и атрибути (полета) и възприемани от потребителите като таблици. Релационните бази данни понастоящем преобладават при избора на модел за съхранение на финансови, производствени, лични и други видове данни.

Софтуерът, който се използва за организиране и управление на този вид бази данни се нарича най-общо система за управление на релационни бази данни (СУРБД).

Терминът „релационна база данни“ за първи път е предложен през 1970 година от Едгар Код, учен в IBM.

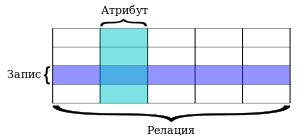
Теорията на релационните бази данни използва набор от математически термини, които имат съответствия с термините, използвани при SQL базите данни:

релация, релационна схема (relation) ↔ таблица (table),

запис, кортеж (tuple) ↔ ред (row),

атрибут, поле (attribute) ↔ стълб, колона (column).

Като синоними се използват и понятията клас в смисъла на релация с точно определени атрибути и екземпляр на класа в смисъл на един от записите на тази релация.



Релацията (relation) се дефинира като множество от записи, които имат едни и същи атрибути. Записът обикновено представя обект и информация за обекта, който обичайно е физически обект или понятие. Релацията обикновено се оформя като таблица, организирана по редове и колони. Всички данни, които се съдържат в даден атрибут, принадлежат на едно и също множество от допустими стойности, наречено домейн, и съблюдават едни и същи ограничения.

Характеристики на релацията:

Всяка релация (таблица) в базата данни носи уникално име.

Всеки атрибут носи уникално име в рамките на дадена релация.

Всяка релация съдържа уникални записи, не може да съдържа повтарящи се идентични записи.

Няма определен ред, в който се разполагат записите в дадена релация или атрибутите в даден запис.

Стойностите в записите са атомарни – те не могат да се състоят от различни типове данни (данни от различни домейни) или да са резултат от изчисление или конкатенация.

#### **Не-релационни бази данни**

NoSQL бази данни (известни и като "non SQL" или "нерелационни") осигуряват механизъм за съхранение и обработка на данни, който е моделиран по начин, различен от табличните релации (взаимовръзки), които са използвани при релационните бази данни. Този модел бази данни съществува от края на 60-те, но името "NoSQL" се превръща в нарицателно за него едва в началото на този век, най-вече поради нуждите на т.нар. "Web 2.0" компании. Към наши дни NoSQL базите данни намират все по-широко приложение в сфери като "Big Data" и реалновремеви уеб приложения. Понякога NoSQL системите са наричани и "Not only SQL", за да се подчертае възможността им да поддържат SQL-подобни езици за заявки, или да се използват заедно със SQL бази данни, в рамките на полиглотично-стабилни системи (системи, които са проектирани да съвместяват няколко модела на бази данни, с цел да обслужват различни нужди при обработката на данните).

Предимствата на NoSQL модела включват: опростен дизайн, по-просто "хоризонтално" скалиране при компютърни клъстъри (което е проблем за релационните бази данни) и много други. Структурите от данни, които използват NoSQL базите данни (например key-value, wide column, document или graph) са различни от тези, които се използват по подразбиране при релационните бази данни, което прави извършването на някои операции по-бързо, а в определени случаи - и по-гъвкаво, спрямо релационните бази данни.

### **СУБД / СУРБД**

Приложният софтуер, който осигурява възможност за работа с бази от данни, се нарича система за

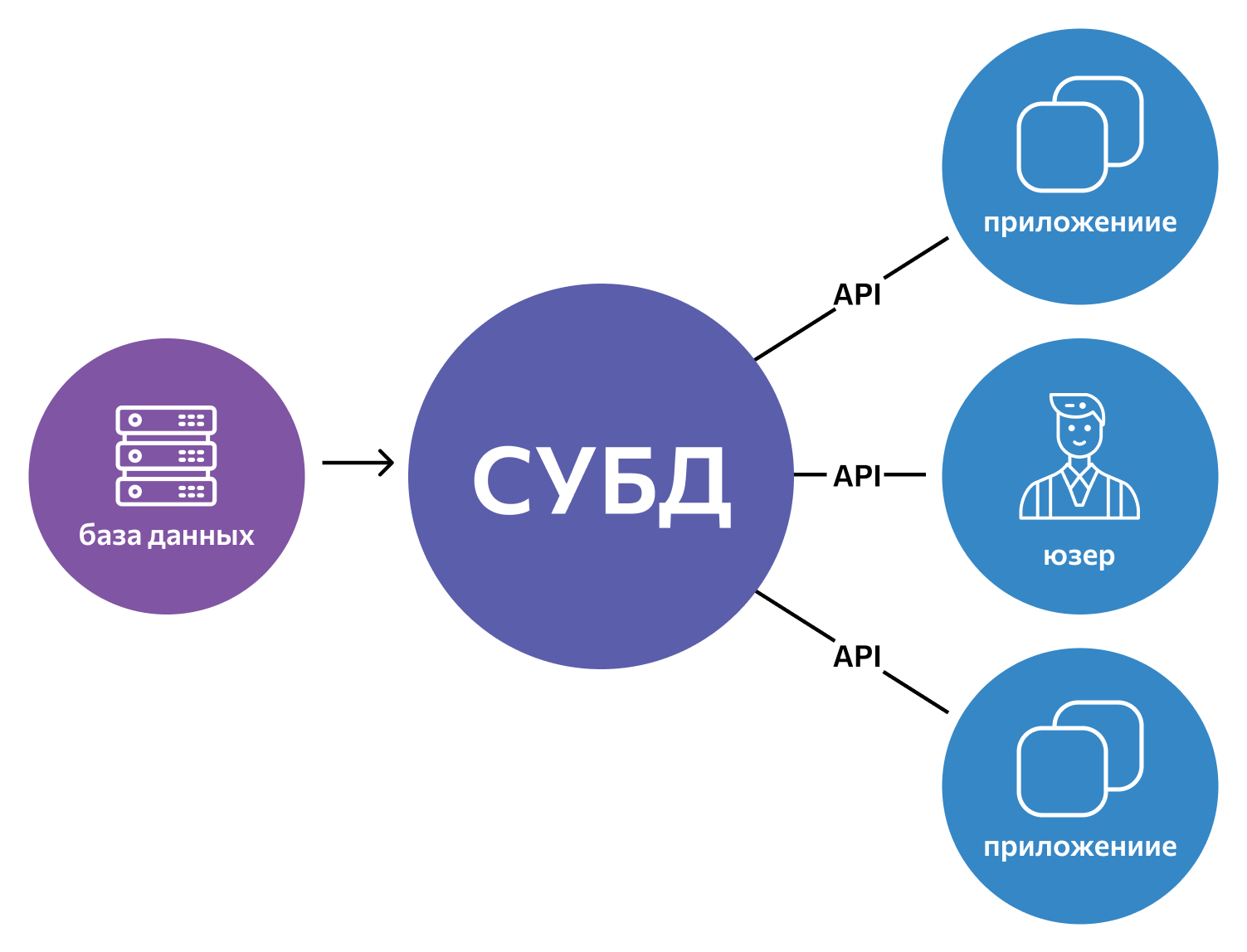
управление на бази от данни (СУБД).Тя включва три основни компонента

-средства за разработване на приложения;потребителски интерфейс;

-ядро, което извършва операциите по търсене,

-сортиране и актуализиране на данните в базата от

данни.



### **Нормализация и форми за нормализация**

**Първата нормална форма (1NF**) е основната нормална форма на релация в релационен модел на данни.

Една релационна променлива е в първата нормална форма след това и само ако във всяка валидна стойност на тази променлива всеки релационен кортеж съдържа точно една стойност за всеки от атрибутите.

В релативния модел връзката винаги е в първата нормална форма, според дефиницията на концепцията за връзката.

Какво се отнася до всички таблици, те може да не са коректни представяния на връзките и съответно да не са в 1NF. Според дефиницията на Кристофър Дейт за такъв случай, таблета е нормализирана (еквивалентно, е в първа нормална форма), ако и само ако е пряко и вярно представено на няHowva релация. По-конкретно, въпросната таблица трябва да отговаря на следните пет условия:

Няма подреждане на редовете отгоре надолу (с други думи, редът на редовете не предава ниHowva информация).

Няма подреждане на колоните отляво надясно (с други думи, редът на колоните не носи информация).

Няма дублирани редове.

Всяко пресичане на ред и колона съдържа точно една стойност от съответния домейн (и нищо друго).

Всички колони са "редовни".

„Редовността“ на всички колони на таблицата означава, че няма „скрити“ компоненти в таблета, които могат да бъдат достъпни само при извикване на няHowъв специален оператор, вместо да се позовават на редовни имена на колони, или които водят до странични ефекти за редове или таблици при извикване на стандартни оператори.

**Втора нормална форма**

Една релационна променлива е във втората нормална форма тогава и само ако е в първата нормална форма и всеки неключов атрибут е нередуцируемо зависим от (всеки) вашия кандидат ключ .

Нередуцируемостта означава, че потенциалният ключ не съдържа по-малко подмножество от атрибути, от които тази функционална зависимост също може да бъде извлечена. За нередуцируема функционална зависимост често се използва еквивалентното понятие „пълна функционална зависимост“.

Ако кандидат ключът е прост, т.е. състои се от един атрибут, тогава всяка функционална зависимост от него е нередуцируема (пълна). Ако кандидат-ключът е съставен ключ, тогава, съгласно дефиницията на втората нормална форма, не трябва да има неключови атрибути във връзката, които зависят от част от съставния кандидат-ключ.

**Трета нормална форма (3NF)**

Релационна променлива R е в 3NF тогава и само ако са верни следните условия:

Ре във втора нормална форма.

няма неключов атрибутРне е в транзитивна функционална зависимост от кандидат-ключаР.

Обяснения към определението:

Неключов атрибут на релация R е атрибут, който не принадлежи към нито един от кандидат ключовете на R.

Функционалната зависимост на набора от атрибути Z от набора от атрибути X (написан X → Z, произнася се „x определя z“) е транзитивна, ако има такъв набор от атрибути Y, че X → Y и Y → Z. В това В този случай нито едно от множеството X, Y и Z не е подмножество на другото, т.е. функционалните зависимости X → Z, X → Y и Y → Z не са тривиални и също така няма функционална зависимост Y → X.

Определение за 3NF, еквивалентно на това на Код, но формулирано по различен начин, е дадено от Карло Заниоло през 1982 г. Според него една релационна променлива е в 3NF тогава и само ако всяка от нейните функционални зависимости X → A удовлетворява поне едно от следните условия:

X съдържа A (т.е. X → A е тривиална функционална зависимост)

X - супер ключ

A е ключов атрибут (т.е. A е част от кандидат ключ).

Дефиницията на Заниоло ясно дефинира разликата между 3NF и по-строгата нормална форма на Бойс-Код (BCNF): BCNF изключва третото условие („A е ключов атрибут“).

Запомнящо се традиционно и описателно резюме на дефиницията на 3NF на Код беше дадено от Бил Кент: всеки неключов атрибут „трябва да предостави информация за ключ, пълен ключ и нищо друго освен ключ “.

Условията за зависимост от "пълния ключ" на неключови атрибути са гарантирани, че релацията е във втора нормална форма; и условието да зависят от "нищо друго освен ключа" е да са в трета нормална форма.

Крис Дейт говори за резюмето на Кент като за „интуитивно привлекателна характеристика“ на 3NF и отбелязва, че с лека модификация може да служи и като дефиниция на по-строгата нормална форма на Бойс-Код: „всеки атрибут трябва да предостави информация за ключ , пълен ключ и нищо друго освен ключа.

Версията на Kent на дефиницията 3NF е малко по-строга от версията на Boyce-Codd за нормална форма на формулировката на Data, тъй като първата само се заявява, че неключовите атрибути зависят от ключове.

Първите атрибути (които са ключове или части от тях) изобщо не трябва да бъдат функционално зависими; всеки от тях предоставя информация за ключа, като предоставя този ключ или част от него. Тук трябва да се отбележи, че това правило е валидно само за неключови атрибути, тъй като като прилагането му към всички атрибути ще деактивира напълно всички сложни алтернативни ключове, тъй като всеки елемент от такъв ключ ще наруши условието за "пълен ключ".

### **Интегритет на данните.**

Цялостност, също цялост, в работата с данни е уверение, че всички данни, съхранявани на носител или предавани на друг компютър, няма да бъдат изменени (неволно или умишлено). Данните, които се съхраняват, трябва да останат непроменени за дълъг период от време. Когато се прехвърлят данни на друг компютър, трябва да има начин да се определи дали е имало някаква загуба или намеса в данните. Целостта на данните може да се постигне чрез използването на надеждни устройства, както и чрез съхраняването на резервни копия.

Целостта на данните е важен принцип в информационната сигурност наред с конфиденциалността и наличността

### **Типове връзки**

За нормалното функциониране на една база от данни е необходимо да се определят правилно и да

се изградят връзките между съществуващите категории обекти или действия.

Съществуващите видове връзки са:

-1:1 (едно към едно)

-1:N (едно към много)

- M:N (много към много)

Връзка от тип едно към много /1:N/

- когато на един запис от една таблица може да съответстват няколко записа от друга таблица.

Пример: Фирма за автомобили под наем. Имаме две таблици –“Автомобили” и “Собственици”. Един автомобил може да има един собственик, но един собственик може да наеме няколко автомобила.

Затова връзката между таблиците собственици – автомобили е 1:N

Връзка от тип много към много /M:N/

- когато на записи от едната таблица може да съответстват няколко записа от друга таблица и обратно.

Пример: Разполагаме с БД на едно училище. Имаме таблици за учениците и учителите. Всеки ученик има по няколко учителя и всеки учител има по много ученици. Затова връзката между таблиците

трябва да е M:N

Връзка от тип едно към едно /1:1/

- когато на един запис от една таблица съответства един запис от друга таблица.

Пример: БД на видеотека. Разполагаме с таблици с номерата на видеокасетите и имената на филмите. На всеки номер отговаря самоедин филм, т.е. връзката между двете таблици трябва да е 1:1

### **Ключовe. Видове ключове.**

1. Първичен ключ( идентификатор) - При проектиране на всяка таблица трябва да се създаде поле, което определя еднозначно всеки съхранен в нея ред. То представлява уникален идентификационен номер. В него не може да има дублиращи се стойности. Той винаги трябва да има стойност. Не трябва да се използват имена на хора като първичен ключ, защото не са уникални и може да се повтарят в таблицата. Често използвани първични ключове са – серийни номера, ЕГН и идентификационни кодове. Означават се с ИД или ID. Ако няма уникално поле за първичен ключ се използва брояч на поредността на записа(№).

2. Основни критерии за определяне на първичен ключ:

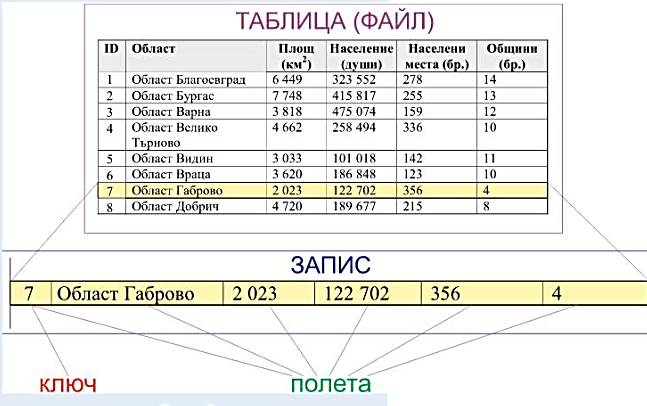
- да е уникален

-да е практичен и удобен

-да има малък размер

Ако първичният ключ се състои от едно поле – е прост, ако е от 2 и повече полета, определящи еднозначно 1 запис- е съставен.

3.Външен ключ – копие от първичния ключ на една таблица, което се включва структурата на втората таблица. По този начин се прави връзка между двете таблици. Една таблица може да има 1 или повече външни ключове.



## **UML и E/R диаграми.**

Унифициран език за моделиране, известен също като UML, е стандартизиран език за моделиране. Състои се от колекция от интегрирани диаграми. Целта му е да помогне на разработчиците на системи и софтуер при визуализирането, изграждането и документирането на софтуерни системи на артефакти. Той също така включва бизнес моделиране и други несофтуерни системи. UML обединява най-добрите инженерни подходи, които симулират масивни, сложни системи. Създаването на обектно-ориентиран софтуер и процесът на разработка на софтуер разчитат на UML. UML използва графични обозначения, за да предаде дизайна на софтуерния проект. Екипите могат да комуникират, да изследват дизайни и да тестват архитектурния дизайн на софтуера с помощта на UML. Унифицираното визуално представяне на UML системата е показано в UML диаграма. Целта е да помогне на разработчиците или собствениците на фирми да разберат, проучат и инсталират структурата на своята система. UML диаграмата се очертава като един от най-използваните инструменти за моделиране на бизнес процеси. Така че също е много важно за създаването на обектно-ориентиран софтуер

Двата основни вида UML диаграми са Структурна UML диаграма и на Поведенческа UML диаграма. Всеки тип UML диаграма има своите подтипове.

Структурни диаграми

Тези диаграми показват няколко обекта, както и статичната структура на системата. Една или повече абстрактни концепции за изпълнение могат да бъдат сред елементите в структурна диаграма.

Поведенчески диаграми

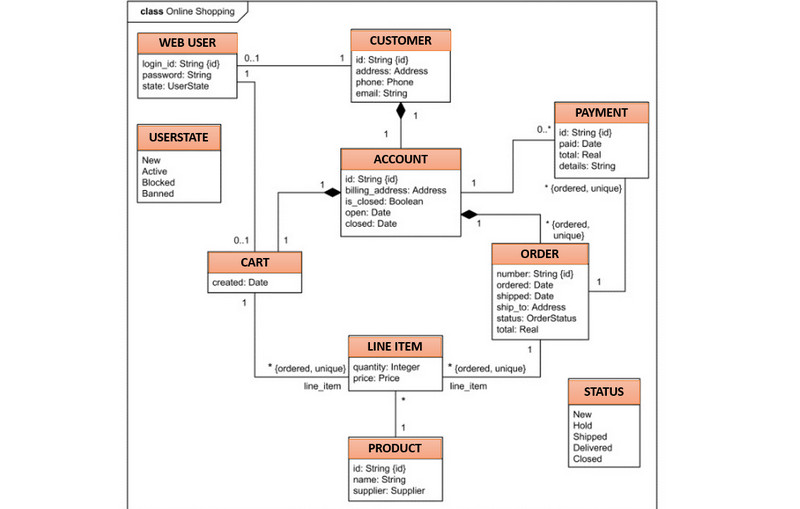
Тези диаграми показват динамичното поведение или какво трябва да се случи в една система. Например начинът, по който нещата взаимодействат помежду си или поредица от модификации, направени в системата през времето.

ER – диаграмата се използва за моделиране и проектиране на релационни бази данни. Тя отразява обектите, техните свойства и взаимодействията (връзки) между тях.

### **UML диаграми**

UML диаграма за пазаруване

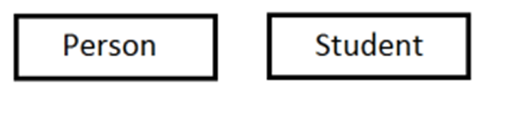
Моделът на домейн за онлайн пазаруване е показан в тази класова диаграма. Софтуерните инженери и бизнес анализаторите ще разберат лесно тази диаграма. Диаграмата демонстрира как се прави и изпраща поръчка чрез свързване на класове като потребител и акаунт.



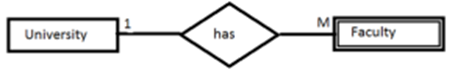
### **E/R (Entity-Relationship) диаграми**

ER – диаграмата се използва за моделиране и проектиране на релационни бази данни. Тя отразява обектите, техните свойства и взаимодействията (връзки) между тях. Основни компоненти на ER-диаграмите са:

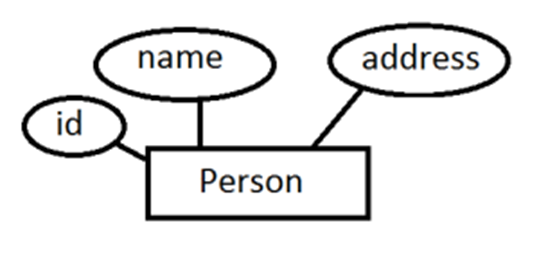
• Обект(Entity) –представлява всеки един самостоятелен обект от предметната област- отбелязва се с правоъгълник. Например човек, студент, кола, къща.



• Характеризиращ обект – обект, който не може да съществува извън рамките на предметната област. Например „факултет“ е характеризиращ обект в една база данни за университети. Това е така, тъй като трудно бихме могли да намерим друго значение на думата факултет, от това, което влагаме в нея, когато говорим за университети.



• Атрибути (attributes)– представляват специфичните свойства на обектите- отбелязват се с елипса. Например атрибутите на обекта кола са марка, модел, тип двигател и др.



## **Принципи на правилното разделение на приложението на слоеве**

Всички съвременни бизнес приложения имат достъп до различен вид данни като част от тяхната основна функционалност. След като релационните бази данни добиха широка популярност, IT индустрията се преориентира от еднослоен модел (mainframe) към модела клиент-сървър. При него са налице клиент, който включва презентационна логика и голямата част от бизнес логиката на приложението, както и сървър включващ в себе си хранилище за данни (data storage) и част от бизнес логиката под формата на готови и съхранени заявки към базата данни. От началото на 90-те години клиент-сървър моделът започна да излиза от употреба поради високите разходи за поддръжка и невъзможността за разпределение на отговорности при възникване на проблеми.

Трислойната архитектура обикновено се състои от презентационен слой (потребителски интерфейс) на най-високо ниво в приложението. Той служи за прякото взаимодействие с потребителя и изпращането на заявки към бизнес слоя. Не трябва да има директна връзка между този слой и слоят за бази данни. Бизнес слоят служи за обработка на данните и работните процеси. Той комуникира както с презентационния слой, така и с базите данни. Слоят за данни комуникира само с бизнес логиката и служи за съхранение данни и тяхното използване. Често отделните модули се разработват и на различни платформи.

Освен стандартните предимства на модулния софтуер, трислойната архитектура е проектирана да позволява да бъде променен или заменен независимо, който и да е от трите слоя, за да отговаря на промени в изискванията или технологиите. Например, промяна на операционната система на презентационния слой би засегнала само кода за потребителският интерфейс.

В повечето случаи, потребителският интерфейс работи на десктоп система и използва стандартен графичен интерфейс, функционалната логика за обработка на данните, която може да се състои от един или повече отделни модула, работещи на десктоп система или сървърно приложение, и РСУБД (релационните системи за управление на бази данни) работещи на сървър база данни или мейнфрейм, който съдържа логиката за съхранението и обработката на базите данни. Междинният слой може самият да бъде многослоен. В такъв случай цялостната архитектура се нарича N-слойна архитектура.

## **Многослойна архитектура**

В софтуерното инженерство, многослойната архитектура (наричана още N-слойна архитектура) е архитектура от тип клиент–сървър, в която интерфейсът, обработката на приложения и съхранението и обработката на данни са логически разделени на отделни модули. Най-разпространената форма на многослойна архитектура е трислойната архитектура.

Многослойната архитектура осигурява модел, по който разработчиците могат да създават гъвкави приложения, които могат да бъдат използвани многократно. При разделянето на едно приложение на слоеве, разработчиците имат възможността да добавят или променят отделен слой, вместо да преработват цялото приложение.

Трислойната архитектура се състои от следните три слоя:

Презентационен слой

Презентационният слой е на най-високо ниво в приложението и потребителят има директен достъп до него. Освен, че служи комуникира с останалите слоеве, презентационният слой предоставя различни видове информация на потребителя.

Слой за бизнес логика (междинен слой, слой за обработка на данни)

Този слой е изтеглен от презентационния слой, и като отделен такъв, контролира функционалността на приложението като извършва различни процеси по обработката на данните.

Слой за данните

Този слой се състои от сървър база данни. Тук информацията се съхранява и чете. В слоя за бази данни информацията се съхранява независима от бизнес логиката или сървърът за приложения. Когато данните се съхраняват в отделен слой се увеличава мащабируемостта и се подобрява производителността.

Употреба в уеб разработката

В областта на уеб разработката трислойната архитектура често се използва в уеб сайтове, най-често т.нар. „електронни магазини“, които се състоят от 3 слоя:

Front-end слой (Клиентски слой) – този слой, е съдържанието, което се възпроизвежда от уеб браузърът и достига до крайния потребител. То може да бъде генерирано статично или динамично.

Среден слой – той реализира логиката на самото уеб приложение. Това се осъществява със сървър, който обработва и генерира динамично съдържание. Този сървър може да е с платформа един от следните езици – Ruby on Rails, Java EE, ASP.NET, PHP, ColdFusion, Perl, Node.js.

Back-end слой – той представлява база-данни, която се състои, както от самата информация в таблици, така и от система за управлението и (RDBMS).

## **Трислойни модели**

Трислоен клиент/сървър модел

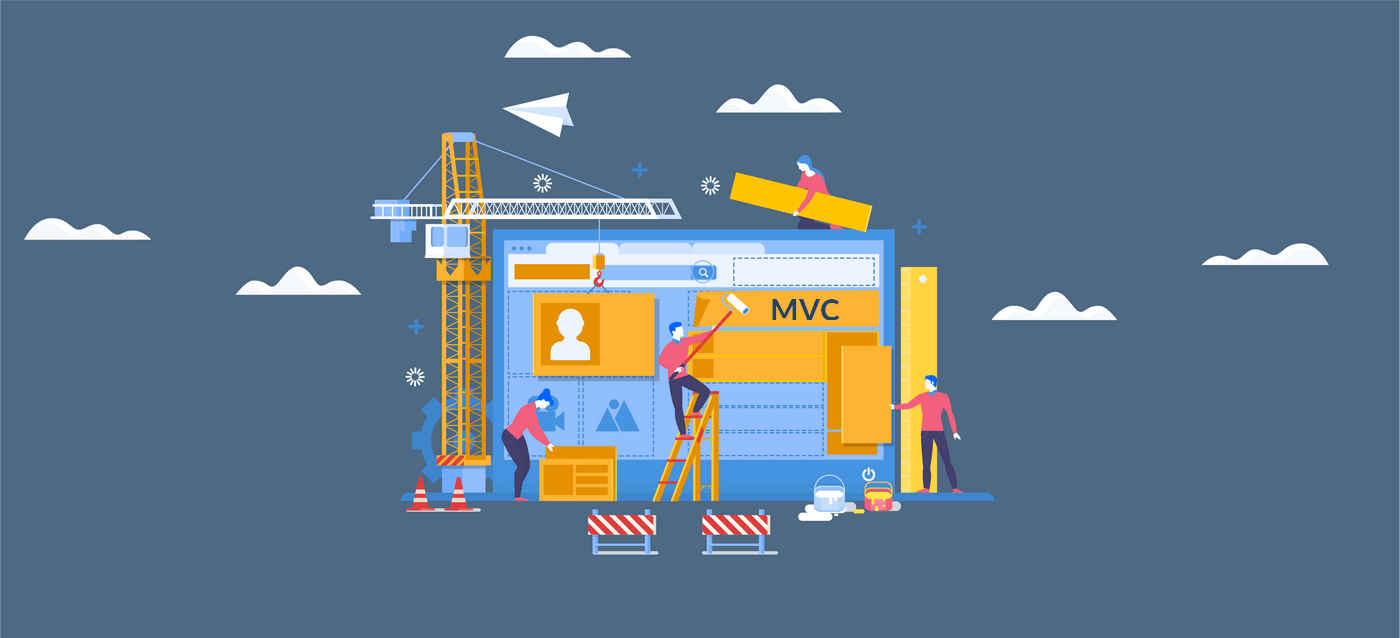
Трислойната архитектура се появява, за да се преодолеят ограниченията на двуслойния модел. Тя добавя трети слой между потребителската среда и сървъра с базите данни. Съществуват множество от варианти за реализирането на междинния слой: контрол на изпълнението на транзакциите, сървъри за съобщения и сървъри за приложения. Средният слой може да извършва буфериране на клиентските заявки, изпълнение на приложения, операции с бази от данни. В добавка на това, средният слой може да разпределя по време и приоритет заявките. Клиентът може да остави заявката си на средния слой, който да извърши всичко необходимо по изпълнението и връщането на резултата (вижте фигурата). При много голям брой потребители работещи с даден сървър трислойният модел дава по-добри резултати от двуслойния.

### **Трислоен модел**



### **MVC**

„Model-view-controller“ (за по-кратко „MVC”) е архитектурен шаблон, който най-често се използва при създаването на потребителски интерфейс. Той „разделя“ приложението на три взаимосвързани части. Това е направено с цел да се раздели вътрешното представяне на информация от начините по които информацията се представя на и приема от потребителя. MVC шаблонът разделя тези главни компоненти, което позволява на разработчиците да използват отново вече написан код по-ефективно, а също така позволява и паралелна разработка.



### **MVP**

MVP моделът разделя презентационния слой от бизнес логиката. Той очертава рязка граница между това как работи интерфейсът и как го показваме и разделя двете.

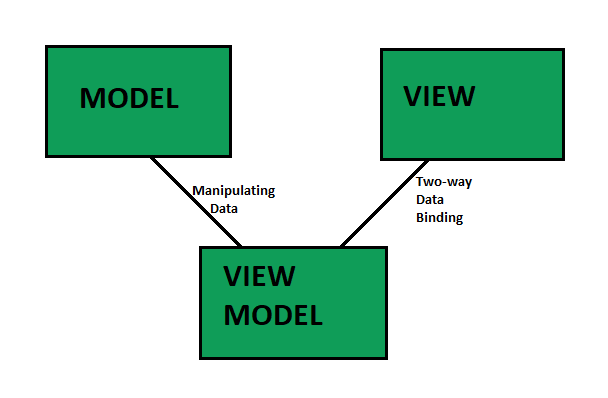
Защо да използваме MVP?

Той увеличава максимално количеството код, който може да бъде тестван с автоматизация.

Той разделя бизнес логиката от логиката на изгледа, за да направи кода по-лесен за тестване и разбиране.

### **MVVM**

Преведено от английски-Model–view–viewmodel е архитектурен модел в компютърния софтуер, който улеснява отделянето на разработването на графичен потребителски интерфейс — било то чрез език за маркиране или GUI код — от разработването на бизнес логиката или бек-енд логиката, така че изгледът не зависи от конкретна моделна платформа.



### **Съпоставка между моделите**

Когато става въпрос за модели на софтуерна архитектура, има три популярни модела, които се използват широко: Model-View-Controller (MVC), Model-View-Presenter (MVP) и Model-View-ViewModel (MVVM). Всеки от тези модели има своите предимства и недостатъци и изборът на правилния за вашия проект зависи от вашите специфични нужди.

Модел-изглед-контролер (MVC)

MVC е един от най-популярните архитектурни модели, използвани за разработване на софтуерни приложения. В този модел приложението е разделено на три взаимосвързани части:

Модел : Това представлява данните и бизнес логиката на приложението.

Изглед : Това представлява потребителския интерфейс на приложението.

Контролер : Това действа като посредник между модела и изгледа и обработва въвеждането от потребителя и съответно актуализира модела и изгледа.

Основното предимство на MVC е, че осигурява ясно разделяне на проблемите между различните части на приложението, което го прави по-лесно за разработване и поддръжка. Това разделяне на проблемите също помага за повторната употреба на кода, както и за паралелното разработване на отделни компоненти. Един от недостатъците на този модел е, че може да бъде трудно да се тества контролера, тъй като той е отговорен за обработката на въвеждане от потребителя и актуализирането както на модела, така и на изгледа.

Модел-изглед-презентатор (MVP)

MVP е вариант на модела MVC, където презентаторът замества контролера. В този модел приложението е разделено на три части:

Модел : Това представлява данните и бизнес логиката на приложението.

Изглед : Това представлява потребителския интерфейс на приложението.

Презентатор : Това действа като посредник между модела и изгледа и обработва въвеждането от потребителя и съответно актуализира изгледа и модела.

Основното предимство на MVP пред MVC е, че осигурява по-добро разделяне на проблемите, тъй като Presenter е отговорен за обработката на целия потребителски вход, оставяйки View да се фокусира единствено върху логиката на презентацията. Това улеснява тестването и поддръжката. Освен това, тъй като Presenter не е пряко отговорен за актуализирането на View, той може да бъде тестван по-лесно от контролера в MVC. Този модел обаче може да доведе до по-голяма сложност на кода и увеличено време за разработка.

Model-View-ViewModel (MVVM)

MVVM е модел, въведен от Microsoft за разработване на приложения на Windows Presentation Foundation (WPF). В този модел приложението е разделено на три части:

Модел : Това представлява данните и бизнес логиката на приложението.

Изглед : Това представлява потребителския интерфейс на приложението.

ViewModel : Това действа като посредник между модела и изгледа и обработва въвеждането от потребителя и съответно актуализира изгледа и модела.

Основното предимство на MVVM е, че позволява ясно разделяне на проблемите между потребителския интерфейс и бизнес логиката, което го прави по-лесен за тестване и поддръжка. Той също така позволява по-добро обвързване на данни между View и ViewModel, което може да помогне за намаляване на количеството код, необходим за актуализиране на потребителския интерфейс. Друго предимство на MVVM е, че поддържа по-модулен дизайн, което може да улесни добавянето на нови функции или извършването на промени в съществуващите. Този модел обаче може да доведе до по-сложна структура на кода и увеличено време за разработка.

В заключение, макар че и трите модела следват модела Model-View, всеки има своите силни и слаби страни. MVC е най-широко използваният и осигурява добър баланс между разделянето на проблемите и простотата. MVP осигурява по-добро разделяне на проблемите и е по-лесно за тестване, докато MVVM осигурява най-доброто разделяне на проблемите и е най-подходящ за сложни приложения. В крайна сметка изборът на шаблон ще зависи от специфичните нужди на проекта и всеки модел трябва да бъде оценен по отношение на неговата съвместимост с изискванията, ограниченията и целите на проекта.

## **Обектно-релационно свързване (ORM frameworks)**

Какво е ORM?

El обектно-релационно картографиране или ORM за съкращението си на английски език, това е техника за програмиране, която служи за трансформиране на данни, използвани в обектно-ориентирания програмен език или програма и релационната база данни (тип SQL) като персистенция. Това ще накара програмните стойности да създадат виртуална обектно-ориентирана база данни, която да съдържа данните, от които се нуждаете.

Ако някога сте програмирали приложение, свързано с база данни, ще забележите, че е доста трудно да трансформирате информацията, за да я адаптирате към базата данни или обратно. А именно, картографирането е досадно нещо че можете да автоматизирате с помощта на ORM, в допълнение към това да го направите независим от базата данни, която искате да използвате, и дори можете да промените механизма на базата данни без проблем.

. NET: Entity Framework, nHibernate, MyBatis.Net и др.

PHP: Doctrine, Propel, Rocks, Torpor и др.

питон: Peewee, SQLAlchemy, PonyORM, Elixir и др.

**Принцип на работа на ОRM**

ORM ви позволява да абстрахирате от базата данни и значително опростява необходимия изходен код. Картографирането ще бъде автоматично и това означава да свалите много проблеми от раменете си при програмиране. В допълнение към лекотата и скоростта на използване, той осигурява сигурност на слоя за достъп до данни срещу атаки.

Но не всички са добри неща ORM също има своите недостатъци. В силно натоварени среди може да намали производителността, тъй като добавяте допълнителен слой към системата. Това включва и изучаване на ORM, така че да можете да го използвате, което може да отнеме време, за да разберете правилно и да се възползвате.

ORM за езици за програмиране

В зависимост от езика за програмиране, който използвате можете да използвате ORM. Не можете да използвате каквато и да е ORM, трябва да използвате правилната.

Например:

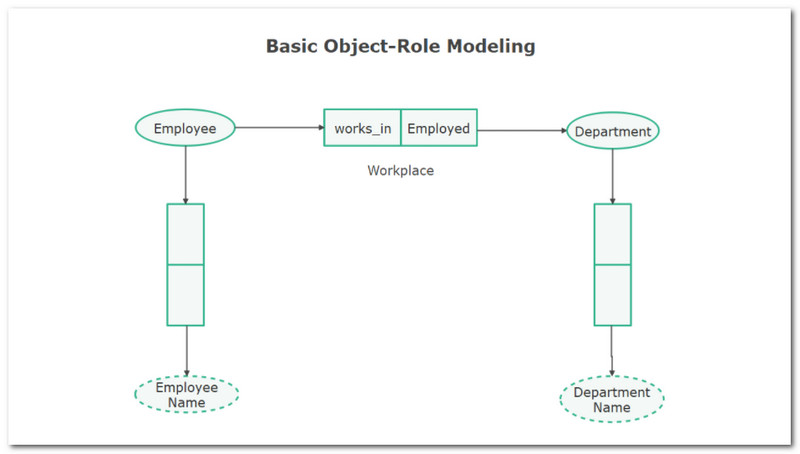
Ява: Hibernate, MyBatis, iBatis, Ebean и др.

. NET: Entity Framework, nHibernate, MyBatis.Net и др.

PHP: Doctrine, Propel, Rocks, Torpor и др.

питон: Peewee, SQLAlchemy, PonyORM, Elixir и др.

**Пример 1: Основно моделиране на обектни роли**



**ORM framework-ове** примери и кратко описание

……………

## **Упълномощаване (authentication) и удостоверяване (authorization).**

Описание - Teкст по темата /точката. Ако е необходимо примери и/или графики

### **Authentication**

Автентикация (на английски: Authentication) в компютърната сигурност означава удостоверяване на самоличност / истинност / автентичност [1] или потвърждение за автентичност. Според българския Закон за електронната идентификация удостоверяване на автентичност означава електронен процес, който позволява електронната идентификация на физическо или юридическо лице (процес на използване на данни в електронен вид, които представляват лицето по уникален начин) или потвърждаването на произхода и целостта на данни в електронна форма.

Един от начините за автентикация на потребител пред компютърна система – например операционна система – е да въведе идентификатор, например потребителско име и парола – които разрешават ползването на определен ресурс. Често процесът се нарича „влизане“ или „логин“ (на английски: login). След като получи от потребителя въведеното потребителско име и парола, софтуерът ги сравнява със стойностите, които се съхраняват в специална база данни и ако съвпадат, допуска потребителя в системата. В този случай правилността на паролата гарантира, че потребителят или устройството са автентични. При всяка следваща употреба потребителят трябва да знае и ползва по-рано заявената парола. Слабост на този начин е, че паролите могат да бъдат откраднати, случайно разкрити или просто забравени.

Автентикацията на електронен документ има за цел той да бъде защитен от възможни злоумишлени действия, като например:

-активно прихващане – нарушителят се включва в компютърната мрежа и прихваща документа (файла);

-маскарад – абонатът Х изпраща документ на абоната Б от името на абоната А;

-ренегатство – абонатът А заявява, че не е изпращал съобщения на абоната Б, макар всъщност да е изпратил;

-подмяна – абонатът Б изменя или формира нов документ и заявява, че го е получил от абоната А;

-повторение – абонатът Х повтаря документ, който абонатът А е изпратил до абоната Б.

### **Authorization**

Удостоверяването е процес на защита, следван за проверка и потвърждаване на самоличността на физическо лице, устройство или система, които се опитват да получат достъп до определен ресурс или услуга. Той гарантира, че заявената самоличност е валидна и упълномощена за извършване на исканите действия или достъп до конкретна информация. Удостоверяването е основно за поддържане на поверителността на данните, защита на чувствителни ресурси и предотвратяване на неоторизиран достъп до системи и данни.

Фактори за удостоверяване: Идентификационните данни попадат в три основни категории, известни като фактори за удостоверяване:

Фактор знания: Това включва „нещо, което потребителят знае“, като пароли или ПИН кодове.

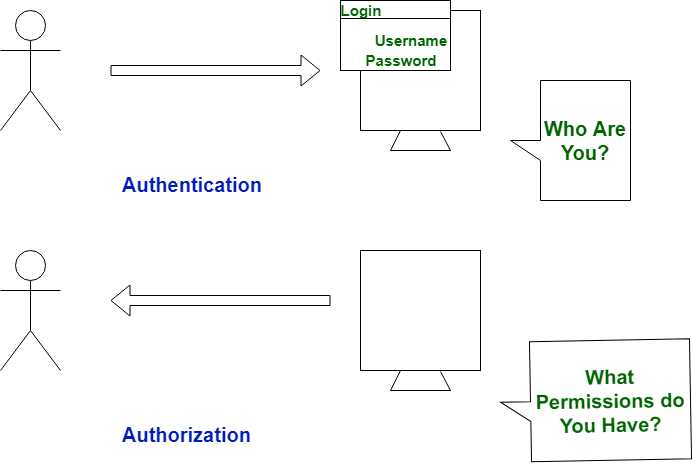
Фактор на притежание: Това включва „нещо, което потребителят има“, като физически токен или мобилен телефон, използван за получаване на кодове за удостоверяване.

Фактор на присъщност: Това се отнася до „нещо, което е потребителят“, което включва биометрични характеристики.



### **Съпоставка**

И двете единици са област за удостоверяване и оторизация и се използват по отношение на сигурността на знанията, която позволява безопасността на автоматична система за данни. Всяка регионална единица има ужасно важни теми, обикновено свързани с онлайн като ключови елементи от инфраструктурата на услугите. Въпреки това, всяка от термините площни единици е напълно различна с напълно различни идеи. като има предвид, че наистина те обикновено се използват в еквивалентен контекст с еквивалентен инструмент, те са напълно различни един от друг. В процеса на удостоверяване се проверява самоличността на потребителите за осигуряване на достъп до системата. Докато е в процеса на оторизация, се проверяват правата на лицето или потребителя за достъп до ресурсите. Удостоверяването се извършва преди процеса на оторизация, докато процесът на оторизация се извършва след процеса на удостоверяване.



## **HTML и основни тагове**

HTML таговете са ключови думи, оградени от ъглови скоби <>. Ако някога си имал профили по форуми, вече си попадал на нещо подобно – т.нар. BBCode, който е олекотен markup език и използва квадратни скоби []. А ако имаш собствен сайт или блог, вероятно вече дори си боравил с HTML тагове. Те са една от първите концепции за езика, с които ще се сблъскаш, започвайки да го изучаваш.

Таговете оформят т.нар. HTML елементи, които имат три съставки:

Отварящ таг – който индикира името на елемента, ограден от ъглови скоби, например <i> за текст в курсив.

Затварящ таг – който индикира къде завършва описваното с този елемент съдържание, а пред името на елемента има наклонена черта, като </b> за удебеляване.

Съдържание – това, което стои между отварящия и затварящия таг и трябва да се визуализира според елемента - <i>съдържание</i>, <b>друго съдържание</b>.

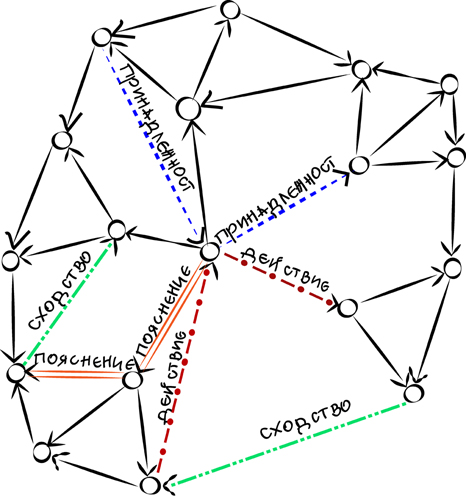


Отварящите и затварящите HTML тагове са ключови за правилното функциониране на HTML елементите. На първо място, ако забравиш да затвориш някой, може да се получи противоречие и вместо две думи, да bold-неш цял абзац. Едно от най-честите ти занимания, когато има грешки и дебъгваш HTML кода си, би било издирването именно на изпуснати или грешно изписани (най-често без наклонената черта) затварящи тагове. Освен това, те подсказват на браузъра къде започва и къде свършва даден тип съдържание.

Всички елементи могат да имат атрибути, които се поставят в отварящите HTML тагове. Например, ако на страницата се добавя картинка, тагът <img> ще бъде разширен с атрибута src или <img src=”your-image-address”>, който указва на браузъра къде да намери картинката, за да я визуализира.

## **Семантични страници**

Семантичен HTML не е друга версия на HTML. Той е най-обикновен HTML а семантичен го прави начина, по който той бива използван. Семантичен означава „значещ“. По принцип HTML не е нещо което е предназначено за крайния потребител пред екрана, а за разработчиците и машините(тръсещите машини като google и тези за незрящи хора). За да бъде „значещ“ за тях, той трябва да е много хубаво структуриран и да описва максимално точно съдържанието което представя. Той пояснява вида на съдържанието и указва неговата йерархия. Един сайт може да бъде изграден без новите семантични елементи и за обикновенният потребител това да не е проблем, но как ще се ориентира един незрящ човек или така наречените уеб паяци. За тях всичко ще е еднакво и те няма да разберат кое съдържание е по-важно от друго. Поради тези причини семантичния HTML е много важен за SEO оптимизацията и лесното разбиране на на сайтовете от хора със специални нужди.



### **Семантични HTML елементи**

Какво е HTML семантичен елемент?

Всеки от HTML семантичните елементи има свое специфично предназначение. Например тагът <h1> е семантичен елемент, който съобщава на ботовете на Google и други търсачки, че затвореното съдържание в маркера е най-значимото заглавие, съдържащо се в HTML документа или в съответната секция от сайта. Тагът <div>, от друга страна, е несемантичен елемент, тъй като той само указва разделяне в HTML документа и не дава информация на търсачаката, за значението и определянето на съдържанието в рамките на маркера.

### **Примерна структура на семантична HTML страница**

Структура на HTML документ

HTML документите са съставени от две основни части:

1. Глава (head)

2. Тяло (body)

Началният и крайният тагове на главата са <head> и </head>.

Началният и крайният тагове на тялото са <body> и </body>.

Типичната структура на един html документ е:

<html>

<head> </head>

<body> </body>

</html>

## **CSS, селектори и основни правила в CSS**

Какво е CSS и как се използва в HTML?

CSS (Cascading Style Sheets) е език за описване на презентацията и стиловете на елементите в един HTML/XML документ. CSS е една от основните технологии, използвани в уеб, редом с HTML и JavaScript.

При създаване на HTML страница, съдържанието й се описва (маркира) с HTML код, а презентацията на това съдържание, тоест как ще изглежда то в браузъра, се описва с CSS код (стил).

CSS кодът се поставя във файл с разширение .css. След това този файл може да се използва във всяка една страница от уеб сайта. Така че стилът ще се намира на едно място и при промяна ще се отразява на всички уеб страници.

Ще направим един най-прост CSS файл. Отворете някакъв текстов редактор, например Notepad и напишете в него следния код:

body

{

background-color: #000000;

font-family: arial, helvetica, sans-serif;

font-size: 16px;

color: #cccccc;

След това съхранете файла като му зададете разширение css – например file.css.

Сега остава във всяка HTML страница в секцията head да сложите следния код:

<link rel=stylesheet type="text/css" href="file.css" />

Пример:

<!DOCTYPE html>

<html>

<head>

<title>Заглавие на страницата</title>

<meta http-equiv=Content-Type content="text/html; charset=utf-8" />

<meta name="description" content="описание на страницата" />

<meta name="ключови, думи" />

<meta name="robots" content="index, follow" />

<link rel=stylesheet type="text/css" href="file.css" />

</head>

<body>

Това е примерен текст, намиращ се в тялото на HTML страницата.

</body>

</html>

Сега страницата ще се покаже със стиловите параметри, които сте задали, а именно – черен фон на страницата и светлосив текст. Всички страници, в които сложите кода <link rel=stylesheet type="text/css" href="file.css" /> ще имат същия външен вид. Ако желаете да промените цвета на фона или някои от параметрите на текста ще трябва да направите промените единствено в CSS файла и те автоматично ще се отразят във всички HTML документи, в които има връзка към файла file.css

За да работи този ефект, както е направен линка към file.css в примера, е нужно CSS файла да се намира в същата директория, в която се намират и HTML страниците. В противен случай трябва да се укаже пътя до CSS файла, например ако HTML страницата е във вътрешна папка, а file.css е в основната директория линка ще бъде

<link rel=stylesheet type="text/css" href="../file.css" />

## **Адаптивно (responsive) оформление на страници**

Какво е адаптивен уеб дизайн?

Адаптивен дизайн или наричан още responsive, респонсив, респонсивен, отзивчив е този дизайн, който притежава функцията да се адаптира автоматично към всички видове дисплеи за монитори, лаптопи, таблети, смартфони, айфони, айпади и др. всички с различна резолюция на екрана, без да се налага ръчно да се преминава към определена мобилна версия, а точно обратното, визуализацията да се осъществява автоматично, в зависимост от екрана на устройството.

Адаптивният дизайн включва специфично форматиране и структуриране на уеб сайта, така че автоматично да се скрие определена не толкова важна информация за сметка на по-важната такава, ако е наложително това, както и автоматично оразмеряване на изображенията, в зависимост от големината и резолюцията на дисплея, всичко това с цел, по-удобно и по-достъпно основно съдържание за потребителите с по-малките дисплеи на мобилни устройства.

Примерни framework-ове за responsive layout – Bootstrap – описание на framework-а, grid layout-a и основните класове

Какво е Bootstrap

Bootstrapе платформа (framework), която е комбинация от HTML, CSS и JavaScript код.

Bootstrap напълно поддържа CSS3 и HTML5 и вече се превръща в стандарт за уеб разработки.

Ползитеот него са:Bootstrap е съвместим с всички основни браузъри.Bootstrapсайтовете се адаптират много лесно за всякакви устройства. Използвайки комбинация от JavaScriptиCSS, Bootstrap позволява да се изпълни почти всеки елемент от дизайна, който е необходим за съответния проект.

Bootstrap е съвместим с всички основни браузъри.

Bootstrapсайтовете се адаптират много лесно за всякакви устройства.

Използвайки комбинация от JavaScriptиCSS, Bootstrap позволява да се изпълни почти всеки елемент от дизайна, който може да се изиска.

# Използвани технологии

# Технологичен стек и framework

# QuestSolve е изграден върху стабилен набор от технологии и frameworks, всяка от които допринася за цялостната функционалност и производителност на приложението

# C# и .NET Framework:

# C# (C Sharp): Разработен от Microsoft, C# е обектно-ориентиран език за програмиране. Той е част от .NET екосистемата и е известен със своята симплистичност, изразителност и страхотна производителност. C # е универсален и се използва в различни видове приложения, включително клиентски приложения на Windows, бекенд услуги, облачно базирани услуги и разработка на игри с Unity. Функции като събиране на боклука(garbage collection), обработка на изключения (exceptions) и безопасност на типа(strict typing) правят C# здрав и поддържан. Сходството му с други популярни езици като Java и C ++ улеснява ученето за тези, които са запознати с тези езици.

# .NET Framework: Това е framework за разработка на софтуер от Microsoft. Той осигурява контролирана среда за програмиране, където софтуерът може да бъде разработен, инсталиран и изпълнен на операционни системи, базирани на Windows. .NET Framework включва голяма библиотека от кодирани решения за предотвратяване на често срещани проблеми с програмирането и виртуална машина, която управлява изпълнението на програми, написани специално за frameworka. .NET Framework е създаден, за да улесни разработването на уеб приложения и услуги, с функции като управление на паметта, функции за сигурност и оперативна съвместимост с различни езици. Той се използва широко както за уеб, така и за настолни приложения.

# Razor Views:

# Razor е синтаксис за програмиране на ASP.NET, използван за създаване на динамични уеб страници със C # (или VB.NET). Това не е самият език за програмиране, а по-скоро начин да се напише C # или VB.NET код в HTML страниците. Razor намалява броя на знаците необходими във файла, и позволява бърз, плавен работен поток за кодиране, позволявайки на изгледа (front-end) лесно да визуализира данните от базата с данни.

# Как работи: Във файл на Razor (който обикновено има разширение на файла .cshtml или .vbhtml) се пишат HTML тагове, както обикновено, но може също да се вгради код от страна на сървъра, като използвате синтаксиса на Razor, който е лесен и ясен, често започващ със знак "@", последван от C # или VB.NET код. Това улеснява обединяването на HTML и логиката от страна на сървъра.

# Предимства: Основното предимство на Razor Views е безпроблемната интеграция на кода от страна на сървъра с HTML, което позволява динамично генериране на уеб съдържание. Той улеснява процеса на писане на код, който взаимодейства с HTML елементи и помага за ефективното създаване на динамични, задвижвани от данни уебсайтове.

# Entity Framework (Object Framework):

# Entity Framework (EF) е open-source object-relational mapping (ORM) framework за ADO.NET, част от .NET Framework. Това са набор от технологии, които поддържат разработването на софтуерни приложения, ориентирани към данните съхранявани на сървара. Предоставяйки възможност на софтуерните инженери да работят на по-високо ниво на абстракция, когато се занимават с данни и могат да се съсредоточат върху бизнес логиката, а не върху спецификата на достъпа до данни.

# Как работи: EF позволява на разработчиците да работят с данни под формата на специфични за домейна обекти и свойства, без да е необходимо да се занимават с основните таблици и колони на базата данни, където се съхраняват тези данни. Той автоматично генерира заявките към базата данни и ги превежда в SQL за изпълнение срещу базата данни. EF поддържа LINQ (Language Integrated Query), което позволява на софтуерните инженери да пишат заявки към базата данни, използвайки синтаксиса на C#.

# Предимства: Основната полза от използването на Entity Framework е, че автоматизира операциите, свързани с базата данни, и намалява количеството код, който разработчиците трябва да пишат за достъп до данните на сървара. Той улеснява процеса на създаване, актуализиране, изтриване и извличане на данни от базата данни. Той също така помага при управлението на промените в схемата и миграциите на базата данни по-гладко с функции като Code-First Migrations.

# Структура на приложението

## **База данни ER диаграма**

База данни и ORM

MySQL е популярна система за управление на релационни бази данни (RDBMS), която използва SQL (Structured Query Language) за управление на данни. Тя е особено подходяща за използване в проекти като QuestSolve поради своите предимства, които включват следното:

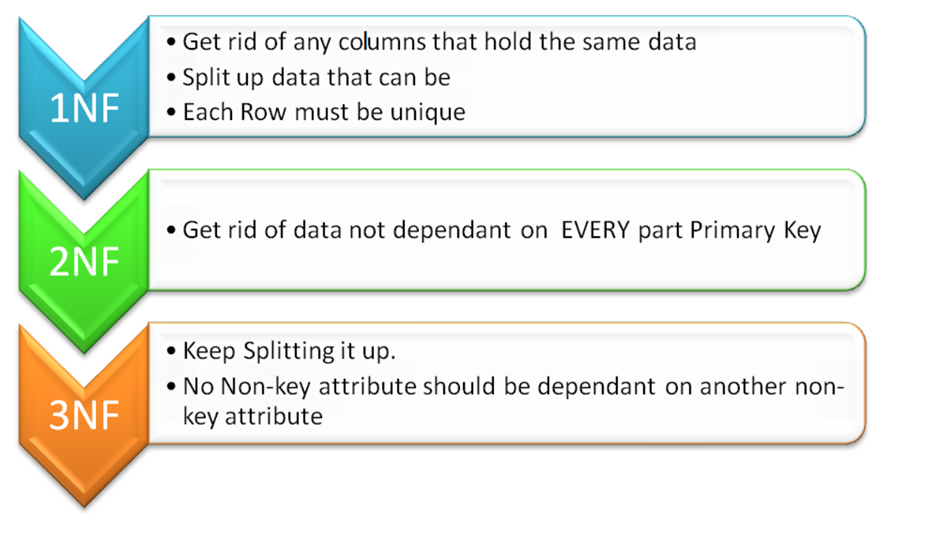
1. **Отворен код и богата общност**: MySQL е open-source проект, което означава, че може да бъде използван и модифициран от всеки безплатно. Богатата общност около MySQL означава, че има много ресурси за поддръжка и усъвършенстване.
2. **Надеждност и сигурност**: MySQL е известна със своята надеждност и предоставя много функции за сигурност, които помагат за защита на данните от неоторизиран достъп или злоупотреби.
3. **Скалируемост и гъвкавост**: MySQL е много гъвкава и лесно скалируема система за управление на бази данни, което я прави идеална за проекти, вариращи от малки приложения до големи корпоративни системи.
4. **Широка поддръжка на платформи**: MySQL поддържа широк спектър от операционни системи, което я прави удобна за различни среди на разработка.
5. **Оптимизация за производителност**: MySQL предлага отлична производителност при четене и писане на данни, което е особено важно за приложения като QuestSolve, които обработват голям обем от транзакции.
6. **Поддържане на стандарти и съвместимост**: MySQL е в съответствие със стандартите на SQL и лесно се интегрира с различни програмни езици и технологии, което я прави подходяща за интеграция в разнообразни проекти.
7. **Лесен за използване и поддръжка**: MySQL има лесна и интуитивна архитектура, което прави нейното изучаване и управление относително лесно за разработчиците.

Тези характеристики правят MySQL отличен избор за базата данни на QuestSolve, осигурявайки стабилна, сигурна и гъвкава среда за управление на данните, които са жизненоважни за функционалността на приложението.

**Проектиране и схема на база данни**

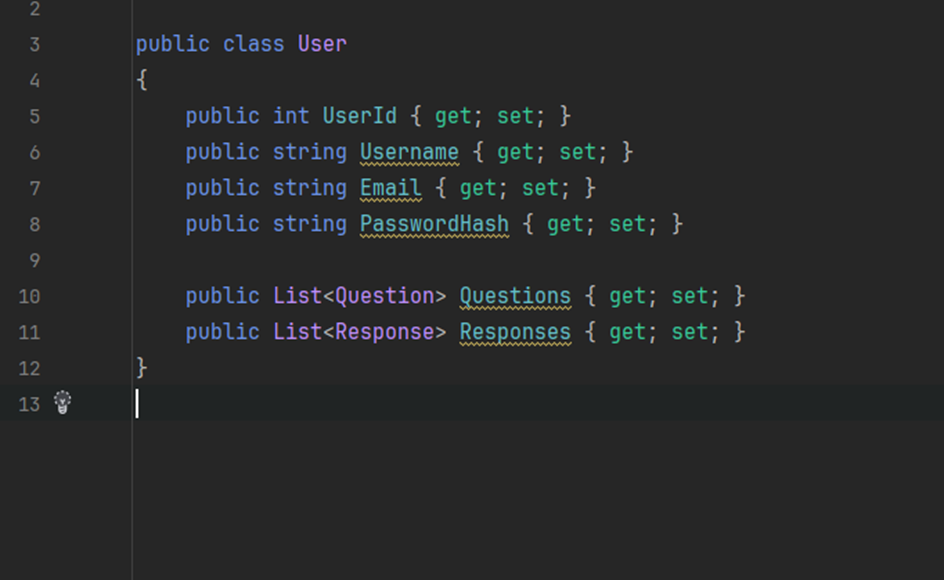
Базата данни на QuestSolve е предназначена за ефективно съхранение и управление на данни, свързани с потребителите, въпроси, отговори и други функционалности на приложението. Добре структурираната схема е от решаващо значение за поддържане на целостта на данните, оптимизиране на производителността и осигуряване на скалиране.

* **Нормализиране**: Схемата на базата данни е преминала през процес на нормализиране, за да се намали излишъкът на данни и да се подобри целостта на данните. Това включва организиране на данни в таблици и колони по начин, по който всяка таблица представлява една концепция или обект, като например потребител, въпрос или отговор.

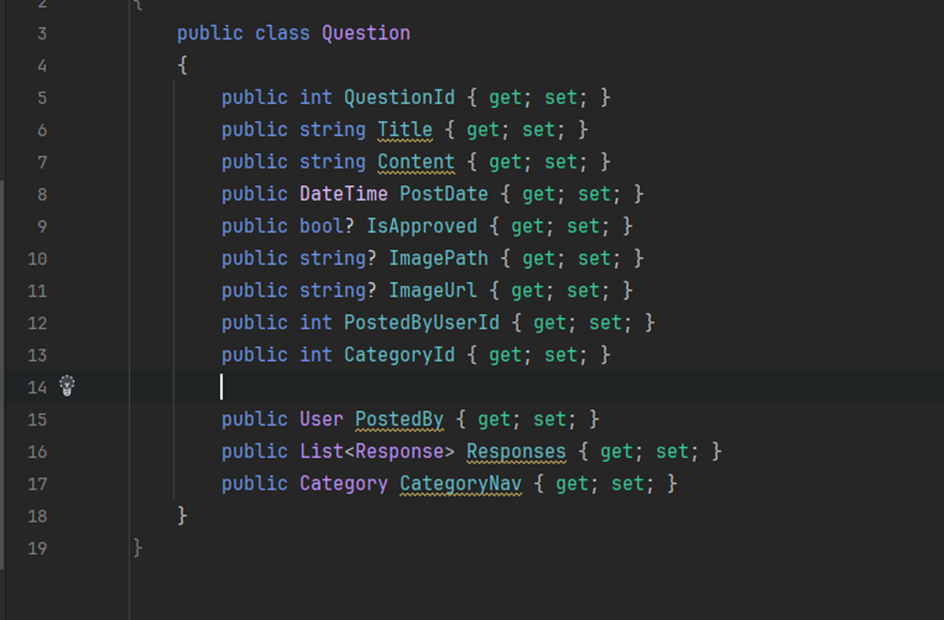


## **Описание на таблиците в базата**

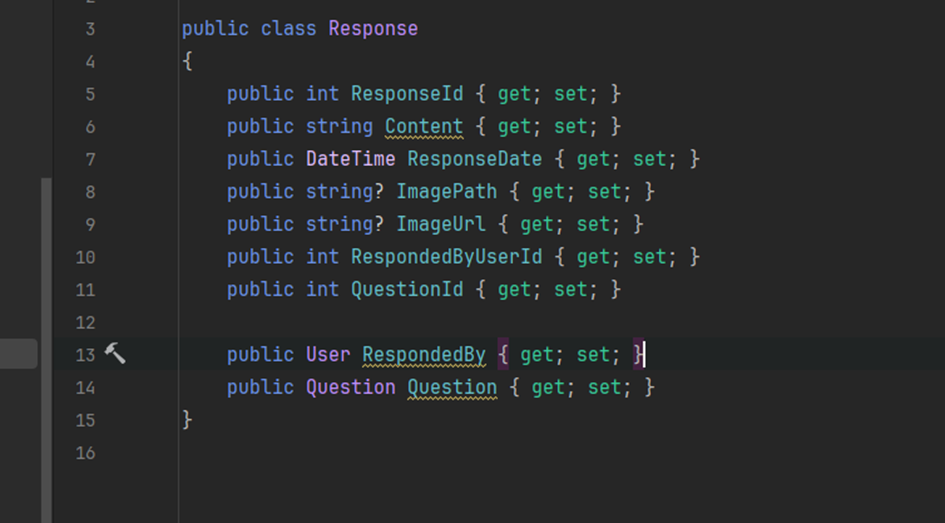
* **Таблици и релации**: Основните таблици включват потребители, въпроси и отговори.
  + Таблицата "Потребители " съхранява потребителска информация.



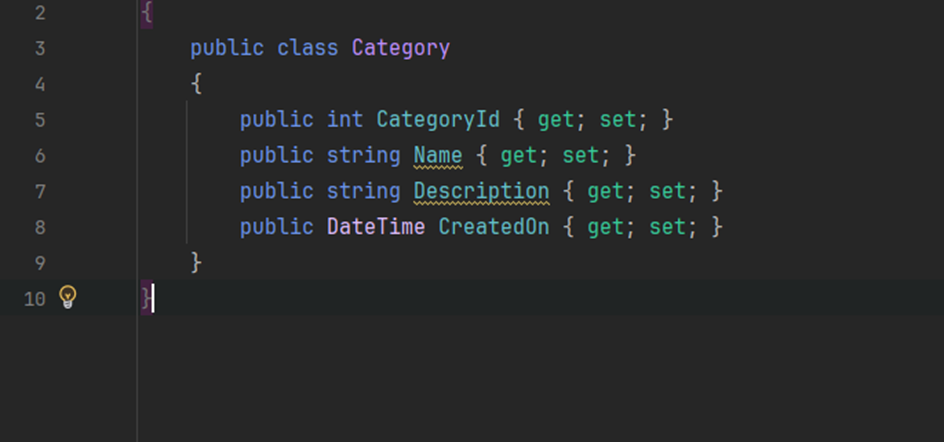
* Таблицата "Въпроси" съдържа подробности за въпросите, публикувани от потребителите, включително препратки към потребителите, които са ги публикували.



* Таблицата "Отговори" съдържа отговорите, предоставени от потребителите на различни въпроси, свързани както с таблиците "Въпроси", така и с "Потребители".



* Таблицата “Категории“ съдържа категориите техният първичен ключ се използва в таблицата за въпросите, предоставяйки възможността всеки въпрос да има категория.

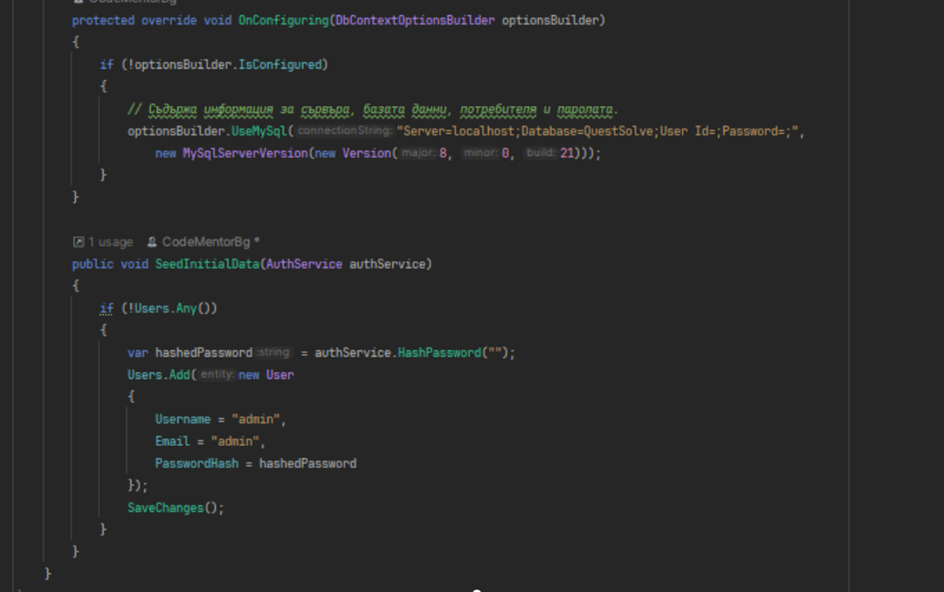


 Индекси и ключове: Подходящи първични ключове (като UserId, QuestionId, ResponseId) се използват за еднозначно идентифициране на записи във всяка таблица. Външните ключове установяват релации между таблиците, като например свързване на отговори към съответните им въпроси и потребители.

QuestSolve използва Entity Framework, мощен ORM (Object-Relational Mapping) инструмент в .NET, за да взаимодейства с базата данни.

А. Функционалност: Entity Framework позволява на приложението да работи с обекти на бази данни, използвайки C# класове, абстрахирайки базовия SQL. Това улеснява операциите с бази данни като CRUD (Създаване, Четене, Актуализиране, Изтриване) действия.

Б. Конфигурация и използване: В QuestSolve, Entity Framework е конфигуриран да съпоставя моделите на C # (потребител, въпрос, отговор) в съответните таблици на базата данни. Контекстният клас (ForumDbContext) управлява обектите на обекта по време на изпълнение, обработвайки връзката с базата данни и заявките.



В. LINQ заявки: QuestSolve използва LINQ (езикова интегрирана заявка) за заявка към базата данни. LINQ заявките са написани на C#, предлагайки четливост и безопасност на типа. Например, HomeController използва LINQ за филтриране и страниране на въпроси въз основа на потребителски вход.

Г. Асинхронни операции: Приложението използва асинхронни операции с бази данни (напр. ToListAsync (), SaveChangesAsync ()), подобрявайки производителността и мащабируемостта, особено при високо натоварване

**Четене на данните**

Четенето на данните в QuestSolve включва преобразуване на записите в базата данни в смислена информация:

**** Извличане и трансформация на данни: При извличане на данни, Entity Framework преобразува SQL данни в C # обекти, които след това се използват за генериране на отговори на клиентски заявки. Например, подробностите за въпросите се извличат от базата данни и се представят в удобен за потребителя формат в интерфейса на приложението.

 Управление на връзките: Entity Framework управлява взаимоотношенията между обектите, като например релацията "един към много" между потребител и въпроси. Това е нужно при операции, при които извличането на въпрос включва и извличане на потребителя, който го е публикувал.

## **Структура на кода**

**Видове релации и ключове в базата данни**

**Преглед**

Разбирането на типовете взаимоотношения и използването на ключове в базата данни е от съществено значение за разбирането как данните са взаимосвързани в рамките на QuestSolve. Този раздел изследва различните типове взаимоотношения, присъстващи в базата данни на приложението, и ролята на първичните и външните ключове за поддържане на целостта на данните и улесняване на ефективното извличане на данни.

**Видове релации**

 **Релации "един към един":**

o Въпреки че не е изрично демонстрирано в предоставения код, релация "един към един" може да съществува, когато един запис в една таблица е свързан само с един запис в друга таблица. Например, ако имаше таблица за потребителски профили, всеки потребител (таблица "Потребител") може да бъде свързан с точно един профил в таблицата "Потребителски профил".

 **Релации "един към много":**

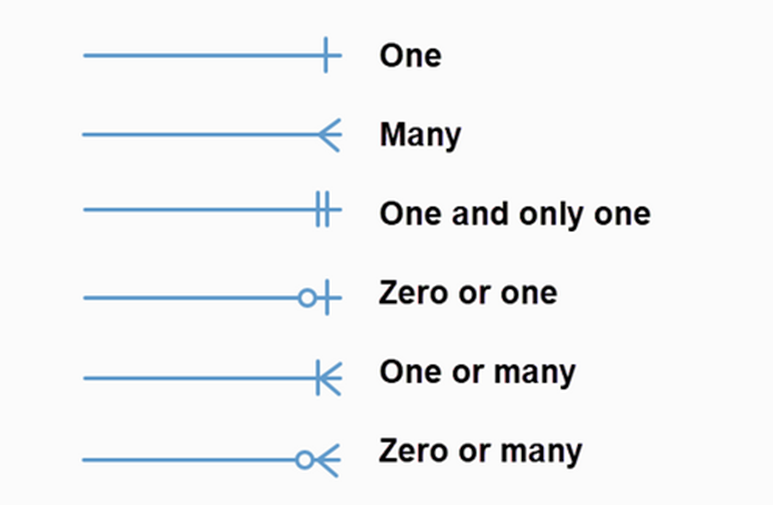
o Това е преобладаваща връзка в QuestSolve. Потребителят може да публикува множество въпроси и отговори, но всеки въпрос и отговор са свързани само с един потребител.

o Пример: Потребителската таблица има релация "един към много" с таблиците "Въпроси и отговори". Всеки въпрос и отговор има чужд ключ, сочещ към съответния потребител.

 **Релации "много към много":**

o Тези релации могат да съществуват в сценарии, при които обектите имат множество асоциации. В QuestSolve практически пример може да бъде, ако въпросите могат да имат няколко категории и всяка категория може да включва множество въпроси.

o Изпълнение: Това обикновено се управлява чрез таблица за присъединяване, като например QuestionCategories, свързване на таблици с въпроси и категории **.**

****

**Ключове в базата данни**

**Първични ключове:**

o Първичните ключове еднозначно идентифицират всеки запис в таблица на база данни.

o Пример: В QuestSolve, UserId в таблицата Потребител, QuestionId в таблицата Въпрос и ResponseId в таблицата Отговор служат като първични ключове.

**Външни ключове:**

o Външните ключове се използват за свързване на таблици заедно и за поддържане на целостта на връзките.

o Пример: В таблицата "Въпроси" PostedByUserId е външен ключ, който препраща към UserId в таблицата Потребител, установявайки връзка между потребител и неговите въпроси.

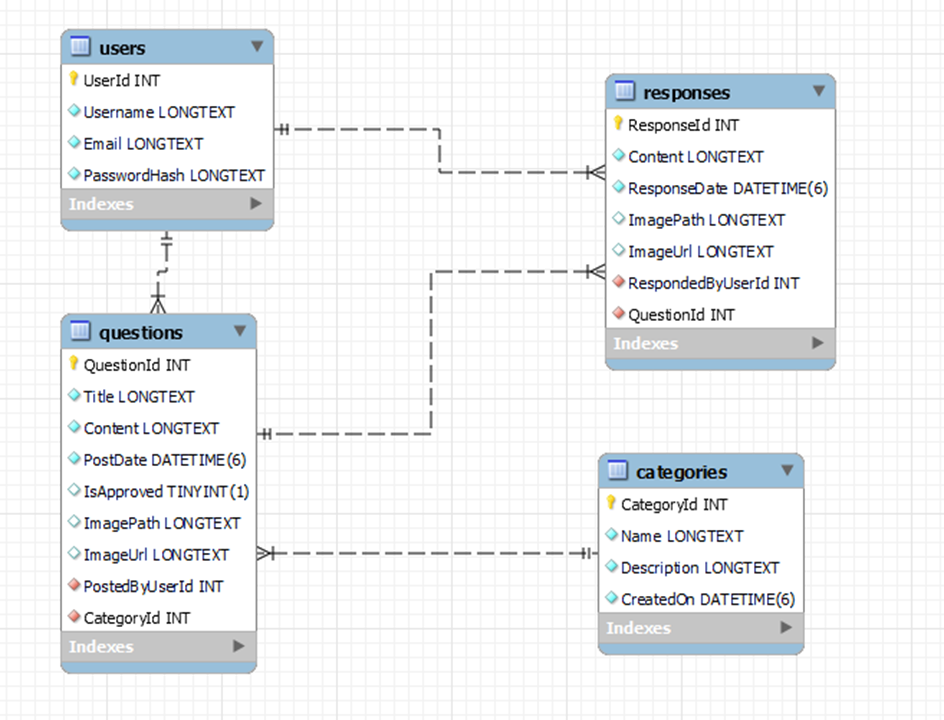
Тези връзки и ключове са от решаващо значение за ефективната организация и извличане на данни. Например, когато показва въпрос заедно с неговите отговори, приложението може бързо да извлече свързаните данни благодарение на тези добре дефинирани взаимоотношения.

В Entity Framework тези релации се отразяват в моделите и се използват за ефективно навигиране между обектите.

**Осигуряване на целостта на данните**

• Използването на външни ключове гарантира, че релациите между различните обекти са последователни. Например, изтриването на потребител може да бъде настроено на каскадно изтриване на всички негови въпроси и отговори, запазвайки целостта на базата данни.

Това е схема на база данни, която представя взаимовръзката между четири основни таблици, които могат да се използват в уеб приложение:



**Таблиците и техните атрибути включват:**

 **Таблица users**: Съдържа информация за потребителите.

 **UserId**: Уникален идентификатор за всеки потребител.

 **Username**: Потребителско име.

 **Email**: Електронна поща на потребителя.

 **PasswordHash**: Хеширана парола на потребителя.

 **Таблица questions**: Включва въпросите, публикувани в приложението.

 **QuestionId**: Уникален идентификатор за въпроса.

 **Title**: Заглавие на въпроса.

 **Content:** Съдържание на въпроса.

 **PostDate:** Дата и време на публикуване.

 **IsApproved:** Статус на одобрение.

 **ImagePath:** Път към свързаната с въпроса изображение.

 **ImageURL**: URL на изображението.

 **PostedByUserId**: Идентификатор на потребителя, който е публикувал въпроса.

 **CategoryId**: Идентификатор на категорията, към която принадлежи въпроса

 **Таблица responses**: Съдържа отговорите на въпросите.

 **ResponseId**: Уникален идентификатор на отговора.

 **Content**: Съдържание на отговора.

 **ResponseDate**: Дата и време на публикуване на отговора.

 **ImagePath**: Път към свързаната с отговора изображение.

 **ImageURL**: URL на изображението.

 **RespondedByUserId**: Идентификатор на потребителя, който е отговорил.

 **QuestionId:** Идентификатор на въпроса, за който е отговора.

 **Таблица categories**: Дефинира категориите за въпросите.

 **CategoryId:** Уникален идентификатор на категорията.

 **Name**: Име на категорията.

 **Description**: Описание на категорията.

 **CreatedOn**: Дата и време на създаване на категорията.

**Code First и Database First подходи в QuestSolve**

**Въведение в подходите за развитие**

В сферата на ORM и Entity Framework съществуват два основни подхода за развитие: Code First и Database First. Изборът на QuestSolve между тези подходи влияе върху начина, по който се създават и управляват схемата на базата данни и моделите на приложения.

Code First подход в QuestSolve

 Определение и употреба:

o В подхода Code First, схемата на базата данни се генерира от класовете на модела C#. Това означава, че разработчиците дефинират обекти, релации и логика на базата данни директно в кода, а рамката създава съответните таблици и релации в базата данни.

o Използването на Code First от QuestSolve е очевидно в акцента върху класовете на C # за определяне на структурата на данните и взаимоотношенията (например - Потребител, Въпрос, Отговор).

 **Предимства:**

o **Гъвкавост и контрол**: Разработчиците имат пълен контрол върху схемата на базата данни чрез код, което е особено полезно по време на ранните етапи на развитие.

o **Гъвкавост**: Поддържа гъвкава среда за разработка, където промените в изискванията могат бързо да бъдат внедрени в кода и схемата на базата данни се актуализира съответно.

 **Миграции**:

o Code First Migrations позволяват постепенно развитие на схемата на базата данни. Промените в класовете на модела могат да бъдат преведени в миграции на бази данни, които променят схемата, без да губят съществуващи данни.

Database First подход

 **Контраст с кода на първо място**:

o Подходът Database First започва със съществуваща база данни, от която ORM рамката генерира класовете модели. Той е по-подходящ за сценарии, при които приложението е изградено около предварително дефинирана схема на базата данни.

 **Релевантност към QuestSolve**:

-Докато QuestSolve използва предимно подхода Code First, разбирането на подхода Database First е от решаващо значение, особено ако приложението трябва да се интегрира със съществуваща система за бази данни в бъдеще.

**Обосновка на избрания подход**

• QuestSolve се възползва от подхода Code First поради своята гъвкавост, лекота на управление на промените в базата данни и възможността да поддържа схемата на базата данни в синхрон с развиващите се изисквания за данни на приложението.

• Подходът се привежда в съответствие с гъвкавата методология за разработка, улеснявайки бързите цикли на разработка, лесното създаване на версии и сътрудничеството.

Преход между подходите

• Докато QuestSolve в момента използва Code First, от съществено значение е да се проектира системата по начин, който не изключва преминаването към подход Database First, ако е необходимо, особено за мащабируемост и интеграция с наследени системи**.**

**Удостоверяване и упълномощаване в QuestSolve**

**Механизъм за удостоверяване, базирано на бисквитки в QuestSolve:**

1.Уникалност на всяка бисквитка: Когато потребител влезе успешно в системата, QuestSolve генерира уникална бисквитка за тази сесия. Тази бисквитка е специфична за всяка потребителска сесия, което означава, че тя съдържа уникален идентификатор, който не може да бъде лесно подправен или използван от други потребители.

2.Съхранение на потребителска информация: В бисквитката се съхранява информация за потребителя, като потребителско име, идентификатор на потребителя (ID), и ролите, които той заема в системата. Тази информация се използва за идентифициране на потребителя при всяка негова заявка към сървъра и за осигуряване на съответните права за достъп и функционалности в приложението.

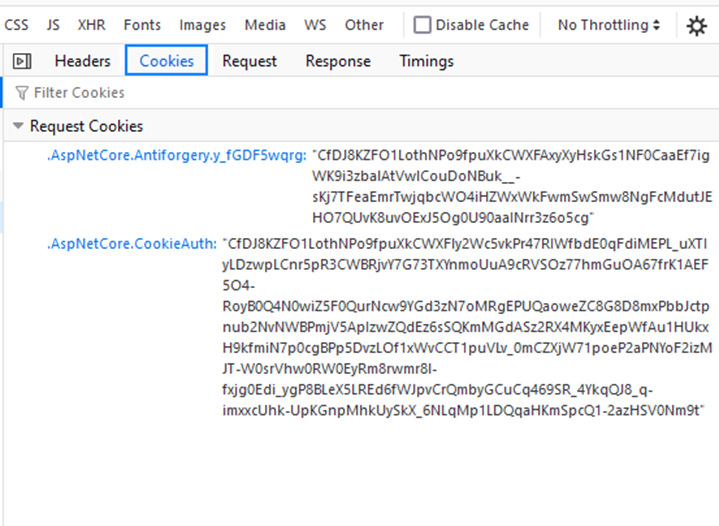
3.Сериализация на идентичността, базирана на искове: Когато се създава идентичността на потребителя, тя се базира на искове (claims-based identity). Това означава, че потребителската идентичност включва редица твърдения (claims) за потребителя, като име, роли, права и други атрибути. Тази идентичност се сериализира и съхранява в бисквитката.

4.Удостоверяване на последващи заявки: Всяка път когато потребителят изпрати заявка към сървъра, бисквитката се изпраща автоматично. Сървърът проверява бисквитката за удостоверяване на самоличността на потребителя и определяне на правата му в системата.

5.Използване на бисквитката за персонализиране: Тъй като бисквитката съдържа потребителска информация като ID и име, тя може да се използва за персонализиране на потребителския интерфейс. Например, когато потребителят посети определена страница в QuestSolve, информацията от бисквитката може да се използва за показване на персонализирани съобщения или функции, специфични за ролята или правата на потребителя.

6.Сигурност и конфиденциалност: QuestSolve трябва да осигури, че бисквитките са сигурни и защитени от неоторизиран достъп или манипулации. Това обикновено се постига чрез използване на мерки за сигурност като HTTPS, шифроване на бисквитки и стратегии за ограничаване на валидността на бисквитките.

На снимката по-долу се показват детайли за бисквитки, изпратени в заявка от уеб браузъра към сървъра. Тези бисквитки са свързани с ASP.NET Core уеб приложение, което показва:



**1. ASP.NET Core AntiForgery Token**: Токен, използван за предотвратяване на CSRF (Cross-Site Request Forgery) атаки. Той гарантира, че подадените форми и заявки са оригинални и идват от съществуващата сесия на потребителя.

**2. ASP.NET Core Cookie Auth Token**: Бисквитка за удостоверяване, съдържаща информация за сесията на потребителя. Тя се използва за удостоверяване на потребителите при всяка заявка и поддържане на състоянието на сесията.

Тези бисквитки са критични за управлението на сесията и сигурността на уеб приложението, като помагат за идентифициране на валидните потребителски заявки и защитават от злонамерени действия.

В заключение, механизмът за удостоверяване, базиран на бисквитки, е ключов за сигурността и ефективността на QuestSolve. Той осигурява, че само удостоверени потребители имат достъп до специфични функции и данни, като същевременно предоставя гъвкавост и удобство при използване на платформата.

**А. Процес на Влизане в QuestSolve:**

1.Проверка на идентификационните данни:

Когато потребителят опитва да влезе в системата, той изпраща своите идентификационни данни (като потребителско име и парола) чрез интерфейса на приложението. Тези данни се обработват от UserController, който ги проверява спрямо съхранените данни в базата данни на QuestSolve. Процесът на проверка включва сравняване на въведените данни с информацията, която е съхранена в системата, за да се установи дали потребителското име и паролата съвпадат с някой съществуващ акаунт.

2.Генериране и изпращане на бисквитка за удостоверяване:

Ако идентификационните данни са проверени успешно, системата генерира бисквитка за удостоверяване. Тази бисквитка съдържа информация, която маркира потребителя като удостоверен, и се изпраща обратно към браузъра на потребителя. След като браузърът получи бисквитката, тя се съхранява и използва за последващо удостоверяване на заявките на потребителя към сървъра.

**Б. Процес на Излизане от QuestSolve:**

1. Анулиране на бисквитката за удостоверяване:

Когато потребителят иска да излезе от системата, се инициира процесът на излизане. Този процес включва анулиране или изтриване на бисквитката за удостоверяване от браузъра на потребителя. Анулирането на бисквитката гарантира, че няма повече валидно удостоверяване на потребителя, и той не може да извършва действия, които изискват удостоверен достъп, без да се впише отново.

**Заключение:**

В процеса на влизане и излизане, QuestSolve използва бисквитки за удостоверяване като ключов компонент за идентифициране и поддържане на състоянието на удостоверените потребители. Този подход осигурява безопасност и удобство при управлението на потребителските сесии и позволява ефективна и безопасна работа с приложението.

**Оторизиране**

1.Ролево базиран контрол на достъпа:

-QuestSolve реализира ролево базиран контрол на достъпа, за да управлява потребителските разрешения. Например, административните функционалности като одобряване или неодобряване на публикации са ограничени до потребители с администраторска роля.

-Това се управлява чрез проверка на потребителските твърдения и роли в бисквитката за удостоверяване.

**Съображения за сигурност**

1.Сигурност на бисквитките:

-Защитените атрибути и флаговете само за HTTP са настроени за бисквитки, за да се предотврати неоторизиран достъп и да се намали рискът от XSS (Cross-Site Scripting) атаки.

-SSL / TLS криптирането се използва за осигуряване на предаване на данни, предотвратявайки прихващането на "бисквитки".

2.Проверка и обработка на грешки:

-Приложението включва стабилно валидиране и обработка на грешки по време на процеса на удостоверяване, за да се предотвратят нарушения на сигурността, като SQL инжектиране или пълнене на идентификационни данни.

3.Управление на пароли:

-Паролите се хешират и съхраняват сигурно, както се вижда в класа AuthService. Това гарантира, че действителните пароли не се съхраняват в базата данни, което повишава сигурността.

4.SHA-256

SHA-256 е функция за хеширане от семейството SHA-2 (Secure Hash Algorithm 2), разработена от Националния институт по стандартите и технологиите на САЩ (NIST). Тя играе критична роля в защитата на информацията и управлението на паролите в системи като QuestSolve. Ето как работи и защо е толкова важна:

-Криптографска хеш функция: SHA-256 преобразува входни данни (каквито и да са те по размер и тип) в уникален низ от битове с фиксирана дължина (256 бита, или 64 символа в хексадецимален формат). Това хеш значение е по същество "отпечатък" на входните данни.

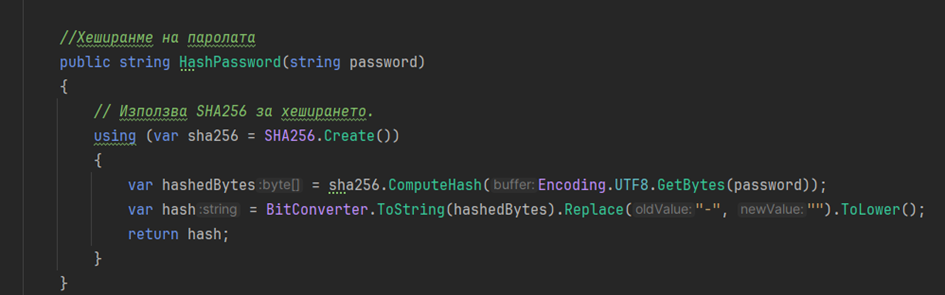
-Устойчивост на колизии: Една от ключовите характеристики на SHA-256 е, че тя е устойчива на колизии, което означава, че е изключително трудно (практически невъзможно със съвременните технологии) два различни входни набора от данни да произведат едно и също хеш значение.

-Необратимост: Хеш функциите като SHA-256 са необратими. Това означава, че не можете лесно да възстановите оригиналните входни данни от хеш стойността. Тази характеристика е особено важна за сигурността на паролите, тъй като дори ако хешът е изложен, реалната парола остава защитена.

-Приложение в управлението на пароли: В QuestSolve, паролите на потребителите се хешират с помощта на SHA-256 преди да бъдат съхранени в базата данни. Това означава, че дори ако базата данни бъде компрометирана, паролите ще останат в хеширана форма, което прави извличането на оригиналните пароли изключително трудно.

- Използване в криптографията: SHA-256 се използва не само за хеширане на пароли, но и в различни криптографски приложения, включително в дигитални подписи и блокчейн технологии.

-Използването на SHA-256 в QuestSolve допринася значително за цялостната сигурност на приложението, осигурявайки защита на чувствителните данни на потребителите и подобрявайки целостта и надеждността на системата.



HTML и CSS Основи в QuestSolve

Преглед

В QuestSolve, HTML (HyperText Markup Language) и CSS (Cascading Style Sheets) играят основна роля в структурирането и оформянето на потребителския интерфейс на уеб приложението. Те са от решаващо значение за създаването на ангажиращо и интуитивно потребителско изживяване.

**Структура и използване на HTML**

-HTML роля: HTML осигурява основната структура на уеб страниците. В QuestSolve се използва за определяне на оформлението и съдържанието на уеб страниците, като текст, изображения, връзки, формуляри и други елементи.

-Razor Views: Синтаксисът на Razor се използва за динамично генериране на HTML съдържание въз основа на данните на приложението. Например, показването на въпроси и отговори на началната страница се контролира от шаблони Razor View, които рендират HTML въз основа на данните на модела.

-Формуляри и въвеждане от потребителя: HTML формулярите се използват широко за потребителски взаимодействия като регистрация, вход, публикуване на въпроси и отговори. Тагът <формуляр> заедно с различни типове входни данни като <вход>, <текстова област> и <изберете>, улеснява тези функционалности.

**CSS стилизиране**

-CSS интеграция: CSS се използва за подобряване на външния вид на HTML съдържание. В QuestSolve, CSS дефинира стилове за оформление, цветове, шрифтове и отзивчиви дизайнерски елементи, допринасяйки за цялостната естетика и използваемост на приложението.

-Bootstrap Framework: QuestSolve включва Bootstrap, популярна CSS рамка, за да рационализира стайлинга и да осигури последователност в приложението. Мрежовата система на Bootstrap, предварително дефинираните класове и помощните програми за отзивчив дизайн помагат за създаването на визуално привлекателен и удобен за мобилни устройства интерфейс.

-Персонализирани стилове: Наред с Bootstrap, персонализираните CSS стилове могат да бъдат дефинирани за по-нататъшно приспособяване на външния вид и усещането на приложението. Тези персонализирани стилове позволяват специфични корекции и подобрения на дизайна извън предложенията по подразбиране на Bootstrap.

**Основни тагове и селектори**

•Тагове: Общите HTML тагове, използвани в QuestSolve, включват <div>, <h1> до <h6>, <p>, <a>, <img> и маркери, свързани с формата, като <input>, <label>, <button>.

•CSS селектори: CSS селекторите се използват за прилагане на стилове към HTML елементи. Те включват селектори на типове (напр. div, h1), селектори на класове (напр. .navbar, .btn) и селектори на ID (напр. #header).

**Динамичен дизайн в QuestSolve (responsive design)**

Преглед на responsive дизайн

Динамичният дизайн е критичен аспект на съвременната уеб разработка, като гарантира, че уеб приложението осигурява оптимално изживяване при използване на редица устройства, от настолни монитори до мобилни телефони. В QuestSolve се реализира динамичен дизайн посредством използване на Bootstrap, за да се осигури използваемост и достъпност, независимо от устройството на потребителя.

Техники и изпълнение

Система за екранно разделяне на Bootstrap:

-QuestSolve използва мрежовата система на Bootstrap. Тази система разделя екрана на мрежа от редове и колони, което позволява съдържанието да бъде поставено и подравненo. Мрежата се настройва автоматично на различни устройства, осигурявайки последователно оформление на различните платформи в зависимост от размерите на екрана.

**Мета етикет на Viewport:**

-Мета маркерът viewport, който се намира в HTML заглавката, е от решаващо значение за отзивчивия дизайн. Той контролира размерите на страницата и мащабирането на различни устройства. Например, <meta name="viewport" content="width=device-width, initial-scale=1.0"> гарантира, че оформлението се регулира според ширината на устройството.

## **Клас диаграми**

**UML и ER диаграми**

**Въведение**

За цялостно разбиране на архитектурата и схемата на базата данни на QuestSolve, UML (Unified Modeling Language) и Entity-Relationship (E / R) диаграмите са основни инструменти. Те визуално представят компонентите на системата, техните взаимовръзки и взаимодействия, осигурявайки ясен преглед на структурата на приложението.

**UML диаграми**

1.**Регистрация**: Този случай на използване позволява на новите потребители да създадат акаунт в QuestSolve. Включва предоставяне на лична информация и настройване на потребителско име и парола.

2.**Вход**: Съществуващите потребители могат да получат достъп до профила си, като въведат своите идентификационни данни. Тази стъпка е необходима за извършване на повечето действия на платформата.

3.**Преглед на въпроси**: Потребителите могат да преглеждат списъка с въпроси, които са публикувани в QuestSolve. Това може да включва различни категории и теми.

4.**Търсене на въпрос**: Ако потребителите търсят конкретна информация, те могат да използват функцията за търсене, за да намерят въпроси, които отговарят на тяхната заявка.

5.**Филтриране по категория**: За да се рационализира процесът на сърфиране, потребителите могат да филтрират въпроси въз основа на различни категории или теми, което улеснява намирането на подходяща информация.

6.**Задаване на въпрос**: Потребителите могат да публикуват свои собствени въпроси на платформата. Това обикновено включва писане на въпроса, добавяне на подробности и избор на подходяща категория.

7.**Актуализиране на зададени въпроси**: Ако потребител иска да промени въпрос, който преди това е задал – например за изясняване на заявката или добавяне на допълнителна информация – той може да използва тази функция.

8.**Напиши коментар**: Потребителите могат да се участват в дискусия, като пишат коментари към различни публикации. Това е начин за предоставяне на отговори или допълнителни запитвания.

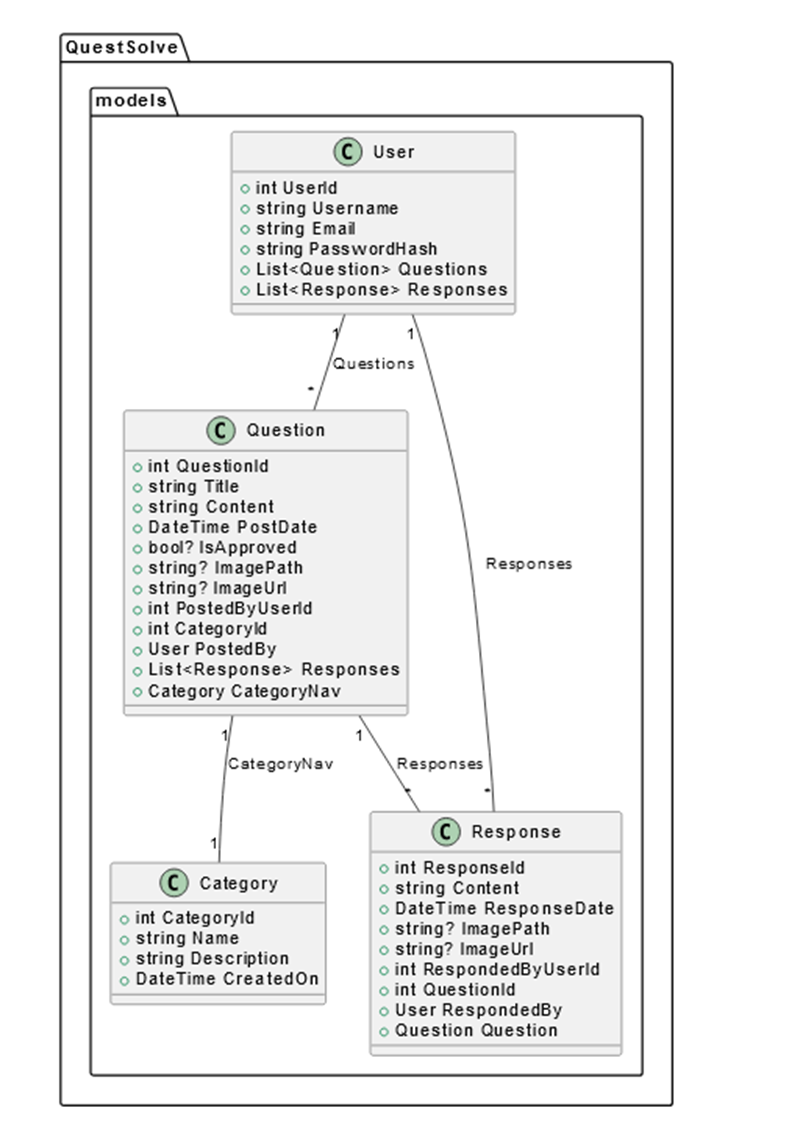
9.**Редактиране на коментар**: Ако потребител иска да промени или коригира нещо в коментара си, тази функция му позволява да редактира предварително изпратените си коментари.

10.**Изтриване на коментар**: Потребителите могат да премахнат коментарите си от публикация, ако вече не желаят те да бъдат видими за другите.

11.**Изтриване на публикация**: Подобно на изтриването на коментар, тази функция позволява на потребителите да премахват своите въпроси или публикации от платформата.

12.**Излизане**: Когато потребителите приключат с използването на платформата, те могат да излязат от профила си, като гарантират, че сесията им е сигурно затворена.

**Диаграми на релации между обекти**

****

1.Потребител към въпрос: Един към много (1 потребител може да публикува няколко въпроса)

2.Потребител към отговор: Един към много (1 потребител може да направи няколко отговора)

3.Въпрос към отговор: Един към много (1 въпрос може да има няколко отговора)

4.Въпрос към категория: Много към един (Много въпроси могат да принадлежат към 1 категория)

5.Категория към въпрос: Един към много (1 Категория може да има няколко въпроса)

**Дизайн и Взаимодействие**

• UML и E / R диаграмите колективно предоставят план на архитектурата и схемата на базата данни на QuestSolve.

• Те помагат да се разбере как различните части на приложението взаимодействат и как данните преминават през системата.

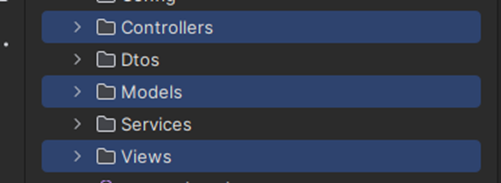
Използване при разработка и поддръжка

• Тези диаграми са не само от решаващо значение по време на първоначалните фази на проектиране, но също така служат като ценни референции по време на поддръжката и при внедряването на нови функции или извършването на модификации.

**Разделяне на компоненти: MVC принципи в QuestSolve**

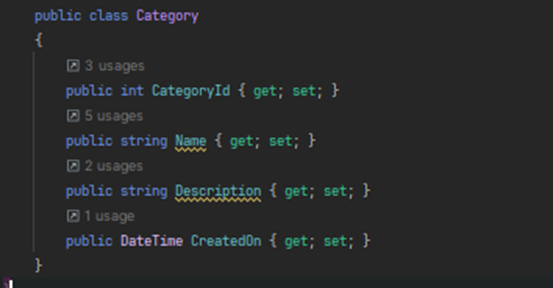
**Преглед на MVC архитектура**

QuestSolve се придържа към архитектурния модел Model-View-Controller (MVC), стандартна дизайнерска практика в разработването на уеб приложения. Този модел разделя приложението на три взаимно свързани компонента, всеки с различна отговорност, насърчавайки чиста организационна структура и улеснявайки поддръжката и мащабируемостта.



**Модел**

• Роля и структура: Моделите в QuestSolve представляват структурата на данните на приложението и бизнес логиката. Те съответстват на обектите на данни като потребител, въпрос, отговор и категория, капсулирайки атрибутите на данните и логиката на взаимодействие с базата данни.

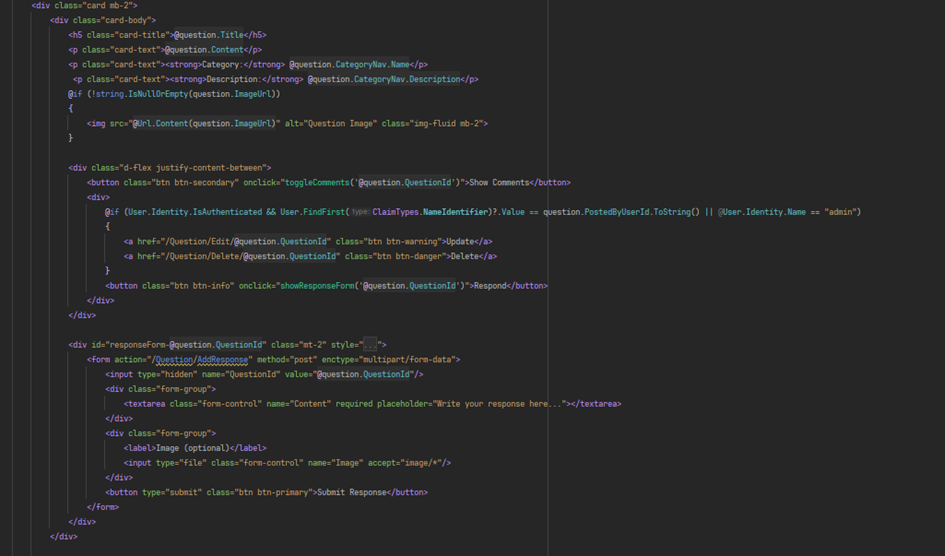


• Интеграция на рамката на обекта: Моделите са интегрирани с Entity Framework за ORM, което позволява ефективно взаимодействие с базата данни, използвайки обектно-ориентирани принципи.

Изглед

• Цел и изпълнение: Изгледите се справят с презентационния слой на QuestSolve. Те се реализират с помощта на Razor View engine, който позволява вграждане на C# код в HTML, създаване на динамични уеб страници.

• Потребителски интерфейс: Предоставените HTML фрагменти демонстрират използването на Bootstrap за стилизиране, осигурявайки отзивчив и удобен за потребителя интерфейс. Изгледите изобразяват данните, предоставени от контролерите, показвайки информация като въпроси, отговори и взаимодействия с потребителите.



**Контролери**

• Функция и примери: Контролерите в QuestSolve действат като посредници между моделите и изгледите. Те обработват входящите заявки, обработват ги (често с помощта на модели) и връщат отговор, обикновено под формата на изглед. Например, HomeController и QuestionController управляват логиката за различни действия като показване на въпроси и обработка на потребителски заявки.

• Маршрутизиране и действия: Контролерите определят маршрути и действия. Те интерпретират потребителските входове и взаимодействия, превръщайки ги в команди за моделите или изгледите.

**Разпределение на отговорностите**

• Подобрена поддръжка и скалация: Чрез разпределение на отговорностите на модели, изгледи и контролери, QuestSolve постига висока степен на модулност. Това разделяне позволява независимо разработване, тестване и поддръжка на всеки компонент.

• Гъвкавост: Промените в бизнес логиката или потребителския интерфейс на приложението могат да се управляват по-ефективно, тъй като модификациите в един компонент обикновено не изискват промени в други.

Специфики на внедряването в QuestSolve

• Поток от данни: Потокът от данни в QuestSolve се придържа към типичния MVC модел. Потребителските заявки се получават от контролери, които след това взаимодействат с модели за извличане или манипулиране на данни. Накрая обработените данни се изпращат обратно към изгледите за представяне.

• Персонализиране и оптимизация: QuestSolve включва персонализации или оптимизации в рамките на MVC, като например специфични конфигурации за маршрутизиране, компоненти за преглед или филтри за действие, съобразени с изискванията на приложението

**ORM взаимоотношения в QuestSolve**

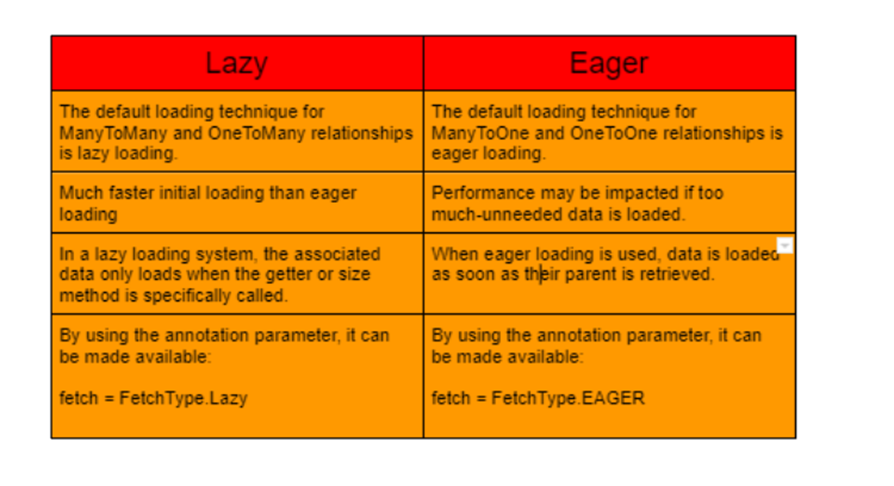
**Конфигуриране на ORM релации**

Работа със свързани данни

 Мързеливо зареждане срещу нетърпеливо зареждане:

o QuestSolve може да използва мързеливо зареждане, при което свързаните данни се извличат при поискване, тъй като са достъпни в кода, или нетърпеливо зареждане, където свързаните данни се зареждат предварително, обикновено използвайки методи като Include.

o Изборът между мързеливо и нетърпеливо зареждане зависи от конкретния случай на употреба и съображенията за ефективност



Предимства в QuestSolve

• Улеснено управление на данни: ORM релациите абстрахират сложността на SQL съединенията и управлението на чужди ключове, което улеснява работата със свързани данни.

• Поддръжка и яснота на кода: Използването на C# обекти за представяне и управление на обекти от бази данни подобрява четливостта и поддръжката на кода.

• Намален код: Автоматизираната обработка на CRUD операции за свързани лица намалява необходимостта от повтарящ се SQL код.

## **Функционалност**

**Функции от страна на сървъра (администратор) в QuestSolve**

Преглед

Функциите от страна на сървъра в QuestSolve, особено тези, пригодени за администратори, играят решаваща роля за поддържането на целостта и гладкото функциониране на платформата. Тези функции позволяват на администраторите да управляват съдържанието, потребителите и различни оперативни аспекти на приложението.

Моделиране на съдържание

 Преглед на въпроси и отговори: Администраторите могат да преглеждат ново публикувани въпроси и отговори, да ги одобряват или да не ги одобряват въз основа на насоките за съдържание на платформата.

 Опции за редактиране и изтриване: Администраторите имат право да редактират или изтриват всеки въпрос или отговор, като гарантират премахването на неподходящо или неподходящо съдържание.

 Управление на категории: Администраторите могат да управляват категории с въпроси, да добавят нови или да редактират съществуващи категории, за да организират по-добре съдържанието.

**Идеи за бъдещи промени и подобрения**

•Управление на категории:

QuestSolve се ангажира със създаването на иновативна и динамична система за рангиране, която ще революционизира начина, по който потребителите взаимодействат с платформата. Тази система е проектирана да автоматично идентифицира, сортира и представя най-популярните и често задавани въпроси от потребителите. С помощта на сложни алгоритми за анализ на данни, системата ще анализира шаблони на търсене и честота на задаване, за да определи кои въпроси са най-важни и актуални за общността.

По този начин, QuestSolve не само ще улесни потребителите в търсенето на информация, но също така ще спомогне за разширяване на знанията и генериране на образователен диалог. Тази система за рангиране ще предостави на потребителите богат избор от подходящи и висококачествени ресурси, които отговарят на техните учебни и информационни нужди.

•Система за гласуване (upvote и downvote):

С въвеждането на системата за гласуване, QuestSolve взема още една стъпка към създаването на интерактивна образователна общност. С тази система, потребителите ще могат активно да участват в оценяването на съдържанието, като използват „Upvote“ за одобрение или „Downvote“ за неодобрение на отговорите. Тази функция ще допринесе за повишаването на обективността и качеството на съдържанието в QuestSolve. Чрез гласуването, потребителите ще имат възможността да изразяват своето мнение и да влияят на видимостта на отговорите в платформата. Това ще спомогне за формирането на среда, в която най-точните, информативни и полезни отговори се издигат на преден план. Системата за гласуване ще насърчава създаването на качествено съдържание от страна на потребителите и ще спомогне за създаването на позитивна и подкрепяща образователна среда.

•Известия при получаване на отговор на въпрос:

С внедряването на системата за нотификации, QuestSolve значително подобрява комуникацията и ангажираността на своите потребители. Всяка път, когато потребителски въпрос получи отговор, авторът на въпроса ще получава известие. Това осигурява непрекъсната връзка между потребителите и дава възможност за бързо реагиране на нова информация или обратна връзка.

Тази функция е от съществено значение за поддържането на високо ниво на ангажираност и интерактивност в рамките на платформата. Потребителите ще могат да следят активно за развитието на техните дискусии и да участват в диалога по един бърз и ефективен начин. Известията ще спомогнат за създаването на по-динамично и ангажиращо учебно преживяване за всички потребители на QuestSolve.

•Административни известия за нови въпроси:

QuestSolve обръща специално внимание на подобряването на управлението на съдържанието и модерацията на платформата. С въвеждането на административни известия, всеки път когато е публикуван нов въпрос, администраторите на платформата ще бъдат уведомявани. Тази функция е ключова за поддържането на високи стандарти на качество и релевантност на съдържанието в QuestSolve.

Известията ще позволят на администраторите да реагират бързо при нужда от модерация, проверка на фактите или предоставяне на допълнителна помощ. Това увеличава ефективността на административния екип и гарантира, че QuestSolve остава сигурно и подкрепящо място за обучение и обмен на информация.

Чрез тези бъдещи разработки и функционалности, QuestSolve се стреми към постоянно усъвършенстване на своята платформа, като цели да предостави на своите потребители най-доброто възможно образователно преживяване.

**Странициранене в QuestSolve**

1. Описание: Странициране е процесът на разделяне на съдържанието на множество страници, което е особено полезно в уеб приложения като QuestSolve, където се обработват големи количества данни, например в списъците с въпроси и отговори. Благодарение на страницирането, потребителите виждат ограничено количество съдържание на всяка страница, вместо да се опитват да прегледат всички данни наведнъж.

2. Предимства на страницирането:

1.Подобрена производителност: Ограничавайки броя на показаните елементи на страница, страницирането намалява времето за зареждане и обработка на данните на сървъра. Това е особено важно за поддържане на бързата реакция на приложението, особено при голям трафик.

2.По-добра потребителска навигация: Страницирането помага на потребителите да се ориентират лесно и да намират конкретно съдържание, което търсят. Това подобрява общото потребителско изживяване, тъй като потребителите могат да преминават между страниците без претоварване от информация.

Реализация в QuestSolve: В QuestSolve, странициране е интегрирана в различни функционалности на приложението, като списъци с въпроси.Това се постига чрез използване на техники и алгоритми, които динамично разделят съдържанието на множество страници в зависимост от броя на записите в базата данни и потребителските предпочитания.

Употребата на странициране в QuestSolve е ключов елемент за осигуряване на ефективно управление на съдържанието, оптимална производителност и отлично потребителско изживяване, особено в контекста на големи обеми данни и активна потребителска интеракция.

# Заключение

QuestSolve представлява платформа, която успешно преодолява предизвикателствата на съвременното дигитално обучение. Интегрирайки динамични образователни ресурси, високо технологични решения, QuestSolve предоставя на училищните общности уникална среда за съвместно обучение и обмен на знания. Отличавайки се със своите строго контролирани форуми и административни одобрения, платформата гарантира безопасност и уважение в учебния процес, докато предоставя възможност за активно и ангажиращо участие на всеки ученик.

Основните предимства на QuestSolve включват:

• Фокус върху училищната среда: Специално създадена за удовлетворяване на нуждите на училищата, QuestSolve предлага сигурна и структурирана платформа за обучение, далеч от общите социални медии или форуми.

• Контрол от администратори: Всички публикации и участници се одобряват от администраторите, което осигурява поддържането на високи стандарти за качество и уместност на съдържанието.

• Интерактивност и сътрудничество: QuestSolve насърчава съвместното учене и взаимопомощ, което обогатява образователния опит на учениците.

• Технологични иновации: Използването на съвременни технологии като Entity Framework и LINQ в QuestSolve допринася за интуитивен и ефективен потребителски опит.

В допълнение към текущите си характеристики, QuestSolve планира въвеждането на нови функции като система за рангиране на често задаваните въпроси, упвоут/даунвоут система за оценка на отговори, и подобрени нотификации, които ще улеснят още повече общуването и обмена на знания в рамките на платформата.

QuestSolve, като иновативно и напредничаво образователно решение, притежава значителен потенциал да трансформира начина, по който училищата използват цифровите ресурси, като същевременно съхранява и поддържа ключовите образователни ценности и цели.

# Ресурси и използвана литература

1. Conole, G. (2012). "Designing for Learning in an Open World".

2. Dempsey, J. V. & Watkins, R. O. (2014). "Privacy and Security in Online Learning: A Guide".

3. Freeman, A. (2017). "Pro ASP.NET Core MVC 2".

4. Ito, M., Gutierrez, K., Livingstone, S., Penuel, B., Rhodes, J., Salen, K., ... Watkins, S. C. (2013). "Connected Learning: An Agenda for Research and Design".

5. Nilsson, J. (2006). "Applying Domain-Driven Design and Patterns: With Examples in C# and .NET".

6. Richardson, L. & Ruby, S. (2007). "RESTful Web Services".

7. Shank, P. (2011). "The Online Learning Idea Book: Proven Ways to Enhance Technology-Based and Blended Learning".

8. Simonson, M., Smaldino, S., & Zvacek, S. (2019). "Teaching and Learning at a Distance: Foundations of Distance Education".

9. Stephens, R. (2008). "Beginning Database Design Solutions".

10. Sullivan, B. & Liu, V. (2011). "Web Application Security, A Beginner's Guide".

11. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

12. https://en.wikipedia.org/wiki/C\_Sharp\_(programming\_language)

13. https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

14. https://levelup.gitconnected.com/database-first-vs-code-first-in-entity-framework-fd786fa1dfd6

15. https://www.codingninjas.com/studio/library/lazy-vs-eager-loading-in-hibernate

16. https://en.wikibooks.org/wiki/A-level\_Computing/AQA/Problem\_Solving,\_Programming,\_Operating\_Systems,\_Databases\_and\_Networking/Databases/Normalisation

17. https://www.javatpoint.com/orm-c-sharp

18. https://www.reddit.com/

19. https://restfulapi.net/

20. https://www.mysql.com/

# Приложения

## **Source code на проекта**

фдфдфг

## **Теоретична разработка – pdf и word файл**

## **Екрани (Screenshots)**

……………….

!!!!

Структурата се надявам да те ориентира. Не всичко е разбито на подточки. Може да си добавяш колкото нива на heading-ите искаш само не ги махай, защото се използват за автоматично генериране на съдържанието. За сега таблицата със съдържание е направена да визуализира 2 нива на heading-ите, но може да се направи и за повече.

Когато си готов с приложението и теоретичната разработка, се носят на флаш и сорса и word файла, като трябва да имаш указания какво е нужно и как да ти се сетне/ стартира приложението.

Искам обаче и кода и теорията качени в github, ако нямаш акаунт ще си направиш. Ако дойдат пак да ни проверяват, няма да ровя из флашките.

Ей тук в 8ма точка един линк към source на приложението и един линк към теорията. Ресурси изброяваш всякакви книги, сайтове и т.н откъдето ти е взета теорията. За схеми препоръчвам да са подходящи за печат в черно и бяло.

След като е готова теоретичната част, разписаното задание е подвързва заедно с дипломния проект. Kогато се стигне до печат е по-добре да го дадеш за принтиране в PDF формат.

**НЕ бързаш с принтирането!!!**

Трябва да проверя до колко е важен типа на шрифта, по принцип миналата годна е било times new roman, но това е отвратително за четене затова съм задала друг шрифт

Допълнително инфо в час ☺