УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ

КАТЕДРА «ИНФОРМАЦИОННИ СИСТЕМИ И ТЕХНОЛОГИИ»

МАГИСТЪРСКА ПРОГРАМА
„ИНФОРМАЦИОННИ СИСТЕМИ И ТЕХНОЛОГИИ“

МАГИСТЪРСКА ТЕЗА

на тема:

ЕТАПИ НА РАЗРАБОТКА НА УЧЕБЕН СОФТУЕР

Дипломант:
Мартин Котрулев
дистанционно обучение
Фак. № 0230-имд

Научен ръководител:…..……..
(Проф. д-р Георги Димитров)

София
2018

УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ

ДЕКЛАРАЦИЯ

От Мартин Борисов Котрулев

С настоящата декларация отстъпвам безвъзмездно право на УНИБИТ да публикува представената моя авторска разработка (курсова работа, дипломна работа, магистърска теза) като общодостъпен ресурс за безплатен публичен достъп чрез информационните системи на УНИБИТ.

Дата:..... **Подпис:**.....

(дипломант)

Съгласен съм авторската разработка на магистърска теза да бъде публикувана като общодостъпен ресурс за безплатен публичен достъп чрез информационните системи на УНИБИТ.

Дата:..... **Научен ръководител:**……

(подпис)

УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ

ДЕКЛАРАЦИЯ

От Мартин Борисов Котрулев

Декларирам, че представената дипломна работа/магистърска теза е подготвена и изпълнена самостоятелно от мен.

При откриване на плагиатство поемам съответната отговорност по смисъла на чл. 31 (1-3) от Наредбата.

Дата:..... **Подпис:**.....

(дипломант)

РЕЗЮМЕ

Котрулев, М. Етапи на разработка на учебен софтуер. Научен ръководител проф. д-р Г. Димитров. София. 2018. Катедра «Информационни системи и технологии». Магистърска програма ”Информационни системи и технологии”. УНИБИТ. 97 с. Брой източници – 15. Брой графики - 35.

Цели на магистърската теза са да бъдат представени и разгледани етапите при разработка на учебен софтуер. Постигнатите резултати свързани с магистърската теза е реализацията на сървърна част с прилежаща база данни и клиентски, уеб базиран клиент, част от реално функциониращо учебно приложение за тестово изпитване.

Ключови думи: База данни; Отворен код; Софтуерна разработка; Тестове; Учебни системи; C#; GitHub; JavaScript; JSON; .NET Core; Node.js; PostgreSQL; React; REST;

СЪДЪРЖАНИЕ

УВОД

Живеем в ера на ежедневни дигитални революции. Това наистина вълнуващо време налага и неизбежния стимул да бъдат преоткривани начините по които работим, учим и прекарваме свободното си време. Почти всеки отрасъл на съвременния живот, по един или друг начин е свързан с дигитализацията. Ежедневно биват разработвани все по-нови и иновативни начини, които да помагат на хората да бъдат по-продуктивни като им се предоставя оптимален и бърз начин да достъпват информацията, която ги интересува.

Перфектен пример за огромните ползи от дигитализацията е образованието. Не само че информационните технологии дават възможност на хора по целия свят да получават евтино и достъпно образование, но предлагат и все по-добри начини да се оптимизира учебният процес. Съществуват множество платформи предлагащи видео уроци, интегрирани системи с училищната система, университетски дистанционни програми, а бумът на независими организации и частни колежи, предлагащи курсове и възможности за личностно развитие доказват, че дигиталното образование е необходимо и търсено.

Създаването на софтуер, който да обслужи високите изисквания на потребителите е сложен процес. Неговото създаване може да отнеме много време и да бъде обвързано с голям колектив от разработчици. Това налага допълнителни съображения относно избор на платформа, начини на доставка и интеграция на софтуера, технологии за разработка както и качествено планиране на процесите.

Разработката на цялостна учебна система би била непосилна задача в рамките на няколко месеца и особено от един разработчик, а документацията би могла да достигне размера на няколко курсови проекта. Именно за това настоящата работа е фокусирана в изграждането на една от основните части на учебните системи - приложение за тестово изпитване.

По един или друг начин всяка учебна институция предлага начин за оценяване и проследяване на учебния напредък. За да може този процес да е максимално опростен и същевременно максимално ефективен, почти всяка система се спира пред избора на някакъв вид тестово изпитване. Това дава бърз и ангажиращ начин на потребителя да проверява своя напредък, а пред изпитващия орган - автоматизиран начин за оценяване и категоризиране на учебната дейност.

Както стана дума по-рано, настоящата работа има за цел да разгледа процеса на разработка, на едно примерно приложение за тестово изпитване. То е наречено ***UniQuizBit*** и идеята, стояща зад него е да предостави максимално бързо и опростено решение за изпитване в класната стая. Лесното създаване и решаване на теста е с цел, да могат да се правят често и периодично. Така материала в учебното заведение може да бъде усвоен по-лесно. Освен това тестовото приложение заимства някои елементи от социалните мрежи. Идеята е всеки да може да създава групи с тестове или единични тестове, да им добавя тагове, с които по лесно да ги намират други потребители и да ги споделя. Така би могло да се създават множество тестове в различни категории и групи.

Първата глава разглежда избраните технологии за разработка, където освен набора от технологии, са разгледани и основните характеристики на всяка една технология, с цел да бъде обосновано тяхното използване. Втора глава продължава със структурата на базата данни, която стои в основата на учебното приложение. Добавено е подробно описание на всяка таблица и е дадено описание за какво се използва. Трета глава запознава с архитектурата на сървърното ***API***, описва как работи и какви похвати са използвани, за да се възползва максимално от технологията, на която е създадено. Четвърта глава е посветена на уеб базираното, потребителско приложение. Описани са и допълнителните технологии, които помагат за неговото по-лесно разработване. В последната глава се разглеждат евентуалните начини на развитие и заложените възможности в сегашната му версия.

Основната причина за избора на темата е именно постоянното и непрестанно развитие на уеб технологиите, създаването на лесни за ползване и

достъпни от много хора приложения. Основните интереси на нейния автор са именно в тази посока. Вълнуващо е че живеем във времена, в които е нещо нормално да чуваме около себе си думи като изкуствен интелект и криптовалути. Още по-вълнуващо е споделянето на знание чрез технологиите. Целите на настоящото приложение не са само да бъде част от по-голяма учебна система, но и да може да бъде използвано и самостоятелно, като един забавен и лесен начин да се споделя знание под формата бързи и достъпни тестове.

Първа глава представя изследване на актуални, онлайн учебни платформи, образователни системи с отворен код и популярни, уеб базирани тестови приложения.

Втора глава разглежда подбора и аргументира използваните технологии за изграждане на цялостното решение. Разгледани са технологиите PostgreSQL за база данни, DBeaver за мениджмънт на базата, C# като сървърен език, платформата .NET Core, езика JavaSript и библиотеката React за финал системата за контрол на версиите GitHub.

Трета глава е посветена на същинската разработка. Обърнато е внимание на базата данни и релационните връзки между моделите, които изграждат гръбнака на сървърната услуга. Представена е архитектурата на сървърното приложение и похватите на работа чрез платформата .NET Core. В последната част за разгледани възможностите за развитие на приложението както и възгледи за интеграция в по-големи, цялостни платформи.

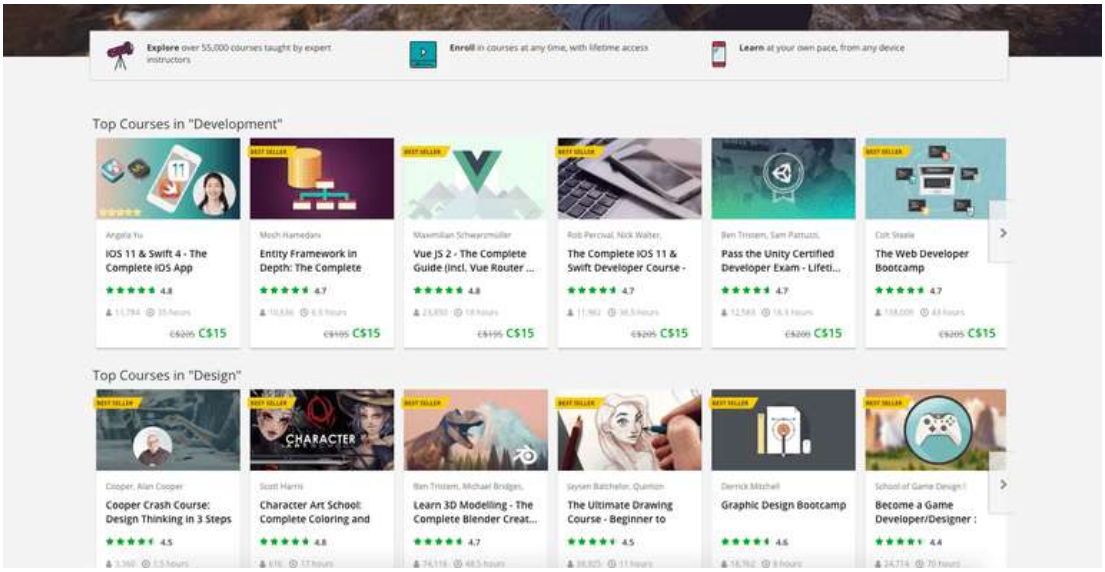
I. УСЛОВИЯ ЗА РЕАЛИЗАЦИЯ НА УСПЕШЕН УЧЕБЕН СОФТУЕР

Условията за реализирането на успешен учебен софтуер са много, но безспорно в основата стои проучването на вече готови продукти, върху които да се изгради основа от добри практики, популярни похвати и актуалност на продукта. В настоящата глава основна цел представлява проучването на популярни платформи и приложения свързани с тематиката.

1.1 Популярни учебни платформи

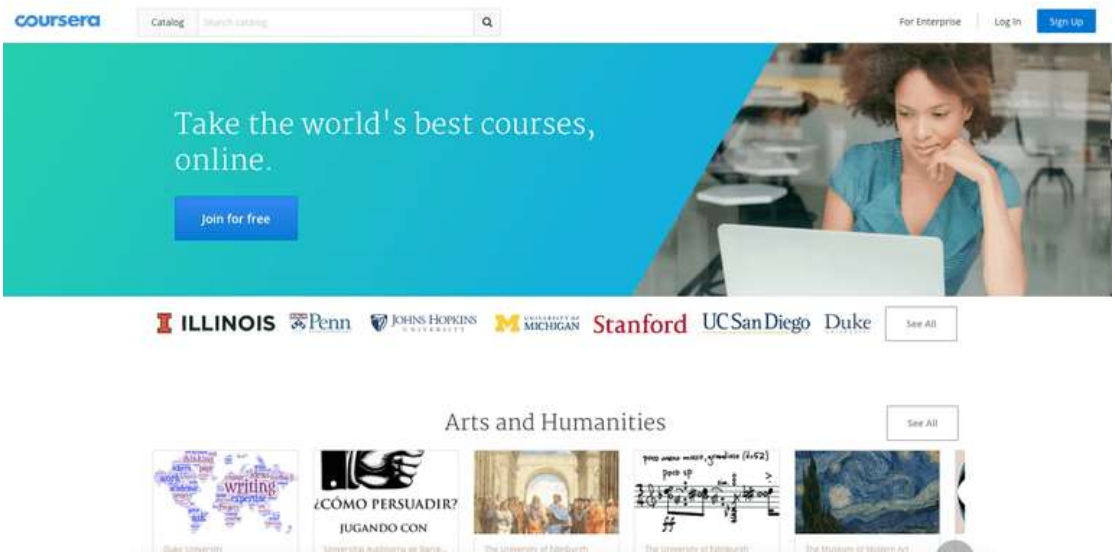
(11) Един от популярните начини за достъпно и евтино образование в момента са учебните уеб платформи. Образователните институции и индивидуалните експерти, които желаят да предлагат курсове в мрежата се нуждаят от място, което да съхранява въпросните курсове и да ги прави достъпни до потребителите. Именно това прави тези платформи силно желани. Някои платформи са тясно свързани с технологиите, а други предлагат курсове в широк диапазон от теми. Следва списък с най-популярните платформи в момента и това, което предлагат – от курсове за напълно начинаещи до професионалисти:

- Udemy



Udemy е сайт предлагащ онлайн обучение в широк диапазон от теми. Той е на първо място в този списък заради голямата си популярност и безценните си ресурси. Могат да бъдат намерени повече от 55,000 курсове в най-различни области, а мобилната апликация позволява ученето да продължи и в движение. Курсовете в Udemy варират от напълно безплатни и стигат до стотици долари, но наличието на редовни намаления стигащи ниско до $10 го прави много изгоден. Всеки, който е експерт в своята област може да стане инструктор в Udemy и да се възползва от огромната база потребители, с които постоянно да бъде в контакт.

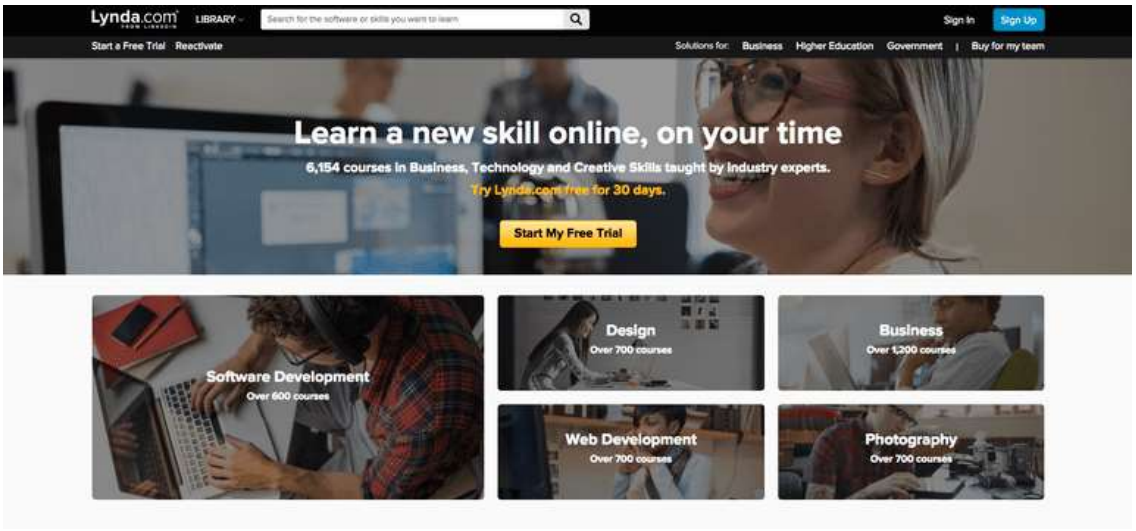
- Coursera



Когато става въпрос за курсове предлагани от над 140 от най-добрите университети и организации в САЩ, Coursera е точното място. Coursera си партнира с университети като Университета на Пенсилвяния, Станфордския Университет, Университета на Мичиган и др. като предлага универсален достъп то най-доброто образование в света. В него могат да бъдат намерени над 2,000 платени и безплатни курсове в над 180 направления като технологии, бизнес, социални науки и др. Coursera

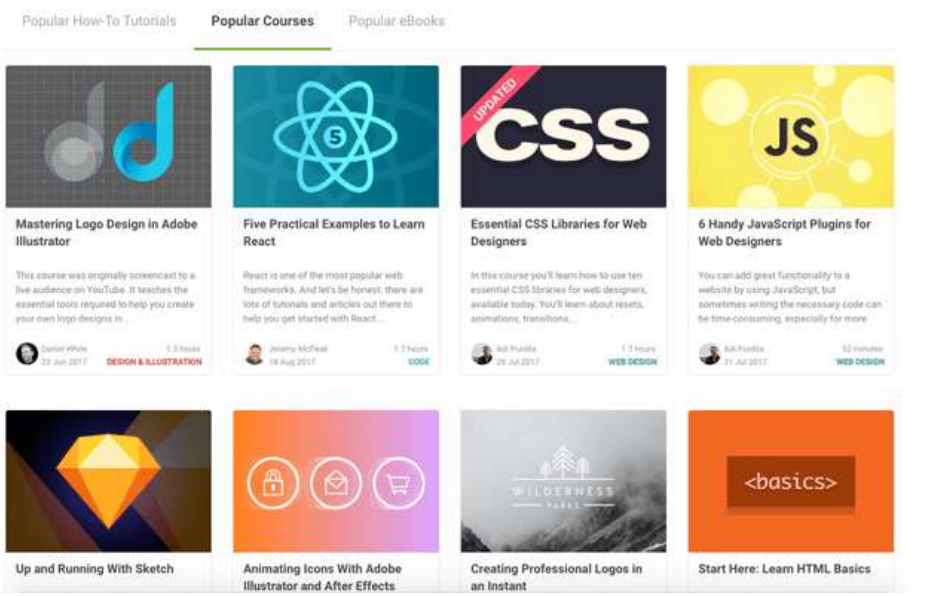
също предлага мобилна апликация, с която всеки може да учи със собственото си темпо.

- Lynda



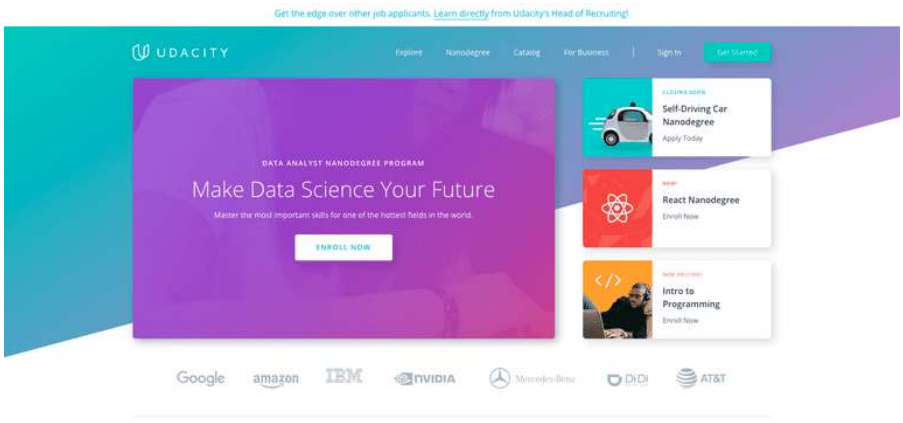
Собственост на LinkedIn, Lynda е популярен образователен център за професионалисти търсещи нови способности в бизнеса, креативността и технологиите. Курсовете спадат под категории като анимация, аудио/музика, бизнес, дизайн, разработка, маркетинг, фотография, видео и др. При регистрация в Lynda, потребителя получава 30-дневен, безплатен тестови период, след което бива таксуван ежемесечно по $20 за основно членство или $30 членство с допълнителни облаги. Lynda притежава, така наречения, „реактивен“ план, при който всеки може да деактивира членството си по всяко време и да продължи когато желае като прогреса по курсовете се възстановява мигновено.

- Tuts+



Tuts+ на Envato е за тези, които работят и се забавляват в сферата на креативните технологии. В допълнение на обширната библиотека от „как да направим“ ръководства съществуват курсове за дизайн, илюстриране, уеб дизайн, фотография, видео обработка, бизнес, музика, аудио, 3D анимация и графика. Tuts+ разполага с повече от 22,000 ръководства и над 870 видео курса, с допълнителни курсове, които биват добавяни всяка седмица. За съжаление отсъства безплатен пробен период за платформата, но въпреки това членството е достъпно с цена от $29 на месец.

- Udacity



Отдаден на доставянето на високо образование за света, по възможно най-достъпните и ефективни начини, Udacity предлага онлайн курсове и материали, обучавайки своите ученици на умения, които с релевантни и търсени в индустриите. Неговите създатели твърдят, че предлаганото образование е на цена, която е скромна част от цената на големи образователни центрове. Това е перфектна платформа за всеки, който планира да работи в сферата на технологиите. С курсове и материали за Android, iOS, наука за данните, софтуерен инженеринг и уеб разработка, платформата предлага образование в крачка със всички съвременни и иновативни методологии използвани от днешните компании и стартъпи.

Горепосочените технологии са само част от огромната култура на дигитално образование. Всяка една платформа предлага на своите потребители персонализиран прогрес, които се съблюдава по различни начини включващи тестове, прогресни значки и сертификати. Един от минусите на тези платформи, е че те са отворени към потребителите да създават курсове и не гарантират винаги най-високо ниво на съдържанието. Това от своя страна, обаче, може да бъде лесно преценено от обратната връзка на обучаващите се, които могат да дават оценки и коментари.

Изводите, които могат да се направят от тези платформи е, че онлайн образованието е търсено , а сериозното предлагане от страна на платформите подобрява всеки ден материалите, които се достъпват от крайния потребител.

1.2 Популярни учебни системи с отворен код

(10) В тази част ще бъде обърнато внимание на сериозните системи с отворен код, които са успешно интегрирани в много образователни центрове по целия свят.

- Moodle

  Без съмнение, Moodle е една от най-популярните LMS (Learning Management System) с отворен код в момента. Тя предлага функционалности като учебни табла, прогрес на обучаващия и мултимедийна поддръжка. Тази обучителна мениджмънт система предлага също способ за създаване на мобилно съвместими онлайн курсове и интеграция с добавки на трети страни. За тези, които желаят да продават техните електронни курсове, Moodle си партнира с PayPal, за да направи процеса по плащане лесен и бърз. Едно от отчетливите предимства на системата е неговата потребителска общност. За разлика от много други отворени LMS решения, всеки може да получи отговори на належали въпроси почти веднага, достъпвайки онлайн поддържаната база данни и да свали предварително подготвени курсове и материали, които да спестят времето и проблемите за създаване на собствени такива. Редно е да се отбележи, че Moodle може да е по-сложен за нови потребители, но усъвършенстването на продукта си заслужава ако се търси абсолютна дизайнерска свобода.

- ATutor

  Друга учебна система с отворен код е ATutor. Тя предлага широк набор от функционалности, вариращи от уведомления по електронна поща до файлово хранилище. Един от основните акценти на ATutor е, че той е приятелски настроен към своите потребители и лесно достъпен, което го прави перфектен за тези, които са нови в света на онлайн дизайна и програмирането на електронни учебни системи. Системата предлага и широк набор от теми, забързващи допълнително процеса на разработка както и инструменти за оценяване, резервни файлови копия, инструменти за анализ и анкетна интеграция.

- Eliademy

  Тази отворена система е напълно безплатна за педагози и създатели на електронно обучение, но за допълнителни екстри се заплаща малка сума на база потребител. Притежава курсови каталози, инструменти за оценка и дори мобилна, Android апликация за учители желаещи да създават модули за мобилно обучение, за своите потребители в движение. Създателите на електронни курсове могат да споделят със своята аудитория веднага, стига потребителите да са регистрирали своята електронна поща.

- Forma LMS

  От анализ на пропуски в уменията до детайлни аналитични и докладни инструменти, Forma LMS е пакетирана с разнообразие от функционалности. Също така се гордее със своите сертификати, мениджмънт на компетентност и широк обхват от мениджмънт инструменти за виртуална стая, включващи календари и мениджър на събития. Тази отворена система е идеална за корпоративни тренировъчни програми и предлага активна онлайн общност където всеки може да потърси съвети и трикове, за да вземе максимално от нея.

- Dokeos

  За всяка организация търсеща предварително сглобени, курсови елементи, Dokeos може да се окаже идеалният избор на LMS, който е и безплатен за до 5 потребителя. Притежава сериозен набор от шаблони и инструменти за бързо създаване на собствени курсове. Съпътстващия сайт предлага голямо количество полезна информация включваща видео упътване за всяка стъпка на процеса по създаване материали. Интерфейса е интуитивен, което го прави идеален за новите професионалисти в сферата на онлайн обучението или за тези, които не желаят да се занимават със сложни за заучаване системи.

- ILIAS

  За финал е оставена една от най-гъвкавите, лесно разширими и полезни в широк диапазон от нужди системи. Това е първата учебна мениджмънт система, която е съвместима със SCORM 1.2 и SCORM 2004 (множество технически стандарти оказващи на програмистите създаващи подобен софтуер как да пишат своите системи така, че да са напълно съвместими с други подобни системи). Също така, ILIAS е една от малкото LMS, които действат двойно като напълно сътрудническа платформа, с която всеки може да комуникира със своя екип и да споделя документи. Тя е напълно безплатна за всички разработчици както и за организациите предлагащи обучение независимо от броя на потребителите. Ако изискванията са за сериозен брой потребители тази система е идеална, тъй като в повечето случаи цената се смята на потребителска база. Някои от основните акценти са собствена система за електронна поща, провеждане на тестове и оценяване и споделяне и изпращане на файлове и ресурси. Важно е да се отбележи, че тази система е избрана от УНИБИТ за доставяне на материали, следене на учебния прогрес и съставяне на тестове за своите студенти, с което прави обучението и комуникацията между студенти и преподаватели по-лесно и по-достъпно за всички.

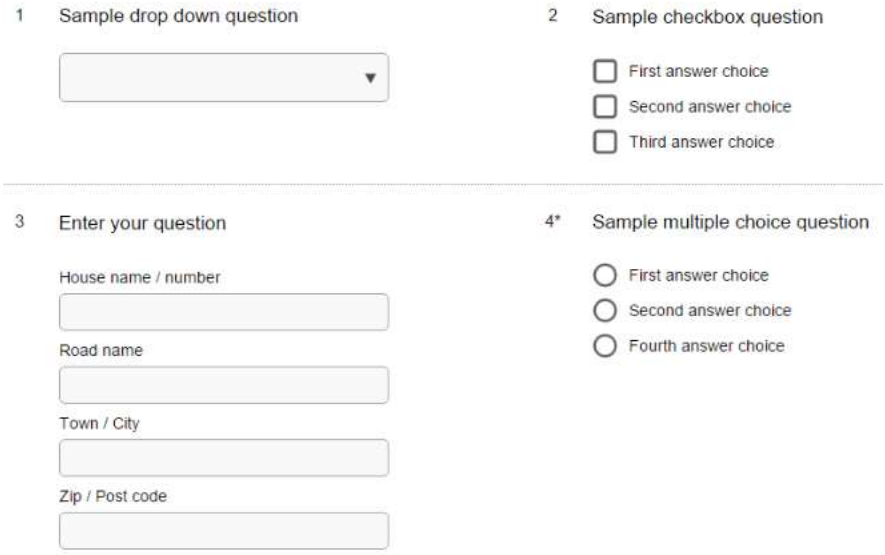
От казаното по-горе става ясно, че за да може една система да е успешна то тя трябва да притежава някои основни качества, които да я направят атрактивна както за обучаващите се така и за разработчиците на учебните материали. Основните качества, които се наблюдават в много системи са поддръжката на потребителско пространство, възможност за споделяне с други потребители, начин за следене и оценяване на обучението както и инструменти за създаване на изпитни материали под формата на тестове. Допълнителни качества, които подобряват достъпа на потребителите и правят приложенията по желани са поддръжката на мобилни устройства и отворения код с, което се дава възможност с свободна интеграция и надграждане на услугата.

1.3 Популярни приложения за тестово изпитване

(12) В края на тази глава ще бъдат разгледани и някои популярни уеб приложения за тестово изпитване, с които да бъдат проследени тенденциите при създаване на този тип приложения тъй като това е основната задача на настоящата работа.

- Free Online Surveys

  FreeOnlineSurveys.com е популярен инструмент за бързо създаване на онлайн запитвания, генериране на гласувания и генериране на форми.



  Функционалности:

  - Създаване на запитване без специално необходими способности.
  - Лесно разпространение (чрез електронна поща, споделяне на кратък URL, добавяне на кратък код в собствен сайт и т.н.).
  - Визуален доклад за събраните данни.
  - Добавяне на картинки, видео, бланки за попълване и въпроси с множество отговори.

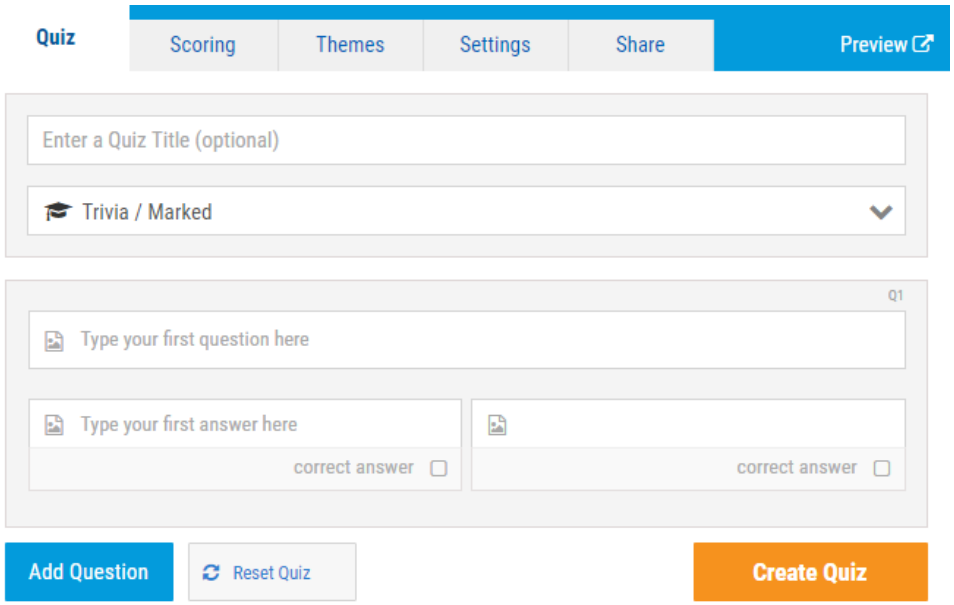
  Лицензионни опции:

  - Този инструмент се предлага безплатно с основните си функционалности. Това може и да не е достатъчно за бизнес/обучителни изисквания като потребителски групи и централизиран мениджмънт. За планове с допълнителни възможности, които дават възможност за съхранение на данните се

заплаща $19.99 на месец (или $9.99 за студенти и академични организации).

- Poll Maker's Quiz Maker

  Quiz Maker от Poll Maker е друга лесна за използване услуга, която спомага за създаването на тестове. Същата лесна процедура на създаване на въпроси, задаване оценки и споделяне на резултатите.



  Функционалности:

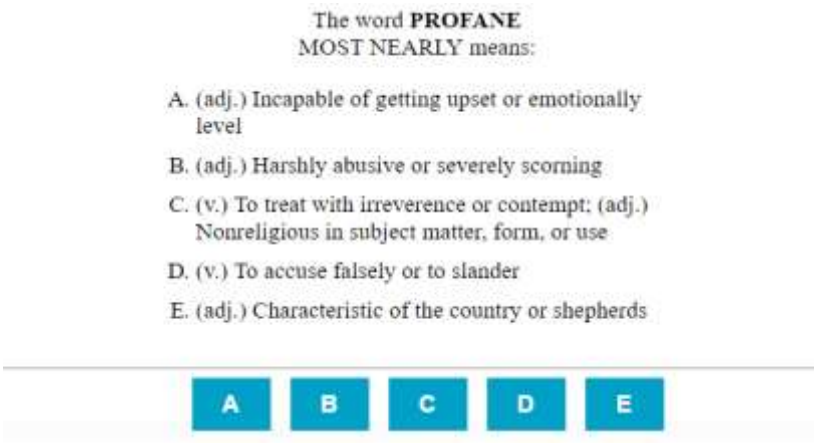
  - Интерактивно и ангажиращо съдържание.
  - Поддръжка на картинки и медия, автоматична интеграция със социални мрежи.
  - Автоматична система за оценка.
  - Тестване чрез мобилно устройство и резултати на живо.

  Лицензионни опции:

  - Напълно 100% безплатна услуга, която дава всички базови възможности.

- Vocabtest

  Онлайн услуга от „старата школа“ за провеждане на тестове свързани с лексиката. Vocabtest идва с безплатен тестов създател, който може да бъде използван за създаване на печатни тестове чрез регистрация.



Функционалности:

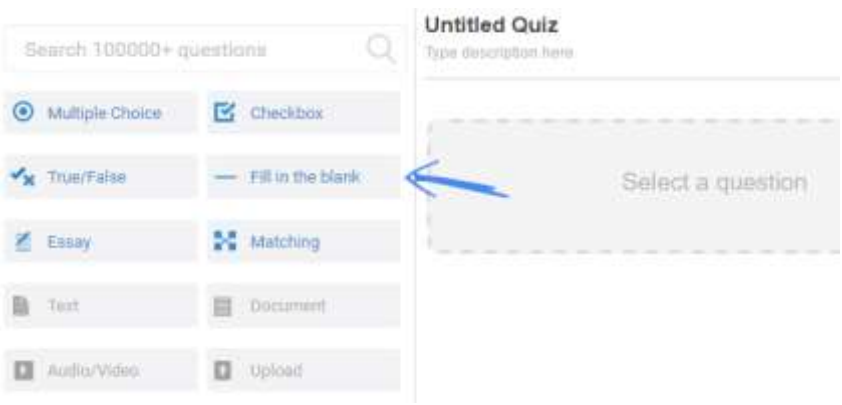
- Избор на брой думи и техните дефиниции.
- Добавка на подсказки, синоними/антоними и частни случаи.
- Споделяне с ученици.

Лицензионни опции:

- Услугата се предлага безвъзмездно. Това идва на цената на изписване на рекламни материали, а желаещите потребители могат да даряват пари за допълнителни подобрения.

- ProProfs Quiz Maker

  ProProfs Quiz Maker е облачно базиран софтуер за създаване и доставяне на онлайн изпити и тестове. Решението ProProfs помага за създаването на авторизира тестове, с които да се дават защитени задания на студенти или служители. Услугата идва с предварително подготвени шаблони, автоматизиран система за оценяване и интеграция с ProProfs LMS.



Функционалности:

- Дава възможност за създаване на тестове чрез шаблони или съвсем нови.
- Интуитивен интерфейс с „привлачване и пускане“.
- Автоматично оценяване.

Лицензионни опции:

- ProProfs Quiz Maker е така наречения freemium софтуер. Безплатната услуга покрива почти всичко необходимо с изключение на доклади и затворени тестове (всички тестове за публични по подразбиране). Платените услуги варират от $9 до 199$ за бизнеса. Допълнителни екстри са облачно хранилище, маркетингови инструменти и поддръжка на сложни екстра планове.

- ClassMarker.com

  ClassMarker е уеб базиран софтуер и тестови създател за ученици и бизнеса. В сценариите за използване фигурират бизнес и тренировъчни тестове, тестове за подбор и преди назначаване, обучителни задания, училища, университети, онлайн курсове и т.н.

Функционалности:

- Създаване на тестове с лимитирано време, публични и частни по достъп тестове, случайни въпроси, моментално отчитане на прогреса, множество отговори, кратки отговори и ред въпросни типове.

- Създаване на множество въпроси или случайно подбиране от банки с въпроси.
- Резултатите се оценяват веднага.
- Предлага обратна връзка за индивидуален въпрос или за цялостния тест в реално време.
- Разделя индивидуалното и групово справяне с теста, въпросите или категориите.

Лицензионни опции:

- ClassMarker предлага бизнес и обучителни планове. Бизнес лицензът струва $39.95/$79.95 месечно (400/1000 теста). Обучителните организации с нестопанска цел и единични потребители могат да се възползват от безплатен лиценз, даващ право на 100 теста месечно. Безплатния акаунт не дава възможност за сертификати, резултати по електронна поща и качване на картинки и файлове.

Става напълно ясно, че при разработката на приложение за тестово изпитване е желателно да присъстват някои основни елементи:

- Трябва да има възможност потребителите да създават свои тестове лесно и бързо както и да ги коригират.
- Да имат възможност да определят броя и тежестта на въпросните отговори.
- Да могат да споделят или ограничават тестовете с аудиторията.
- Да създават групи в които да споделят своите тестове.
- Да присъства автоматизирана система за оценка.

Към допълнителните качества на приложението, което да го направят максимално популярно и достъпно могат да спаднат следните:

- Да бъде с отворен код за модификации.
- Да може да бъде интегрирано с големи учебни системи.

- Да бъде безплатно или да предлага достъпни планове за своите потребители.
- Да предлага ограничени от времето или защитени от парола тестове.
- Да бъдат разбъркванивъпросите и отговорите.
- Бързо намиране на групи и тестове по име или категория.

Именно това са и основните точки по които се придържа автора на настоящата работа при разработката на тестовата услуга.

С тази глава бяха положени основите при проучването на съществуващите платформи, обучителни мениджмънт системи и самостоятелни приложения за тестово изпитване. Чрез проучването бяха изведени основните качества, които трябва да притежава един успешен инструмент за онлайн обучение.

II. ПОДБОР НА НЕОБХОДИМИ ТЕХНОЛОГИИ ЗА РАЗРАБОТКА

Основен етап от разработката на всеки един софтуер, без значение дали е учебен или не, е подбора на технологиите за неговото разработване. Добре обмисления избор е важно да бъде съобразен с много фактори (като големина на екипа от разработчици и тяхната запознатост с конкретната технология, бюджет, сложност на работа и т.н.) и изиграва основна роля за успешното и бързо завършване на проекта. В настоящата глава ще бъдат разгледан подбора на технологии и причините за конкретните избори.

2.1 База данни: PostgreSQL

PostgreSQL (9) е специална, защото не е просто база данни. Тя е също и платформа за приложения и то впечатляваща. PostgreSQL позволява писането на съхранени процедури и функции на няколко програмни езика. В допълнение на това към предварително пакетираните езици, могат да се добави поддръжка и за други чрез използването на разширения. Примери за вградените езици, за писане на съхранени функции са ***SQL*** и ***PL/pgSQL***. Някои от езиците, които могат да се добавят чрез разширения са ***PL/Perl, PL/Python, PL/V8*** (също познат като ***PL/JavaScrip***t), и **PL/R**. Много от тези пакети идват наготово с най-често срещаните дистрибуции. Тази широка поддръжка от разнообразни езици, позволява решаването на проблеми, най-добре засегнати от специфични в областта, процедурни или функционални езици. За пример, използването на статистическите и графични функции на R и неговите кратки и стегнати идиоми в областта, използване на уеб услуги чрез Python или писане на оптимизирани конструкции и използването им директно в SQL изрази. Могат да бъдат писани дори агрегиращи функции във всеки един от тези езици, с което да бъдат комбинирани агрегиращите способности на SQL и силните черти на всеки един език, за да се постигне повече отколкото само със SQL. В допълнение е възможно писането и извикването на съхранени функции директно на C. Функции писани на няколко различни езика, могат да вземат участие в една заявка. Могат да бъдат дефинирани дори агрегиращи функции, съдържащи само SQL. За разлика от ***MySQL*** и ***SQL Server***, не се налага компилация за построяването на агрегиращите функции в PostgreSQL. Накратко - могат да се използват правилните инструменти за всяка задача дори когато подзадачите изискват различни инструменти. Чист SQL може да бъде

използван в области, които повечето бази данни не позволяват. Могат да бъдат създавани много сериозни функции без да се налага компилация.

Възможността за създаване на персонализирани типове в PostgreSQL е усъвършенствана и лесна за използване, съперничеща си и в много случай и дори надминаваща други бази данни. Най-близкия съперник от гледна точка на поддръжка на персонализирани типове е ***Oracle***. Могат да се дефинират нови типове данни в PostgreSQL, които след това да бъдат използвани като типове на табличните колони. Всеки от новите типове върви с съответстващ масив, за да могат да се запазват и масиви или да се използват в SQL изрази. В допълнение могат да бъдат дефинирани и оператори, функции и индексатори за работа с новите типове. Много разширения, производство на трети страни, за PostgreSQL се възползват от тези допълнителни възможности, за да постигнат по-добра производителност, областно-специфични конструкции за по-кратък и лесен за поддръжка код и за изпълнението на задачи, за които може само да се мечтае в други бази данни.

Ако реализацията на собствени типове и функции не е търсена от потребителя, PostgreSQL предлага широк набор от вградени такива какъвто е JSON (добавен с версия 9.2) както и разширения добавящи още много други. Много от тези разширения, както беше споменато по-рано, са пакетирани заедно с PostgreSQL дистрибуции.

PostgreSQL 9.1 внедрява нова SQL конструкция, ***CREATE EXTENSION,*** която позволява инсталирането на добавка чрез единичен SQL израз. Всяка добавка трябва да бъде инсталирана в конкретната база, с която се работи. Чрез CREATE EXTENSION, могат да бъдат инсталирани споменатите PL езици и популярни типове заедно с техните функции и оператори като ***hstore key-value*** хранилище, ***ltree hierarchical*** хранилище, ***PostGIS*** пространственото разширение и безброй други. За инсталирането, например, на популярното разширение PostgreSQL key-value хранилищен тип и прилежащите му функции, оператори и индексаторни класове, командата която трябва да бъде изпълнена е ***CREATE EXTENSION hstore;*** В допълнение съществува и команда, с която да бъдат изброени наличните и инсталирани разширения.

Всяка нова версия на PostgreSQL добавя нови възможности, увеличава използваемостта, подобрява скоростта на работа и разтегля границите на възможното по отношение на това, което може да бъде постигнато с релационна база. Много хора дори биха се зачудили - защо някои би използвал друга база данни при положение, че PostgreSQL е в състояние да свърши всичко, за което човек се надява и то безплатно. Няма четене, на лицензионните правила и цени на другите бази данни и чудене, колко ще струва използването на 8 ядра на сървъра с добавянето на X,Y, и Z функционалности. От горе на това PostgreSQL работи сравнително консистентно под всички поддържани платформи. Разработено приложение, което е създадено с цел дистрибуция между клиенти използващи Unix, Linux, Mac OS X, или Windows, ще работи без проблем на всяка една от тези операционни системи. Също така инсталации са налични за всички тях, правейки достъпа бърз и надежден без да се налага компилация.

Важно е да се отбележи, че за целите на настоящата работа и връзката с .NET Core сървърното приложение се грижи ***Npgsql***, което представлява ***ADO.NET*** доставчик на данни с отворен сорс код. Npgsql позволява връзката между програми написани на C#, Visual Basic, F# да достъпват PostgreSQL сървър за база данни. Имплементиран е изцяло в C# код използването му е напълно безплатно. Конкретната имплементация, която използва настоящото приложение е доставчика за Entity Framework Core. Наличен е и доставчик за Entity Framework 6.x.

2.2 Мениджър на база данни: DBeaver

DBeaver (5) се появява през 2011 година и стабилно добива популярност от тогава. Към момента DBeaver е един от най-популярните инструменти за мениджмънт на бази данни в световен мащаб. Неговата популярност се простира измежду над 500, 000 активни потребители и продължава да расте. Една от неговите основни черти, допринасяща за това е, че се предлага в две основни версии - ***Enterprise*** и ***Community*** като втората е напълно безплатна и с отворен код. Другата е, че поддържа всички популярни бази данни като ***MySQL, PostgreSQL, MariaDB, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata, Firebird, Derby*** и др.

Разработчиците на "бобъра" стартират проекта, за да създадат надежден и умен инструмент, удобен за мениджмънта на множество административни дейности свързани с базите данни.

Какво представлява DBeaver всъщност?

Мултифункционалност - С използването на DBeaver, нуждата от множество инструменти за контрол на отделните аспекти от мениджмънта на базата данни отпада. DBeaver консолидира всички тях:

- Анализ на базата данни - удобни инструменти за разработката на визуални диаграми, изградени от отделни обекти или от цели схеми, тяхното персонализиране и експорт, както и мениджмънт на метаданните.

- Обработка и промяна на данните - Лесен и интуитивен начин за разглеждане и редакция на данните, подсилена допълнително от сортираща и филтрираща функционалност.

- SQL Разработка - Пълноценен и бързо работещ SQL редактор за разработка, изпълнение, съхранение, експортиране и преизползване на скриптове с функции за профилиране и форматиране. SQL би могъл да се превърне, сам по себе си, в отделен продукт за разработка и използване от професионални SQL разработчици, но е част от DBeaver, което е страхотно.

- Администрация на базата данни - Широк обхват на функционалности за поддръжка и анализ на базата данни, експорт и импорт на данни, потребителски сесии, мениджмънт на заключващите механизми и много др. Разработчиците постоянно разработват нови възможности и доразвиват съществуващи, за да заздравят способностите на DBeaver.

- Поддръжка на всички основни платформи - DBeaver работи под всички основни операционни системи: Windows, MacOS и всички популярни Linux дистрибуции – CentOS, Ubuntu, и т.н. Инсталаторите на DBeaver притежават всички необходими сертификати и задоволяват изискванията за сигурност.

- Поддръжка на почти всички бази данни - Директно от кутията DBeaver поддържа всяка база която притежава ***JDBC*** или ***ODBC*** драйвер – а това е почти всяка релационна база и повечето NoSQL бази. Освен JDBC и ODBC, DBeaver поддържа бази данни, които нямат стандартни xDBC драйвери като ***Mongo DB, Redis*** или ***WMI***. Пълният лист на всички поддържани бази се намира на адрес https://dbeaver.com/databases/ Добавянето на поддръжка за все повече бази данни е постоянен процес, тъй като

основни тенденции в момента са науката за данни, работата с огромни данни, бази данни за търсене (***Solr***) и облачни бази данни.

- Лесен за използване от всяка категория потребители - DBeaver е еднакво добър за професионалисти и аматьори в мениджмънта на бази данни. Това е така, защото превръща работата в интуитивен, лесен и удобен процес. С DBeaver, комплексните задачи за анализ и администрация се оказват доста по-лесни отколкото изглеждат на пръв поглед. DBeaver не изисква специализирано, техническо знание за поддръжката на базата данни и осигурява помощ, когато тя е нужна.

Удобството, което предоставя DBeaver е незаменимо в днешната динамична среда, тъй като внезапна смяна на базата данни при промяна на изискванията или по-време на взимането на решение за доставчика, не налага изучаването, всеки път, на нов инструмент за работа. Освен активното участие на DBeaver при разработката на сървърното приложение, всички диаграми представени в настоящата писмена работа са директно взети от него.

2.3 Сървърен език: C#

(13) C# е програмен език, в основата на който стои синтаксис, много близък до този на Java. Въпреки това да бъде наречен C# клонинг на Java би било доста неточно. Реално и C# и Java са част от, така нареченото, C семейство от програмни езици (вкл. C, Objective C, C++) и следователно споделят подобен синтаксис. Много от синтактичните конструкции на C# са моделирани спрямо различни аспекти от Visual Basic (VB) и C#. Например, както VB, C# поддържа

концепцията за ***class properties*** (заместваща традиционните getter и setter методи) както и незадължителните параметри. Както C++, C# позволява презаписването на функционалността на операторите (***operators overload***) както и създаването на енумерации и ***callback*** функции (чрез делегати). Не само това но C# поддържа и много от похватите на функционалните езици като ***LISP*** и **Haskell** (***lambda*** изрази и анонимни типове). Това което прави езика още по-уникален, обаче, е появата на Езиково Интегрираните Заявки (Language Integrated Query или LINQ както е неговата абревиатура), който позволява множество интересни конструкции. Въпреки всичко казано до сега C# е силно повлиян от C-езиците, а това че е хибрид на много от тях, резултатът е чист (ако не и по-чист) като Java, почти толкова семпъл, колкото е VB и притежаващ почти толкова мощ и гъвкавост, колкото и C++. Ето и основните характеристики намиращи се във всяка версия на C#:

- Липса на указатели! C# програмите рядко имат нужда от директна манипулация на указателите (въпреки че подобна функционалност е налична).

- Автоматичен мениджмънт на паметта - чрез така наречения ***garbage collector***, в следствие на което липсва и ключовата дума ***delete*** в езика.

- Формални конструкции за класове, интерфейси, структури, енумерации и делегати.

- C++ способност за презаписване на функционалността на операторите, при добавени типове, без допълнително усложняване.

- Поддръжка на атрибутно-базирано програмиране. Този вид разработка позволява добавянето на анотации към типовете, с цел да се допълни тяхното поведение. Например ако се отбележи метод с [Obsolete] атрибут, програмистите, които го използват ще получат предупреждение, че методът вече не се използва и в бъдеще може да бъда премахнат.

С появата на .NET 2.0 (2005 година) C# беше обновен с много нови функционалности като по-значимите са:

- Възможността за създаването на обобщени типове (***ganeric types)*** и обобщени членове. Използвайки обобщени типове, програмистът може да създава ефикасен и типово-защитен код, в който типовете могат да бъдат дефинирани в момента на използване.

- Поддръжка на анонимни методи позволяващи дефинирането на функции директно на мястото на аргументи от тип делегат.

- Способността за дефиниране на един тип в множество файлове чрез ключовата дума ***partial.***

.NET 3.5 (пуснат през 2008) добавя дори повече нови функционалности:

- Поддръжка на силно типизирани заявки (LINQ), които се използват за обработка на различни видове данни.

- Поддръжка на анонимни типове позволяващи моделирането на типове, без предварително да са дефинирани.

- Възможност за допълване на функционалността на даден тип (без да се налага наследяване) чрез ***extension*** методи.

- Добавя се lambda оператора (=>), който допълнително улеснява работата с делегатните типове.

- Нов синтаксис позволяващ задаването на обектните свойства още по време на неговото инициализиране.

.NET 4.0 (2010) отново добавя значими добавки към C#:

- Поддръжка на незадължителни параметри към методите както и поименно подаване на аргументи.

- Поддръжка на динамично достъпване на членове чрез ключовата дума ***dynamic.*** Това позволява унифициран начин за достъпване на типовете без значение в коя библиотека са имплементирани.
- Работата с обобщение типове (***generics)*** става по интуитивно

С излизането на версия 4.5, C# получава двойка нови ключови думи, а именно ***await*** и ***async***, с които многонишковото и асинхронно програмиране се улесняват значително. Работилите с по-ранни версии на C# биха си спомнили, че за извикването на методи в нови нишки се изискваше сериозно количество код и използването на различни .NET пространства от имена. Чрез новата двойка процесът по извикване на методи асинхронно става лесно като извикването им синхронно. C# 6 е издаден заедно с .NET 4.5 и добавя редица миниатюрни подобрения за по-добър код:

- Автоматични свойствата и инициализацията им директно на същия ред и поддръжка на ***read-only*** такива.

- Поддръжка на lambda методи имплементирани на един ред.

- Добавени са статични импорти позволяващи директен достъп до статичните членове, част от конкретно пространство от имена.

- Null-условният оператор, който помага при достъп до методи на null членове както и несъществуващи елементи от масив.

- Нов синтаксис за форматиране на низове чрез интерполация.

- Способност за филтриране на изключения с новата ключова дума - ***when***.

- Използване на ***await*** в ***catch*** и ***finally*** блокове.

- Израза ***nameOf*** връща символно-низово представяне.

- Нов подобрен начин за ***overload.***

Най-новата версия излязла към момента на писане на настоящия труд - .NET 4.7 (от март 2017) идва с версия 7 на C# и подобрява допълнително кодовото конструиране. В тази версия са добавени и някои значителни нововъведения (като ***tuples***, ***ref locals*** и ***ref return***), които разработчиците са искали да се добавят към спецификацията от доста време:

- Деклариране на променливи директно като аргументи.

- Локални функции в методи с цел да се ограничи обхвата.

- Допълнителни изразни членове (напр. C# 7 позволява конструктора да бъде деклариран с ламбда израз).

- Генерализирани ***async*** return типове.

- Леки анонимни типове (наречени ***tuples***) които притежават множество полета.

- Подобрен логически поток чрез допълнително съпоставяне на типове, а не само на стойност.

- Способност за връщане на референция към стойност, а не само стойността при употреба на ***ref locals*** и ***ref returns***.

- Добавени са леки неизползваеми променливи (наречени ***discards*** и използвани при деконструирането на ***tuples*** напр.).

- ***Throw*** изрази позволяващи хвърлянето на изключение на повече места като условни изрази, ламбди и др.

Не дълго след като C# 7 е пуснат излиза и първото разширение - C# 7.1 (През Август 2017). Това малко разширение добавя следните допълнения:

- Основния ***main*** метод може да бъде асинхронен.

- Нов литерал ***default*** позволяващ инициализирането на всеки тип.

- Корекция на проблема свързан с новото шаблонното сравняване(при употреба на ***is*** и ***as*** операторите), който не позволяваше сравняването с общи типове.

- Автоматично съвпадане на локалните променливи при задаване на елементи в ***tuple,*** съкращаващ дублирането на поименните елементи.

Нужно е да се отбележи, че езикът C# може да бъде използван за създаването на софтуер, който се управлява от ***.NET runtime*** среда (не е възможно създаването на ***unmanaged (***неуправляемо***)*** приложение както C/C++). Официалният термин, описващ код, който таргетира .NET платформата се нарича ***managed*** (управляем, тъй като се управлява от ***CLR - Common Language Runtime***). Бинарният пакет съдържащ управляемия код носи наименованието асембли. .NET платформата може да работи на различни операционни системи. Това дава възможност приложение, което е разработвано на ***Windows*** машина може да бъде пуснато на ***macOS*** и обратното чрез ***.NET Core.***

Управляемата среда, заедно с официалното пускане на .NET Core дава възможност за разработка и доставяне на C# приложение на всички големи операционни системи в момента. Лекотата на работа с езика, огромната общност от програмисти, добрата документация и конкурентна производителност прави C# идеален избор за сървърното приложение към настоящият труд.

2.4 Сървърно приложение: ASP.NET Core

(13) На 27 Юни, 2016, Microsoft обявява първото издание на .NET Core версия 1.0. Революционна нова платформа за всички .NET разработчици по целия свят. Тази нова платформа се издава едновременно за Windows, macOS и Linux като се базира на C# и библиотеката .NET. Основното издание включва в себе си .NET Core средата за работа, ***ASP.NET Core*** и ***Entity Framework Core***. Следват две допълнителни, основни ревизии: 1.1 и 2.0 (в момента на писане). Допълнително към способностите за доставяне на множество платформи е въведена и друга сериозна промяна. .NET Core платформата и обвързаните библиотеки са напълно с отворен код. Не само са с отворен код, но са и напълно разработени отворени проекти. Заявки за сваляне се приемат и дори се очакват. Сериозен принос от повече от 10,000 разработчици се включват в началното издание на .NET Core. Това, което започва като сравнително малък екип от разработчици в Microsoft, сега достига широката общност програмисти, които имат възможността да участват в добавянето на допълнителна функционалност, подобрения в производителността и изчистване на бъгове. Тази отворена инициатива включва не само софтуера, но и документацията.

Целта на .NET Core да работи под различни операционни системи, сама по себе си достатъчно висока, не е единствената. Като продукционна библиотека, .NET е наоколо от 2002 година и доста неща са се променили от тогава. Разработчиците са станали по-умели, компютрите - по-бързи, а изискванията на потребителите все по-високи. Допълнително мобилните приложения са водещи към настоящия момент. Следва списък с основните насоки в които е съсредоточен .NET Core:

- Едновременна поддръжка на множество платформи - .NET Core може да работи под Windows, Linux и macOS. Приложенията могат да бъдат разработени на различни платформи чрез ***Visual Studio Code*** или ***Visual Studio for Mac.*** Допълнително ***Xamarin*** добавя ***iOS*** и ***Android*** поддръжка към целевите платформи.

- Производителност - производителността на .NET Core стои консистентно близо до върха във всички релевантни чартове и всяка нова версия носи със себе си допълнителни подобрения. Когато става въпрос за производителност то тя стои на първо място в дизайна на ASP.NET Core. Основната библиотека .NET е нараснала масивно през последните 15 години и изтискването на производителност от трудно-променим код би било трудно в най-добрия случай. Тъй като .NET Core е до голяма степен пренаписване на платформата и прилежащите библиотеки, архитектите са имали възможността да помислят за оптимизации преди и по време на разработката. Основната дума в случая е "преди". Производителността сега е интегрална част при взимането на решение за бъдещите разработки.

- Портативни, класови библиотеки използваеми на всички версии на .NET runtime - .NET Core въвежда така наречения .NET ***Standard***, което представлява унифицирана спецификация, целяща еднаквост на поведението при различни версии на .NET средата. Оперативната съвместимост с .NET библиотеката, от появата на Visual Studio 15.3 и .NET Core 2.0, позволява и референция към основните библиотеки. Тази добавка предоставя по-добър механизъм за преносимостта на съществуващ вече код от и към .NET Core. Съществуват и някои ограничения разбира се. Асемблитата, които биват реферирани могат да използват само типове, част от .NET Standard.

- Преносимост или самостоятелност при доставка на софтуера - приложенията могат да бъдат доставени заедно с .NET библиотеката или да използват вече инсталирана такава на целевата машина. .NET Core позволява съвместно съществуване на различни инсталации на платформата за разлика от основната .NET библиотека. Всяка инсталация на нова версия не променя съществуващите вече. Това разширява моделите на доставяне. Преносимите приложения се конфигурират да таргетират версия инсталирана на целевата машина и добавят само специфичните си пакети. Така се запазва малкия размер на инсталацията, но изискват таргетираната библиотека да е предварително инсталирана. Самостоятелните пакети съдържат всички специфични файлове заедно с необходимите CoreFX и CoreCLR файлове за таргетираната платформа. Това очевидно прави инсталационния пакет по-голям, но изолира приложението от проблеми на машинно ниво (като премахване на целевата версия на .NET Core).

- Пълен контрол през командния ред - осъзнавайки, че не всеки ще използва Visual Studio за разработка, тимът стоящ зад .NET Core решава да се съсредоточи първо върху поддръжката на тази функционалност преди да разработи инструменти за Visual Studio. Преди версия 2.0 инструментите изостават спрямо това, което програмистът може да постигне с текстов редактор и .NET Core ***CLI*** (***Command Line Interface***). Самите инструменти са доста напреднали от версия 1.0 насам, но все още са назад спрямо инструментите за основната .NET библиотека.

- Допълнителни опции за доставяне на софтуера - .NET Core носи със себе си допълнителни възможности. Linux сървърите традиционно са по ефтини и по-лесни за поддръжка от Windows еквивалентите, особено когато става въпрос за облачни технологии, така че възможността за доставяне на уеб приложения и услуги може да доведе до значителни спестявания на средства. Това е особено привлекателно за стартъпи, и

малки компании, които се опитват да намалят броя на Windows сървърите си.

- Контейнеризация - допълнително към поддръжката на Linux дистрибуции, .NET Core поддържа и контейнеризация на приложения. Популярни доставчици на контейнери като Docker например значително намаляват сложността при издаване на приложения за множество платформи. Всички необходими файлове на приложението (включително и необходимата операционна система) се пакетират заедно в един контейнер, който може да бъде копиран от среда на среда без да се налага инсталация. С добавянето на Docker поддръжка под Windows и Azure нещата се подобряват още повече. Възможна е разработка директо в контейнера и когато стане време за интеграция и тест просто се мести контейнера под необходимата среда. Няма нужда от инсталации и сложност на доставянето. Отпада и грижата за несъвместимост между средите.

Плюсовете на ASP.NET Core са неоспорими когато става въпрос за създаване на качествени и надеждни приложения в днешната динамична среда. Основните черти като лесна преносимост, максимално добра производителност и разбира се мултиплатформеност стоят в основата на избора за сървърната технология.

2.5 Уеб език: EcmaScipt 6

В своето ядро JavaScript езикът и неговите характеристики са дефинирани в ECMA-262 стандартът. Езикът дефиниран по този стандарт се нарича ECMAScript. Когато се споменава JavaScript в отношение на интернет браузър или Node.js всъщност става въпрос за разширение на ECMAScript. Браузърите и Node.js добавят допълнителна функционалност чрез добавянето на допълнителни обекти и методи, но основата на езикът си остава този, който е дефиниран в ECMAScript. Продължаващото разработване на ECMA-262 е основополагащо за успехът на JavaScript като цяло, а ECMAScript 6 е най-новата основна версия.

TC-39, основният комитет отговорен за движението на ECMAScript разработката, сглобява огромна чернова на спецификацията за ECMAScript 4, който е масивен по обхват и включва много малки и големи промени в езика. Някои от тях са нов синтаксис, модули, класове, класическо наследяване, private обектни членове, незадължителни типови анотации и др. Обхватът на ECMAScript 4 промените предизвикват разцепление в TC-39. Някои от членовете чувстват, че с четвъртата версия се прави опит за постигането на твърде много. Група лидери от Yahoo!, Google, и Microsoft предлагат алтернатива за ECMAScript, която наричат ECMAScript 3.1. “3.1” номерирането цели да покаже, че версията има за цел да подобри съществуващия стандарт. ECMAScript 3.1 добавя само няколко подобрения в лицето на добавени методи към съществуващи обекти, подобрения по работата със свойствата на обектите и техните атрибути и вградена поддръжка на JSON. Въпреки че е положено

усилие да се събере ECMAScript 3.1 и ECMAScript 4, в крайна сметка това води до провал, тъй като двата лагера не успяват да стигнат до съгласие относно бъдещото развитие на езика. През 2008 Брендан Ейк, създателят на JavaScript, заявява че ТС-39 ще се фокусира в стандартизирането на ECMAScript 3.1 като работата по ECMAScript 4 ще продължи само след като текущата версия бъде стандартизирана, а инициативата ще бъде наречена ECMAScript Harmony. В крайна сметка ECMAScript 3.1 е стандартизиран като петото издание на ECMA-262 и е обявен като ECMAScript 5. Комитетът никога не издава ECMAScript 4, за да не се асоциира с неразбирателството, което предизвиква. Работата продължава по така наречения ECMAScript Harmony с ECMAScript 6 като първа, основна версия на новия "хармоничен" начин на работа, включващ всички членове на комитета.

ECMAScript 6 достига завършен статус през 2015 и формално е наречен ECMAScript 2015. Нововъведенията варират от напълно нов вид обекти и дизайн шаблони до синтактични промени и добавени методи на съществуващи обекти. Вълнуващото на промените е, че те са насочени директно към решаването на проблеми, които разработчиците срещат постоянно в своята работа:

- Блоков обхват - ключовите думи ***let*** и ***const*** добавят блоково обвързване и обхват на променливите в JavaScript. Тези декларации биват издигнати в началото на блока и съществуват само в него. Блоковият обхват предлага поведение присъщо на повечето езици и ограничават случайните грешки, тъй като променливите се декларират само и единствено там където са необходими. Страничен ефект от това е невъзможността да се достъпват променливи, които не са декларирани дори с сигурни оператори като ***typeof***. Опита за достъп до такива променливи предизвикват грешка ако преди това съществува външна променлива със същото име преди блока. В много отношения, ***let*** и ***const*** се държат по начин сходен с ***var***. Това не е варно, когато става въпрос за цикли. Декларираните функции в рамките на цикъла могат да достъпват сегашната ***let***-итерирана стойност, а не само финалната (както е поведението с ***var***), защото получават ново копие при

всяка итерация. Най-добрите практики за блоковия обхват са да се използва ***const*** по подразбиране и ***let*** когато се очаква промяна в променливата. Следвайки този принцип се осигурява основно ниво на неизменимост в кода, което помага за избягването на някои типове грешки.

- Низове - пълната поддръжка на UTF-16 в ECMAScript 6 позволява логичната работа със символи. Способността да се превръщат символи в код чрез ***codePointAt()*** и ***String.fromCodePoint()*** е важна стъпка в низовото манипулиране. Добавянето на "u" флага в регулярните изрази дава възможност за оперирането върху кодови точки вместо 16-битови символи, а ***normalize()*** методът позволява по сигурни низови сравнения. ECMAScript 6 добавя и нови методи за обработване на низове, позволяващи по лесна идентификация на поднизове независимо от тяхната позиция.

  Допълнителна функционалност е добавена и в регулярните изрази. Друго много ценно нововъведение са шаблонните литерали, които позволяват по-лесното създаване на низове. Способността да се вграждат променливи директно в шаблона дава на разработчиците по-надежден начин за композицията на сложни низове чрез.

  Вградената поддръжка на многоредовите низове прави шаблонните литерали незаменима добавка, която не е била част от езика до този момент. Въпреки, че новите редове са позволени директно в шаблона, използването на ***\n*** и други избягващи символи също са позволени.

  Най-важната част от шаблоните са допълнително дефинираните таговете. Това са функции, които приемат частите на шаблона като аргументи. След това използват данните за да върнат подходяща низова стойност. Данните могат да бъдат други литерали, суровият им еквивалент както и други заместители.

- Функции - Функциите не претърпяват огромни промени с ECMAScript 6, а по-скоро получават подобрения за улеснение на работата. Незадължителните параметри позволяват да бъде зададена изрична стойност когато даден аргумент не е подаден. Преди, това става с допълнителен код за проверка на аргументите.

  "Останалите" параметри (дефинирани с оператора за разпъване **…**) позволяват масив по подразбиране, който да приеме останалите аргументи. Използването на истински масив (вместо вградения ***arguments***) позволява много повече гъвкавост. Разпъващият оператор позволява и деконструирането на масиви, и обекти в отделни параметри. Преди ECMAScript 6 имаше само два начина да се подадат отделни параметри, които са част от масив: ръчно подаване или използвайки функцията ***apply()***. Чрез разпъващия оператор към всяка функцкия лесно може да се подаде масив без притеснение за зачисляването на ***this***.

  Добавянето на ***name*** свойството помага за по-лесното идентифициране на функциите при тестване.

  ECMAScript 6 формално дефинира и поведението на блок-функциите, така че да не бъдат синтактична грешка в стриктен режим.

  В ECMAScript 6, поведението на функцията е дефинирано от ***[[Call]]***, при нормална функционална работа и от ***[[Construct]]*** когато е извикана с оператора ***new***. Най-голямата добавка са стрелковите функции. Те са създадени с цел да заменят анонимните функционални изрази. Функциите със стрелка има по-стегнат синтаксис, не презаписват ***this*** контекста и нямат вградения ***arguments***. Липсата на ***this*** презаписване не позволява стрелковите функции да се използват като конструктори.

- Обекти - обектите стоят в центъра на JavaScript програмирането и ECMAScript 6 добавя някои помощни промени, с които ги прави доста по-гъвкави и лесни за работа. Променени са обектните литерали. По-лесно зачисляване на свойствата на обекта, носещи същите имана на дефинирани променливи в текущия обхват. Добавен е и по-кратък начин

за добавяне на методи позволяващи пълното изключване на двоеточието и ключовата дума ***function***.

Методът ***Object.assign()*** прави промяната на няколко свойства едновременно по-лесно и е много полезно, когато се използва шаблона ***mixin.***

Методът ***Object.is()*** прави стриктна проверка за еднаквост на всяка стойност, превръщайки се в по-сигурно версия на оператора ===.

ECMAScript 6 ясно дефинира реда на собствени стойности на обекта. Ключовете с числови стойности са първи, подредени в изкачващ ред, след което следват низовите и символните в реда в който са добавени.

Сега е възможно модифицирането на обектния прототип чрез метода ***Object.setPrototypeOf()*** след като е създаден обекта. В добавка на това, може да бъде използвана ключовата дума ***super*** за извикване на методи от прототипа. Причисляването на ***this*** в супер метода става автоматично към сегашната стойност на ***this.***

- Итератори и генератори - Итераторите са важна част от ECMAScript 6 и са от основно значение за някои ключови елементи на езика. На повърхността итераторите дават лесен начин да върнем поредица от стойности, използвайки семпло API. Въпреки това съществуват доста по-комплексни начини да се използват. Новият ***for-of*** цикъл прикрито използва итератор, за да върне серия от стойности в цикъла. Използването му е по-лесно от традиционния ***for*** цикъл, защото не се налага проследяването на стойностите, за да се определи кога приключва. За да се улесни допълнително работата с този цикъл, всички колекции в ECMAScript 6 притежават итератори, правейки стойностите им лесни за достъп. Низовете също притежават такъв итератор, с който е възможно обхождането по символи (вместо по кодови групи). Разпъващият оператор работи с всички итериращи обекти и конвертирането им в

масиви става лесно. Самото конвертиране се извършва чрез прочитане на стойностите от итератора и добавянето им индивидуално в масива.

Генераторите представляват специална функция, която създава итератор при нейното извикване. Дефинициите на генераторите се индикират чрез звезда (*) и използват ключовата дума ***yield***, която показва коя стойност да бъде върната при всяко поредно извикване на метода ***next()***. Делегирането през генераторите насърчава добра енкапсулация за поведените на итераторите, давайки възможност за преизползването на генератори в нови генератори чрез използването на ***yield** **** вместо ***yield***. Този процес дава възможност за създаването на итератор, който има способността да връща стойности от множество итератори. Може би най-интересния и вълнуващ аспект на генераторите и итераторите е способноста да се създава по-чист асинхронен код. Нуждата от използването на ***callback***-функции навсякъде е заменена от код, който прилича на синхронен, но фактически използва ***yield*** и изчаква завършването на асинхронните операции.

- Обещания (***Promises***) - “Обещанията” са проектирани да подобрят асинхронното програмиране в JavaScript, давайки повече контрол над синхронните операции, от колкото дават събитията и отвръщащите функции. Обещанията планират задачи, които да бъдат добавени в опашката за събития, част от двигателя на JavaScript, за изпълнение на бъдещи задачи, а втора опашка за задачи следи изпълнението или отхвърлянето, за да осигури правилно осъществяване.

  Обещанията имат три състояния: в очакване, осъществено и отказано. Обещанието започва в състояние на очакване и преминава в осъществено при успех на изпълнението или в отхвърлено при провал. И в двата случая могат да се добавят обработващи методи. Методът ***then()*** служи за отработване на успех или отгвърлане, а ***catch()*** - само за отгвърляне.

Обещанията могат да бъдат навързани верижно по много начини, за да подават информация помежду си. Всяко извикване на ***then()*** създават ново обещание, което се обработва след като е обработено предишното. Подобно навързване може да бъде използвано за активирането на отговори към серия от асинхронни събития. Съществуват и методите ***Promise.race()*** и ***Promise.all()*** за мониторинг по прогреса на множество обещания наведнъж.

Асинхронните задачи са облагодетелствани и от комбинирането между обещания и генератори, защото обещанията осигуряват общ интерфейс, който асинхронните операции могат да връщат. След това могат да бъдат използвани генераторите с оператора ***yield***, за да се изчака асинхронния отговор. Повечето модерни уеб ***API***-та са изградени на базата на обещания и могат да се очакват още много други да последват в бъдеще.

- Модули - ECMAScript 6 добавя модулите като вградена способност към езика като начин за пакетиране и енкапсулация на функционалност. Модулите се държат различно от скриптовете, по това че не модифицират глобалния обхват с техните променливи, функции и класове на топ ниво, а стойността на ***this*** е ***undefined***. За да се постигне това поведение, модулите се зареждат в различно състояние. Всяка функционалност, която ще бъде използвана, трябва да бъде експортирана преди това. Могат да бъдат експортирани променливи, функции и класове, както и да бъде зададена експортирана (само една) стойност по подразбиране. След като бъдат експортирани, друг модул може да импортира всички или само някои от членовете. Имената които се задават на импортите, играят роля на нормално дефинирани променливи с ***let***, оперират на блоково ниво и не могат да бъдат повторно декларирани в един и същ модул.

  Модулите няма нужда да експортират каквото и да е, ако манипулират нещо в глобалния обхват. Такива модули може да се импортират без добавянето на обвързващи променливи.

Тъй като модулите трябва да работят по различен начин, браузърите добавят ***<script type="module">***, с който се заявява че сорс файла или кода стоящ директно между таговете, трябва да бъде изпълнен като модул. Модулите, които се зареждат с ***<script type="module">*** зареждат с атрибута ***defer***(отложено). Също така модулите се зареждат в реда, по който са подредени в съдържащия документ, след като той е напълно зареден и проверен.

JavaScript е гъвкав и способен език. Важно е да се отбележи, че нововъведенията, които идват с ECMAScript 6 го сближават много по синтактични конструкции с C#. Това ги прави идеална комбина за разработката на уеб приложения от така наречените ***full-stack*** разработчици. Разбира се, ECMAScript 6 дава възможност за разработка и на сървърната част с помощта на Node.js, но това не влиза в рамките на настоящата работа. Node.js ще бъде обсъден от гледна точка на работата с React в следващите глави.

2.6 Уеб приложение: React

(14) Съществуват множество JavaScript ***MVC (Model View Controller)*** библиотеки. Защо разработчици от Facebook създават React и какви са неговите предимства?

React не е MVC базирана библиотека.
React е библиотека са създаване на композитни, потребителски интерфейси. Това ще рече, че библиотеката насърчава създаването на преизползваеми компоненти, представящи изменчиви данни, които се променят с времето.

React не използва шаблони.

Традиционно, потребителския интерфейс на уеб приложенията се изгражда посредством шаблони или HTML директиви. Тези шаблони диктуват пълния набор от абстракции, които са необходими за изграждането на потребителски интерфейс. React подхожда към това изграждане различно - чрез разбиването на интерфейса по компоненти. Това означава, че React използва истински, пълноценен програмен език за рендирането на изгледите в приложенията, което е предимство поради няколко причини:

- JavaScript е гъвкав, мощен, програмен език имащ способността да създава абстракции. Това е от изключителна важност при създаването на големи приложения.

- Чрез унифицирането на маркиращия език (какъвто е HTML) и прилежащата логика на изгледа, React всъщност прави изгледите по-лесни за надграждане и поддръжка.

- Чрез предварителното "сготвено" разбиране на маркировката и съдържанието директно в JavaScript кода се избягва ръчното навързване на низове и следователно се намалява площта за ***XSS*** (***Cross-Site Scripting***) уязвимост. В допълнение идва и разширението JSX (позволяващо миксирането на HTML с JavaScript и позволяващ дефинирането на персонализирани тагове) за тези, които предпочитат четимостта на HTML пред суровите JavaScript методи. Именно JSX е предпочетен при разработката на уеб приложението към настоящата работа.

Реактивните промени са максимално интуитивни.

React наистина грее, когато става въпрос за динамичното обновяване на данните. В традиционния подход на JavaScript приложенията се налага постоянна проверка, какви данни се променят, за да се направят изрични промени по документно-обектния модел (DOM), за да се осигури неговата

адекватност. Дори библиотеката ***AngularJS***, която дава декларативен интерфейс с помощта на директиви и обвързване на данните, налага свързващи функции, с които ръчно да се обновяват DOM възлите.

Различният подход на React.

Когато даден компонент е първоначално създаден се извиква неговият ***render()*** метод, който генерира представителен изглед. От този представителен изглед се изгражда низ от маркиращият код, който се инжектира в документа. Когато има промяна в данните, рендериращият метод се извиква отново. С цел да се постигне максимална ефективност, стойностите между извикванията на ***render()*** биват сравнявани, за да се генерират минимален брой промени по документа. Данните, които се връщат от рендериращият метод не са нито низове, нито нито DOM възли - те са леки описания на това как трябва да изглежда документа. Този процес, създателите на React, наричат "спогодба" (от англ. ***reconciliation***). Тъй като този процес на презареждане и толкова бърз (около 1милисекунда за тривиални задачи), не се налага изрично уточняване на връзки с данните. Разработчиците на React намират този подход за по-лесен, когато става дума за създаване на приложения.

HTML е само началото.

Тъй като React притежава своя олекотена версия на документа, с него могат да бъдат създадени някои наистина интересни неща:

- Facebook притежава динамични чартове, които се рендерират директно в **<*canvas*>** вместо HTML.
- Instagram е приложение на единична страница (single page), създадено изцяло с React и Backbone (олекотена библиотека работеща по парадигмата ***модел-изглед-презентатор*** - ***MVP***). Дизайнерите му редовно допринасят React кода с JSX.
- Създадени са вътрешни прототипи, които задвижват React приложенията в уеб работник и използват React за работа в естествена ***iOS*** среда чрез Objective-C мост.

- React може да бъде предварително, сървърно рендериран с цел оптимизация за интернет търсачки (***SEO - Search Engine Optimization***), производителност, споделяне на код и цялостна гъвкавост.
- Събитията се държат консистентно и съвместимо по стандартите с всички браузъри (включително IE8), и се използва автоматично делегиране.

2.7 Редактор за сорс код: Visual Studio Code

Visual Studio Code (15) комбинира простотата сорс код редакторите с мощни инструменти за разработка като IntelliSense автоматично довършване при писане, проследяване и отстраняване на грешки. Първото и основно негово качество на този редактор е, че не се “пречка”. Задоволително плавния цикъл на редакция-компилация-проседяване на грешки означава по-малко време, прекарано в настройване на средата и повече време за изпълнение на идеи.

Наличен е за macOS, Linux, and Windows.

Visual Studio Code поддържа macOS, Linux, and Windows - за да може да бъде максимално достъпен без операционната система да е пречка.

Лекота при редакция, компилация и проследяване на грешки.

В своето ядро, Visual Studio Code включва светкавично бърз сорс код редактор, идеален за ежедневна употреба. С поддръжка на стотици програмни езици, VS Code помага на продуктивността със синтактично подчертаване, автоматично наместване на скоби, автоматично подравняване на кода, блоково избиране,

кодови шаблони, и много др. Интуитивни кратки, клавиатурни команди, лесна персонализация и клавиатурни команди, допринесени от страна на общността, помагат за лесната навигацията в кода. За сериозна работа свързана с код, нуждата от инструменти, които разбират кода, а не само блокове текст, става и по-голяма. Visual Studio Code притежава интегрирана поддръжка за IntelliSense кодово довършване, богато семантично разбиране на кода и навигацията както и кодовата подредба. Когато става въпрос за сериозен код, идва и нуждата от сериозен инструмент за откриване на грешките. Това е функционалност, която липсва на разработчиците най-много, при гладко протичане на процеса по изграждане на софтуер. Именно заради това хората стоящи зад Visual Studio Code добавят интерактивен ***debugger***, който позволява постъпкова проверка на кода, инспекция на променливи, преглед на стека на извиванията и възможност за изпълнение на команди директно в конзолата. VS Code също се интегрира с инструменти за компилиране и скриптиращи инструменти, автоматизиращи общи задачи от ежедневния работен процес. VS Code има и вградена интеграция за ***Git***, с която работата с този вид сорс контрол системи се осъществява без да се налага излизането от редактора и възможностите му се простират дори за сравнение на чакащите одобрение разлики.

Отворен за модификация.

Всяка особеност на редактора може да бъде персонализирана според нуждите с множество допълнения, създадени от трети страни. Докато повечето сценарии работят “директно от кутията” без допълнителни конфигурации, VS Code расте с този, който го използва и насърчава персоналната оптимизация, така че да се постигне максимално задоволително преживяване. VS Code е проект с отворен код, към който всеки би могъл да стане сътрудник за разширяването на изпълнената с живот общност в ***GitHub.***

Създаден с отношение към уеб програмирането.

VS Code включва обогатена и вградена поддръжка за Node.js разработка с JavaScript и **TypeScript**, захранена от същите съществени технологии, задвижващи и пълноценното Visual Studio. VS Code поддържа и страхотни инструменти за работа с други уеб-базирани технологии като JSX/React, HTML, CSS, Less, Sass, и JSON.

Здрава и разширима архитектура

Архитектурно погледнато, Visual Studio Code комбинира най-доброто от уеб, естествени за средата и специфични за конкретен език технологии. Използвайки ***Electron*** (библиотека с отворен код за създаване на desktop приложение чрез похвати заимствани от уеб програмирането), VS Code комбинира технологии като JavaScript и Node.js със скоростта и гъвкавостта на естествени приложения.. VS Code използва нова, по-бърза версия на мощтния HTML-базиран редактор, който задвижва облачния редактор “Monaco”, F12 инструментите на Internet Explorer, и други проекти. Допълнително VS Code използва архитектура за инструментни услуги, давайки възможност за интегрирането с много от същите технологии, които задвижват Visual Studio, като Roslyn за .NET, TypeScript, двигателя за откриване на грешки на Visual Studio и др.

Visual Studio Code включва публичен модел за допълнения, който позволява на разработчиците та създават и използват разширение с цел обогатяване и персонализиране на работния процес.

Перфектната интеграция с технологиите изброени до този момент, правят Visual Studio Code незаменим помощник, който е способен да обхване целия процес на разработка както на сървърни така и на уеб приложения едновременно. Всички отрязъци от код, които участват в представянето на приложенията към настоящата работа са взети директно от редактора.

2.8 Система за контрол на версиите: Github

(2) За финал на главата представяща избраните технологии за разработката на приложенията, неслучайно е избрано представянето на системите за контрол на версиите и по-точно една от тях - ***Git*** предоставена от ***GitHub.*** Сорс контрол системите са може би един от основните инструменти за разработка на софтуер, защото предоставят две концепции от изключителна важност - съхранението и версионирането на сорс кода. Особено когато става дума за големи проекти и множество екипи от разработчици е немислимо да не се използва някакъв вид система за сорс контрол. Тези системи осигуряват правилния интегритет при работа с една и съща кодова база и помагат за качествена, лишена от грешки колаборация.

И така, какво е Git?

Git е система за контрол на версиите. Подобна система представлява софтуер, създаден да следи промените, които се извършват по файловете с времето. По точно, Git е разпределена система за контрол на версиите, което означава, че всеки работещ по проекта разполага със свое копие на пълната история, а не само на последното състояние на файловете.

Какво е GitHub?

GitHub е уеб базирана система където всеки може да споделя копие на своето хранилище. Тя помага за много по-лесно колаборация между хора работещи по общи проекти. Това се извършва чрез осигуряването на централизирана локация за споделени хранилища, уеб-базиран интерфейс предлагащ изглед, способност създаване на разклонения, заявки за следене на хранилището, публикуване на проблеми и разширения, wiki документи позволяващи по-

ефективни спецификация, преглеждания и дискусии по промените между членовете на тима.

Защо е необходим Git?

Дори разработка при самостоятелна работа при обработка на текстови файлове, съществуват множество облаги от използването на Git. Тези облаги включват следното:

- Способността за връщане на промените - Ако бъде направена грешка, винаги може да се върне работата към предишна точка във времето, за да се възстанови изгубеното.

- Пълна история на всички промени.

- При желание да бъде направена проверка как е изглеждал проекта преди ден, седмица, месец или година винаги може да бъде свалена версия, която да отговаря на тогавашното състояние.

- Документация обясняваща, защо промените са направени - Често е трудно да се запомни, защо дадена промяна е направена. Чрез съобщенията при предаване за съхранение, става лесно да се добави документация за бъдеща препратка.

- Увереност при правенето на промени - Лекотата при връщане към предишна версия на проекта, дават необходимата увереност при експериментирането с нови промени.

- Множество потоци при водене на история - Създаването на отделни клонове в хранилището, дава възможност за експериментиране с различни промени, отделна разработка на функционалност или запазване на различни версии. Тези нови разработки могат да бъдат слети в

последствие към основния проект (така наречения главен клон) или да бъдат унищожени изцяло.

При работа в екип позитивите са още повече. Някои от тези позитиви са:

- Възможността за разрешаване на конфликти - С Git много хора могат работят върху един и същ файл, по едно и също време. Обикновено Git успява да слее промените автоматично. Когато това не се случи, конфликтите биват маркирани, за да могат да бъдат разрешени.
- Независим поток на историята - Отделните хора работещи по проекта могат да работят по отделни клонове като по този начин се разделят функционалностите отделно от основния поток, а по-късно могат да бъдат слети когато са готови.

Защо GitHub?

GitHub е много повече от място за съхранение на ***Git*** хранилища. Той дава много допълнително функционалности:

- Документация за изискванията - С използването на ***Issues***, могат да бъдат документирани намерени грешки или да бъдат описвани допълнителни функционалности.

- Колаборация чрез отделни потоци на историята - чрез използването на клонове и така наречените ***pull requests***, всеки може вземе участие в различни клонове или разработки.

- Следене конкретен прогрес - С разглеждане на листа от заявки, може да се следят всички отделни разработки, над които се работи, а с натискане на конкретна заявка могат да бъдат разгледани най-новите промени както и дискусиите свързани с тях.

- Следене на цялостния прогрес - С бърз поглед върху диаграмата на пулсация, бързо може да се придобие представа над какво работи целия екип в момента.

GitHub е чудесна система представяща както безплатни акаунти така и абонаменти за фирми и компании. Сорс кода към настоящите проекти се съхранява именно там на следните адреси:

https://github.com/Martin-Kotrulev/unibit-school-quiz - за сървърното приложение.

https://github.com/Martin-Kotrulev/unibit-quiz-react-frontend - за уеб приложението.

III. ПРАКТИЧЕСКА РАЗРАБОТКА НА УЧЕБЕН СОФТУЕР

3.1 Структура на базата данни

В тази глава ще бъде разгледана структурата на базата данни, как си взаимодействат таблиците на основните модели и ще бъде описано тяхното участие.

3.1.1 Описание на основните таблици

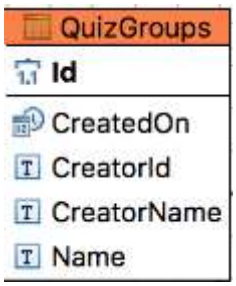
Важно е да бъде направена една предварителна уговорка. Таблиците, които са част от базата данни ***unibit_quiz*** не са създадени ръчно. Те са генерирани от ***Npsql*** дизайнера и ***Entity Framework Core*** чрез така наречения подход ***Code-First***, при който таблиците се създават на база съществуващи модели в сървърното приложение. Това е много удобна техника, която заедно със системата за миграции, която идва с Entity Framework Core, прави промените по структурата надеждни и бързи. Следва представяне на най-основните таблици:

- Първата основна таблица, която ще бъде представена се нарича ***QuizGroups*** **(***фиг. 2***)**. Тя отговаря за съхранението на групите в базата данни и отговаря на класа QuizGroup в сървърното приложени. Нейните колони са както следва:
    - ***CreatedOn*** - отбелязва дата и час на създаване на групата.
    - ***CreateorId*** - външен ключ към създателя на групата.
    - ***CreatorName*** - име на създателя. Използва се за визуализация когато групите са изброени в лист.
    - ***Name*** - име на самата група.

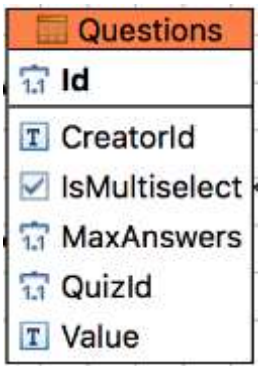
- Веднага след групите в йерархията се нарежда тестовете с таблицата ***Quizzes*** (фиг. 1), и както следва от името съхранява всички тестове. Тук основните колони се дублират с ***QuizGroups*** за това следват само новите:
    - ***Ends*** - посочва крайна дата и час на теста.

- ***GroupId*** - външен ключ към групата на която принадлежи, незадължително.
- ***Locked*** - посочва дали теста е заключен с парола.
- ***Once*** - посова дали теста ще бъде проведен само веднъж.
- ***Password*** - парола за теста ако е заключен
- ***Published*** - посочва дали теста е публикуван. Непубликуваните тестове не могат да се достъпват. А публикуваните не могат да бъдат редактирани.
- ***Ends*** - посочва датата и часа на започване на теста.

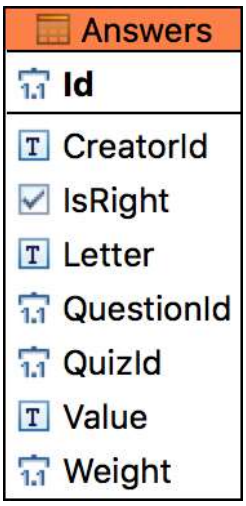
Фиг. 1 Таблица Quizzes



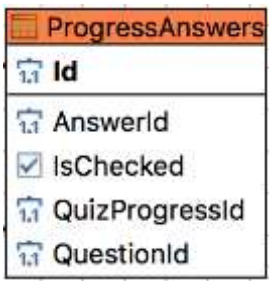
Фиг. 2 Таблица QuizGroups



Фиг. 3 Таблица Questions



Фиг. 4 Таблица Answers



Фиг. 5 Таблица ProgressAnswer

- След тестовете идват самите въпроси, които се пазят в таблицата ***Questions*** (фиг. 3). Тук отново има връзка към създателя им и допълнителен външен ключ към теста:
  - ***IsMultiselect*** - определя дали въпроса може да е само с един верен отговор или с няколко.

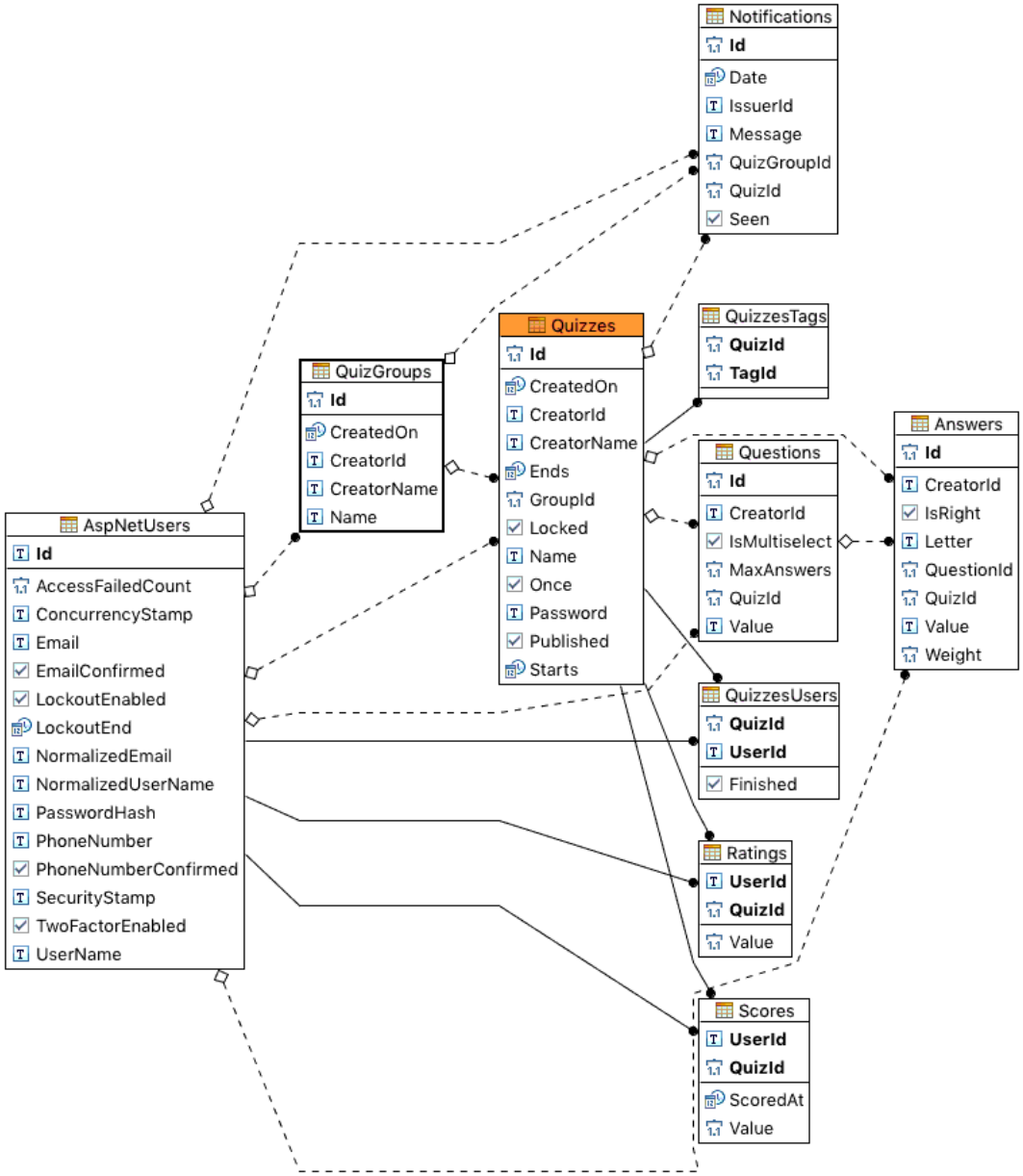
- ***MaxAnswers*** - ограничител на максималната бройка на позволени отговори.
- ***Value*** - текста на самия въпрос.

- Таблицата за отговорите на всеки въпрос (фиг. 4) притежава външни ключове към своя създател, въпрос и тест заедно с допълнителните:
  - ***IsRight*** - колона посочваща дали това е верен отговор. Манипулира се само от създателя.
  - ***Letter*** - малка буква служеща за табелка на отговора. Автоматично генерирани и пренареждани от програмната логика.
  - ***Value*** - текста на самия отговор.
  - ***Weight*** - тежест позволяваща различни отговори да бъдат третирани с различна важност. В диапазона от 1 до 10.

- Последната важна таблица, която си заслужава да се спомене е ***ProgressAnswer*** (фиг. 5)**.** Тя служи за отбелязване на прогреса по даден въпрос и се пише в нея всеки път когато потребителя, правещ теста, избере отговор:
  - ***AnswerId*** - външен ключ сочещ към същинския въпрос.
  - ***IsChecked*** - флаг показващ дали въпроса е избран за верен.
  - ***QuizProgressId*** - външен ключ към таблицата с прогреси.
  - ***QuestionId*** - външен ключ към въпроса към който принадлежи отговора.

3.1.2 Релационни връзки между основните таблици

В тази част ще бъдат описани някои от основните релации с цел да бъде дадена повече яснота относно работата с базата.

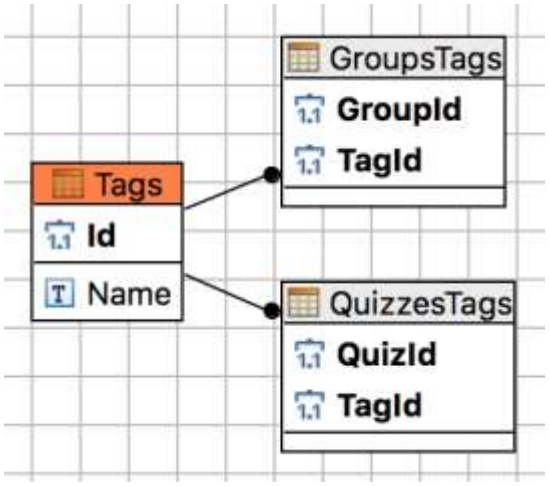


Фиг. 6 Цялостна диаграма на релациите в базата

На фиг. 6 може да се види цялостната диаграма на релациите в базата. Умишлено някои от таблиците бяха пропуснати в предишната част, за да може да им се обърне внимание в контекста на цялостната логика. Както може да се види от диаграмата в базата фигурират таблиците ***Notifications*** и ***Ratings***. Тези таблици нямат отношение към работа на приложението, но с добавени по време

на дизайна с цел за евентуално доразвиване на приложението. Тяхната цел е да може да се получават уведомителни съобщения в случай на събития свързани с тестовете (в случая на Notifications) и начин потребителите да дават рейтинг на тестовете.

Основната логика на базата започва от потребителите. Те са представени от таблицата ***AspNetUsers***, която идва на готово от EntityFramwork. Всеки потребител може да създава ***QuizGroups*** (Групи), които се пазят с връзка едно към много и ***Quizzes*** (Тестове), които отново са едно към много. Тестовете могат да принадлежат към даден група, но могат да съществуват и самостоятелно. Тази свобода е дадена с цел улеснение за потребителя. Следователно съществува и връзка между групите и тестовете, която също е едно към много. И тестовете и групите могат да имат тагове. Уникалните таговете се пазят по само веднъж в базата, за да бъде избегнато безкрайно нарастване, а групите и тестовете се свързват към таговете с междинни таблици и връзки много към много (фиг. 7). Таблицата ***QuizzesUsers*** е междинна и служи

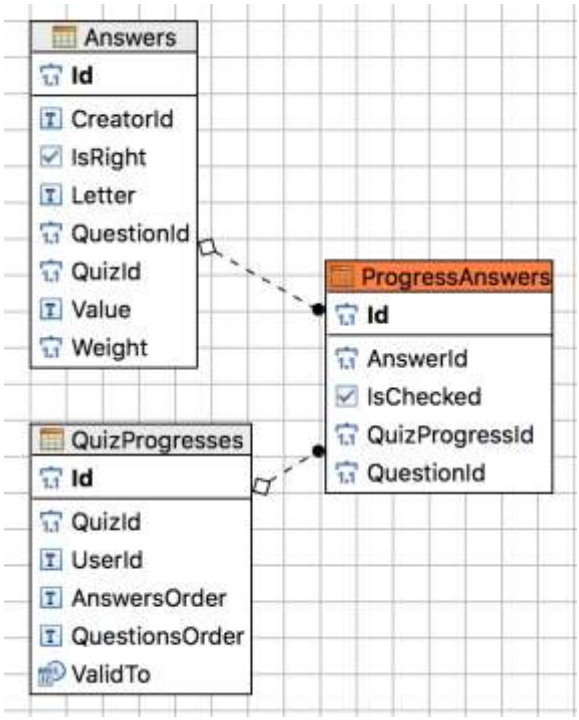


за отбелязване дали даден потребител е взел участие в теста.

Фиг. 7 Междинните таблици са връзка с таговете

Всеки **Question** (Въпрос), от своя страна принадлежи към даден тест, а всеки - ***Answer*** (Отговор) към - въпрос.

Интересен е начина по който се пази прогреса по даден тест, за да може потребителя да излиза и след това да може да продължава с теста до където е стигнал (фиг. 8).



Фиг. 8 Таблицата QuizProgress и пазеща прогреса на потребителя по даден тест

Когато потребителя отвори даден тест, идеята е да може да му бъде генериран прогресен обект, който да пази информация за реда на въпросите (за да може теста да изглежда при всяко отваряне както първия път), а след това към всеки прогресен обект се добавят прогресни отговори. Когато потребителя е готов да завърши теста се минава по отговорите от прогреса, които се сравняват с оригиналните дали ***IsChecked*** отговаря на ***IsRight*** и ако да, се взима тежестта за пресмятане на крайния резултат. Повече за това в следващата глава.

В тази глава беше направен разбор на основните таблици и взаимоотношенията в базата данни между основните обекти.

3.2 Архитектура на сървърната част

3.2.1 REST API и проектна структура

За сървърната услуга е използван подхода за изграждане на REST (Representational state transfer) API. Това накратко е методология на комуникация и пренос на данни, при която вместо цели статични изгледи се пренасят ресурси (най-често под формата на JSON или XML). Контролерите дефинират ***URI*** (***Uniform Resource Identifiers***) крайни точки под формата на унифициран интерфейс , с който клиента да може да прави заявки с основните методи от HTTP (като GET, POST, DELETE), за да извлича или променя данни. По късно в частта за REST контролерите ще бъде описан и самия интерфейс за комуникация. За да може уеб услугата да връща унифициран отговор към клиента е създаден унифициран обект представящ самия отговор. Той е достатъчно гъвкав и обхваща всички случаи, които могат да възникнат при отговор. ***Success*** е флаг, който се грижи за статуса на върнатия резултат. ***Message*** е за изпращане на съобщения. ***Result*** е резултатния обект на заявката, а ***Errors*** масив съдържащ всички грешки. Както се вижда на фигурата, обекта пропуска всички полета които са с нулева стойност, за да не бъдат изпращани под формата на JSON.

```
10  public class ApiResponse
11  {
12      public bool Success { get; set; }
13
14      [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
15      public string Message { get; set; }
16
17      [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
18      public object Result { get; set; }
19
20      [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
21      public IEnumerable<string> Errors { get; set; }
22
23      public ApiResponse()
24      {
25      }
26
```

Фиг. 9 Унифициран обект за отговор от уеб услугата

Що се отнася до структурата на сървърното приложение, то е разделено на четири основни проекта:

- ***Uniquizbit.Web*** проекта съдържа всички файлове свързани с уеб услугата като конфигурации, контролери, ресурсни модели и т.н. От тук се стартира и сървъра.

- ***Uniquizbit.Common*** - съдържа полезни общи файлове като енумерации и общи модели използващи се на много места.

- ***Uniquizbit.Services*** носи със себе си всички сервизни интерфейси и имплементации.

- ***Uniquizbit.Data*** **-** съдържа всички модели и референции към Entity Framework Core. От тук се контролират миграциите и операциите за обновяване на базата.

3.2.2 LINQ

(7) Индустрията е достигнала стабилна точка в своята еволюция на обектно ориентираното програмиране и технологиите свързани с него. Програмистите приемат за даденост функционалности като класове, обекти и методи. Обръщайки се към сегашните и бъдещи технологии, става ясно, че следващото голямо предизвикателство в компютърните технологии е намаляването на комплексността и достъпа до интегрирана информация, която не е естествена, използвайки сегашните обектно ориентирани похвати. Двата най-често срещани източници на не-обектно-ориентирана информация са релационните бази данни и езика XML.

Вместо добавянето на релационни похвати или специфични за XML техники към програмните се езици Microsoft поема по генерален подход чрез LINQ проекта, с който добавят обобщаващи техники за заявки към .NET Framework, които се отнасят за всякакъв вид информация, а не само към релационна или XML-базирана. Този подход се нарича .NET Езиково-Интегрираните-Заявки (.NET Language-Integrated Query LINQ).

Термина езиково-интегрирани заявки индикира, че заявките интегрирана функционалност директно в основния език, с който работи

разработчика (напр. Visual C#, Visual Basic). Езиково-интегрираните заявки позволяват писането на изрази, които могат да се възползват от богати мета-данни, проверка на синтаксиса по време на компилация, статично писане и IntelliSense, които до преди това са възможни само в императивния код. Езиково-интегрираните заявки притежават също обобщено, декларативно умение на заявките да бъдат изпълнени директно върху информация в паметта, а не само от външни източници.

.NET Езиково-интегрираните заявки дефинират множество от стандартни оператори за заявка, които позволяват прекосяването, филтрирането и възможността за изпълнението на проектни конструкции по директен и същевременно декларативен начин във всеки .NET-базиран програмен език. Стандартните оператори за заявки позволяват изпълнението на заявки върху всеки IEnumerable<T>-базиран източник на информация. LINQ позволява на трети страни да допълват дефинираните, стандартни оператори с нови, специфични за средата оператори, които от своя страна са подходящи да таргетират конкретната технологична среда. По-важното е, че трети страни са свободни да заменят стандартните оператори с техни собствени имплементации, които да подобрят средата с допълнителни сървиси като отдалечена оценка, превод на заявките, оптимизации и др. Поддържайки конвенциите на LINQ шаблона, тези имплементации се възползват от същата интеграция в езика и поддръжка от същите инструментите поддържащи стандарта.

Лесния начин за разширяване на архитектурата за заявки е използвана и в проекта LINQ, за да поддържа имплементации, които работят едновременно с XML и SQL данни. Операторите за заявки за XML (LINQ към XML) използват ефикасен, лесен за ползване, директно-в-паметта XML способност за доставка на XPath/XQuery функционалност в основния програмен език. Операторите за заявка действащи с релационни данни (LINQ към SQL) надграждат интеграцията с SQL-базирани, схематични дефиниции в типовата система на общата езикова среда (CLR). Тази интеграция дава силно типизиране над релационните данни, докато същевременно запазва

изразителната мощ на релационния модел и производителността от изпълнението на заявките директно в хранилището.

От всичко казано до този момент става ясно, че LINQ предоставя един от най-добрите начини за работа с данни.

3.2.3 Entity Framework Core

(8) Entity Framework (EF) Core е лека, разширяема и междуплатформена версия на популярната технология за достъп до данни на Entity Framework.

EF Core е обектно-ориентиран mapper (ORM), който позволява на .NET разработчиците да работят с база данни, използвайки .NET обекти. Това премахва необходимостта от по-голямата част от кода за достъп до данни, който разработчиците обикновено трябва да напишат. Entity Framework се допълва идеално от LINQ, като дефинира и допълнителни верижни методи за работа.

Какво е новото във версия 2.0?
Разделяне на таблици - Вече е възможно да се съберат два или повече типа обекти в една и съща таблица, където първичните ключове ще бъдат споделени и всеки ред ще съответства на два или повече обекти.

Споделени таблици
За да използваме разделянето на таблици, идентифициращата връзка (където чуждите ключове са от първичния ключ) трябва да бъде конфигурирана между всички типове обекти, споделящи таблицата.

Притежателни типове
Типът на притежаван обект в релационната връзка може да има същия тип от CLR с друг притежаван, но тъй като типа не може да бъде идентифициран само

от CLR, трябва да има навигация към него от друг тип обект. Елементът, съдържащ дефиниращата навигация, е собственикът. При заявка на собственика собствените типове ще бъдат включени по подразбиране.

По конвенция ще бъде създаден сенчест първичен ключ за собствения тип и ще бъде преобразуван в същата таблица като собственика чрез разделяне на таблицата.

Филтри за заявки на ниво модел

EF Core 2.0 включва нова функция, която е наричена филтри за заявки на ниво модел. Тази функция позволява предикати на LINQ за заявки (булеви изрази, обикновено предавани на оператора на заявката), да бъдат дефинирани директно върху Entity типовете в моделирането на метаданните (обикновено в OnModelCreating). Такива филтри се прилагат автоматично към всички LINQ заявки, включващи тези Entity типове, включително такива, към които се свързани индиректно, например чрез използване на Include или директни навигационни свойства. Някои често срещани приложения на тази функция са:

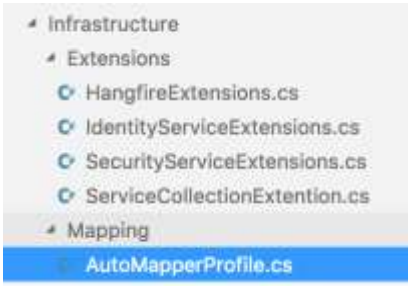
- Мекото изтриване - типа определя IsDeleted свойство.
- Многостранно наемане - типа на обекта определя свойството TenantId.

Висока производителност

- Обединяване на DbContext - Основният модел за използване на EF Core обикновено включва регистриране на персонализиран тип DbContext в системата за инжектиране на зависимости и по-късно получаване на копия на този тип чрез параметри в контролерните конструктори. Това означава, че за всяка заявка се създава ново копие на DbContext. Във версия 2.0 е въведен нов начин за регистриране на DbContext в механизма на инжектиране, която прозрачно въвежда набор от преизползваеми DbContext инстанции. За да използвате обединяването на DbContext, използвайте AddDbContextPool вместо AddDbContext по време на регистрацията на услугата.

3.2.4 Конфигурационни файлове

Основните конфигурационни файлове се изразяват в класове с ***extension*** методи, които отделят логиката по настройване на конфигурацията, за да не замърсяват ***Configure*** метода. На фиг. 10 са показани основните конфигурационни файлове. ***HangfireExtensions*** съдържа методи за конфигуриране на Hangfire задачите. ***IdentityServiceExtensions*** настройват библиотеката ***Identity*** която генерира таблиците за сигурност, ***SecurityServiceExtensions*** са допълнителни настройки за сигурност, конфигуриращи приложението да използва ***token*** за верификация на самоличността и ***ServiceCollectionExtension***, който притежава метод за сканиране на асемблито (чрез ***reflection*** фиг. 11***)*** за съществуващи сървиси и автоматично ги регистрира. ***AutoMapperProfile*** съдържа конфигурацията на библиотеката ***Automapper***, която улеснява превръщането на базови модели към ресурсни. На фигури 12 и 13 са демонстрирани и конфигурациите за средата.



Фиг. 10 Извадка от проектната структура на Uniquizbit.Web

```
10      public static IServiceCollection AddDomainServices(
11          this IServiceCollection services)
12      {
13        Assembly
14            .GetAssembly(typeof(IService))
15            .GetTypes()
16            .Where(t => t.IsClass && t.GetInterfaces().Any(i => i.Name == $"I{t.Name}"))
17            .Select(t => new
18            {
19              Interface = t.GetInterface($"I{t.Name}"),
20              Implementation = t
21            })
22            .ToList()
23            .ForEach(s => services.AddTransient(s.Interface, s.Implementation));
24
25        return services;
26      }
27    }
28  }
```

Фиг. 11 Сканираща функционалност за автоматично намиране и регистриране на сървиси

```
82      // Configure cross origin resource sharing
83      app.UseCors(
84        options => options.WithOrigins("http://localhost:3000")
85          .WithHeaders("authorization", "accept", "content-type", "origin", "x-custom-header")
86          .AllowAnyMethod()
87      );
88
```

Фиг. 12 Конфигуриране на CORS, за да може да се достъпва сървърното приложени е от външен клиент. Важно е да се отбележи, че това е сървърна конфигурация и работи само с сървъра Kestrel, който идва с .NET Core

```
{
  "ConnectionStrings": {
    "Default": "server=localhost; database=unibit-quiz; user id=postgres; password=postgres;"
  },
  "Logging": {-
  },
  "JWTSettings": {
    "SecretKey": "api-secret-key!@#$%",
    "Issuer": "Uniquizbit",
    "Audience": "Uniquizbit",
    "ExpirationDays": 5
  },
  "GradesSettings": {
    "Excellent": 100,
    "VeryGood": 90,
    "Good": 80,
    "Average": 70,
    "Weak": 60
  }
}
```

Фиг. 13 Основен конфигурационен файл за средата - appsettings.json. Могат да бъдат създавани множество такива файлове за отделните среди на работа. Тук се помещават и настройките за JWT, които се използват за генерирането на token за сигурност както и настройките за процентите използвани при оценяването на тестовете. Този подход за конфигурация дава предимство и гъвкавост при промени.

3.2.5 Automapper

Automapper е много полезна библиотека спестяващо ценното време на разработчика. С негова помощ превръщането на моделите от базата данни във олекотени ресурсни модели за пренос на данните, става много лесно. Единственото, което е необходимо за да може Automapper коректно е създаването на профилен клас, който да наследява ***Profile*** класа идващ с библиотеката. Този конфигурационен файл служи са дефиниране на точните правила по които да стане превръщането от ресурсен модел към основен и от основен към ресурсен. На фиг. 14 е даден примерен отрязък от профила използван в сървърното приложение. Най-големия плюс е, че за тривиалните преобразувания не необходимо само посочването на посоката. Ако са

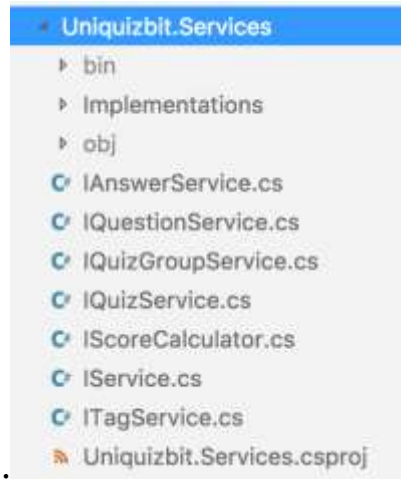
необходими по специални конфигурации съществуват много допълнителни методи за отделните свойства на моделите.

```
10  public class AutoMapperProfile : Profile
11  {
12    public AutoMapperProfile()
13    {
14      // Domain to resource
15      CreateMap<Answer, AnswerResource>();
16
17      CreateMap<QuizGroup, QuizGroupResource>()
18        .ForMember(qg => qg.Tags, opt =>
19          opt.MapFrom(qg =>
20            qg.Tags.Select(t => t.Tag.Name)));
21
22      CreateMap<QuizGroup, IdNamePairResource>();
23
24      CreateMap<Quiz, QuizResource>()
25        .ForMember(q => q.Password, opt => opt.Ignore())
26        .ForMember(q => q.Scores, opt => opt.Ignore())
27        .ForMember(q => q.Participants, opt => opt.Ignore())
28        .ForMember(q => q.Tags, opt =>
29          opt.MapFrom(q =>
30            q.Tags.Select(t => t.Tag.Name)));
31
```

Фиг. 14 Извадка от конфигурационния клас на Automapper

3.2.6 Сервизен слой за работната логика

За да се избегне директно достъпването на контекста при заявки към базата чрез Entity Framework Core е добавен един допълнителен сервизен слой. Целта на този слой е да абстрахира логиката за работа с базата данни като предостави ясен интерфейс за работа с набор от логически свързани методи вместо конкретна имплементация. Вградената поддръжка на ***DI (Dependency Injection)*** прави този подход идеален за целите на приложените. .NET Core се грижи за инстанцирането и “инжектирането” на сървисите където са необходими като за целта работи с абстрактни интерфейси, които се заменят конкретните си имплементации в момента на инстанциране в конструктора. Тази техника носи много позитиви вариращи от добър контрол на паметта, сигурност на обхвата и възможност за тестване. На фиг. 15 е показан лист със всички сървиси, които са налични за сървърното приложение



Фиг. 15 Извадка от Uniquizibt.Service проекта съдържащ всички

Следва кратко описание на услугите предоставени от наличните сървиси:

- AnswerService - в него са обобщени методите за манипулация на отделните въпроси. Дава свобода при подхода за манипулация.

- QuestionService - занимава се манипулацията на отделните въпроси, проверява за собственост на въпроса и притежава мощен обобщен метод, с който могат да се модифицират всички въпроси на тест едновременно.

- QuizGroupService - метод за манипулация и контрол на групите.

- QuizService - както подсказва името му, предлага услуги свързани с контрола над тестовете.

- ScoreCalculator - предлага методите за оценяване на представянето в тестовете.

- IScoreService - единствено помощен интерфейс за автоматичното сканиране в асемблито на всички сървиси. Всички сървиси трябва да го наследят, за да работи коректно сканирането.

- TagService - предлага методи за работа с таговете.

На фиг. 16 е представен и един от по-интересните методи в TagService, грижи се да провери съществуват ли таговете подадени му в списъка ***tags*** и да добави всички тагове, които не съществуват в базата. Важно е да се отбележи, че за такова търсене е важно да бъдат дефинирани индекси в базата, за да може да се оптимизира заявката. Подобна таблица би нараснала доста бързо.

```
19      public async Task<IEnumerable<Tag>> UpdateTagsAsync(ICollection<string> tags)
20      {
21          var existingTags = await _dbContext.Tags
22              .Where(t => tags.Contains(t.Name))
23              .ToListAsync();
24
25          var newTags = tags
26              .Where(t => !existingTags.Exists(tag => tag.Name == t))
27              .Select(t => new Tag { Name = t });
28
29          await _dbContext.Tags.AddRangeAsync(newTags);
30          await _dbContext.SaveChangesAsync();
31
32          existingTags = await _dbContext.Tags
33              .Where(t => tags.Contains(t.Name))
34              .ToListAsync();
35
36          return existingTags;
37      }
38  }
```

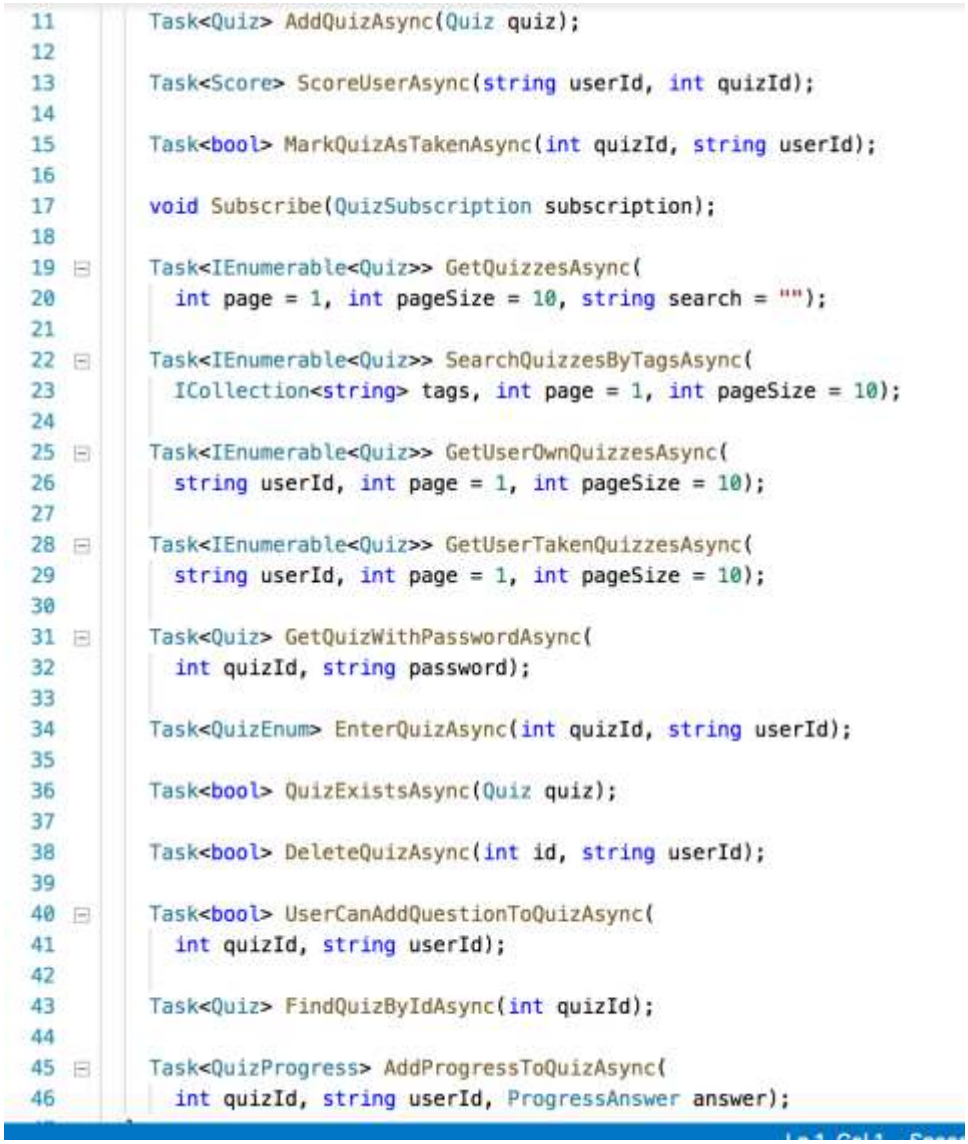
Фиг. 16 Метод за проверка и запис на нови тагове в базата

Един от основните сървиси, на който си заслужава да се обърне по обширно внимание е QuizService (фиг. 17). Той предоставя най-много методи и съответно най-много централна функционалност. Прави впечатление, че повечето методи са асинхронни, защото се възползват от предоставените, асинхронни методи за достъп до базата от EntityFramework По-интересните методи на интерфейса са:

- ScoreUserAsync - използва се за генериране на оценка на потребителя като се прави сравнение между неговите прогресни отговори и тези, създадени от автора.

- MarkQuizAsTaken - функционалност маркираща статуса на участие на потребителя в даден тест

- EnterQuizAsync - маркиращ метод за контрол на участниците в тестовете.
- AddProgressToQuizAsync - ръчен метод за добавяне на прогрес. Заменим от общия и много по мощен, намиращ се в QuestionService.



```
11    Task<Quiz> AddQuizAsync(Quiz quiz);
12
13    Task<Score> ScoreUserAsync(string userId, int quizId);
14
15    Task<bool> MarkQuizAsTakenAsync(int quizId, string userId);
16
17    void Subscribe(QuizSubscription subscription);
18
19    Task<IEnumerable<Quiz>> GetQuizzesAsync(
20      int page = 1, int pageSize = 10, string search = "");
21
22    Task<IEnumerable<Quiz>> SearchQuizzesByTagsAsync(
23      ICollection<string> tags, int page = 1, int pageSize = 10);
24
25    Task<IEnumerable<Quiz>> GetUserOwnQuizzesAsync(
26      string userId, int page = 1, int pageSize = 10);
27
28    Task<IEnumerable<Quiz>> GetUserTakenQuizzesAsync(
29      string userId, int page = 1, int pageSize = 10);
30
31    Task<Quiz> GetQuizWithPasswordAsync(
32      int quizId, string password);
33
34    Task<QuizEnum> EnterQuizAsync(int quizId, string userId);
35
36    Task<bool> QuizExistsAsync(Quiz quiz);
37
38    Task<bool> DeleteQuizAsync(int id, string userId);
39
40    Task<bool> UserCanAddQuestionToQuizAsync(
41      int quizId, string userId);
42
43    Task<Quiz> FindQuizByIdAsync(int quizId);
44
45    Task<QuizProgress> AddProgressToQuizAsync(
46      int quizId, string userId, ProgressAnswer answer);
```

Фиг. 17 Интерфейсът на QuizService

Следващите няколко фигури ще опишат един от процесите, който до сега беше само споменаван. Това е процесът по запазване на прогреса в тестовете. Това е не само интересен процес, но и може би един от най-важните в приложението, защото дава свобода и гъвкавост при постигнат прогрес в теста. Не само, че защитава потребителя от евентуален токов удар и внезапно прекъсване на интернета, но участва в процеса на оценяването. На фиг. 18 е представен целия метод за добавяне на прогрес.

```
218   public async Task<QuizProgress> AddProgressToQuizAsync(
219     int quizId, string userId, ProgressAnswer answer)
220   {
221     var progress = await _dbContext.QuizProgresses
222       .Include(qp => qp.GivenAnswers)
223       .FirstOrDefaultAsync(qp => qp.UserId == userId && qp.QuizId == quizId);
224
225     if (progress != null)
226     {
227       var progressAnswer = progress.GivenAnswers
228         .FirstOrDefault(ga => ga.AnswerId == answer.AnswerId);
229
230       if (progressAnswer != null)
231       {
232         progressAnswer.IsChecked = answer.IsChecked;
233       }
234       else
235       {
236         var question = await _dbContext.Questions
237           .FindAsync(answer.QuestionId);
238
239         if (!question.IsMultiselect)
240         {
241           var questionAnswersIds = await _dbContext.Entry(question)
242             .Collection(q => q.Answers)
243             .Query()
244             .Select(a => a.Id)
245             .ToListAsync();
246
247           progress.GivenAnswers = progress.GivenAnswers
248             .Where(ga => !questionAnswersIds.Contains(ga.AnswerId))
249             .ToList();
250         }
251
252         progress.GivenAnswers.Add(answer);
253       }
254
255       await _dbContext.SaveChangesAsync();
256     }
```

Фиг. 18 Метод за запазване на досегашния прогрес в теста

Методът приема като аргументи идентификационните номера на потребителя както и тези на теста. След това се проверява в базата има ли такъв прогресен обект и ако да се намират неговият маркиращ отговор, който се взима от ресурса ***ProgressAnswer*** и се променя неговото състояние. Ако не бъде намерен по ключа прогресния отговор, се търси по идентификационния номер на

ресурса дали този отговор съществува за дадения въпрос и на базата на това дали въпроса е с много отговори или не, отговорния ресурс преминава през контролна логика дали да бъде добавен.

Относно генерирането на първоначалния прогрес, то той става в метода ***GetQuestionsForQuizAsync()*** (демонстриран на фиг. 19). Това е мощен метод предоставящ няколко вида функционалности, за да може да се работи с цял тест едновременно.

```
40    public async Task<IEnumerable<Question>> GetQuestionsForQuizAsync(int quizId, string userId)
41    {
42      var isCreator = await _dbContext.Quizzes
43        .FirstOrDefaultAsync(q => q.Id == quizId && q.CreatorId == userId) != null;
44
45      if (isCreator)
46      {
47        return await GetQuestionsWithAnswersForQuizAsync(quizId);
48      }
49      else
50      {
51        if (await UserHasProgressOnQuizAsync(quizId, userId))
52        {
53          return await GetQuestionsFromProgressAsync(quizId, userId);
54        }
55        else
56        {
57          return await GetQuestionsWithNewProgressAsync(quizId, userId);
58        }
59      }
```

Фиг. 19 Метода за извличане на въпросите от даден тест

Как работи метода зависи основно от това дали се извиква от неговия собственик. Ако потребителя е едновременно и създател на тест, извличането на въпросите става праволинейно. Когато се извика от друг потребител обаче става по-интересно. Ако потребителя достъпва въпросите на този тест за пръв път се извиква метода ***GetQuestionsWithNewProgressAsync()***, който има грижата да извлече въпросите в разбъркан ред да запише разбъркания ред в полето за ред на въпросите в прогреса, да разбърка всички отговори на въпросите, да запише и техния ред и накрая да върне отговорите. Следващите пъти, в които се достъпват въпросите, прогресния обект ще се погрижи да бъдат извлечени с правилните подредби и с всички отбелязвания на отговорите. Процеса на извличане на отговорите от прогреса е демонстриран на фиг. 20.

```
147   private async Task<IEnumerable<Question>> GetQuestionsFromProgressAsync(int quizId, string userId)
148   {
149     var progress = await _dbContext.QuizProgresses
150       .Where(qp => qp.QuizId == quizId && qp.UserId == userId)
151       .FirstOrDefaultAsync();
152
153     var questionOrder = GetListFromOrderString(progress.QuestionsOrder);
154     var answersOrder = GetListFromOrderString(progress.AnswersOrder);
155
156     var progressAnswers = await _dbContext.QuizProgresses
157       .Where(qp => qp.QuizId == quizId && qp.UserId == userId)
158       .Include(qp => qp.GivenAnswers)
159       .SelectMany(qp => qp.GivenAnswers)
160       .ToListAsync();
161
162     var questions = await _dbContext.Questions
163       .Where(q => q.QuizId == quizId)
164       .Include(q => q.Answers)
165       .ToListAsync();
166
167     questions = questions
168       .OrderBy(q => questionOrder.IndexOf(q.Id))
169       .ToList();
170
171     foreach (var q in questions)
172     {
173       q.Answers = q.Answers
174         .Select(a =>
175         {
176           progressAnswers.Any(pa =>
177           {
178             var equal = pa.AnswerId == a.Id;
179
180             if (equal)
181               a.IsChecked = pa.IsChecked;
182
183             return equal;
184           });
185           return a;
186         })
187         .OrderBy(a => answersOrder.IndexOf(a.Id))
188         .ToList();
189     }
```

Фиг. 20 Метод за извличане на въпросите за тест от прогресен обект

За финал на тази част е представен и метода за оценяване, който е част от ***ScoreCalculator***. Неговата работа се базира на зададените проценти в конфигурационния файл, за който стана дума по-рано. Базирайки се на процентните граници за 6, 5, 4, и т.н се изчисляват процентите на постигнатия резултат от потребителя спрямо максималния брой точки на теста. Така, за прецизното изчисляване се взима ниската граница на оценката и към нея се добавя нормализираната разлика в процентите за да се постигне най-много разлика от 1.0. Част от процеса се вижда на фиг. 21.

```
public double GetScore(double maxScore, double userScore)
{
  double finalScore = 2.0;
  double scoreInPercentage = Math.Round((((double)userScore / (double)maxScore)) * 100, 2);

  if (scoreInPercentage > _gradesSettings.VeryGood)
  {
    finalScore = 5.0 + Normalize(
      _gradesSettings.Excellent,
      _gradesSettings.VeryGood,
      scoreInPercentage);
  }
  else if (scoreInPercentage > _gradesSettings.Good)
  {
    finalScore = 4.0 + Normalize(
      _gradesSettings.VeryGood,
      _gradesSettings.Good,
      scoreInPercentage);
```

Фиг. 21 Метод за изчисляване на оценката за тест

3.2.7 Описание на REST контролерите

В тази част са описани крайните точки на REST контролерите с кратко описание как и с какви ресурси работят:

POST /api/account/signup - служи са регистрация в приложението и приема RegisterResource

POST /api/account/login - служи за вписване в приложението и приема CredentialsResource

DELETE /api/answers/{answerId} - трие отговор по идентификационен номер

POST /api/groups - добавя група и приема QuizGroupResource

DELETE /api/groups/{groupId} - изтрива група по идентификационен номер

POST /api/groups/{groupId} - добавя тест към група по идентификационен номер и приема QuizResource

GET /api/groups - връща всички групи, приема ***query*** параметър ***search*** който филтрира по имена и тагове в страници по 10

GET /api/groups/{groupId}/quizzes - връща всички тестове на дадена група по идентификационен номер в страници по 10

GET /api/groups/mine - връща всички собствени групи в страници по 10

POST /api/questions/{questionId}/answers - добавя отговор към въпрос по идентификационен номер и приема AnswerResource

DELETE /api/questions/{questionId} - изтрива въпрос по идентификационен

POST /api/quizzes - добавя нов тест и приема QuizResource

POST /api/quizzes/{quizId}/questions - обновява едновременно всички въпроси на даден тест и приема QuestionsResource[]

DELETE /api/quizzes/{quizId} - изтрива теста по идентификационен номер

POST /api/quizzes/{quizId}/enter - използва се за взимане на участие в тест

POST /api/quizzes/{quizId}/questions/{questionId} - добавя прогрес към даден въпрос и приема ProgressAnswerResource

GET /api/quizzes - връща всички тестове, приема ***query*** параметър ***search*** който филтрира по имена и тагове в страници по 10

GET /api/quizzes/{quizId}/questions - връща всички въпроси в даден тест. Ако се достъпва от различен потребител от собственика се записва прогрес, от който в последствие се зареждат въпросите заедно с маркираните отговори. Флага за вярност на отговорите се скрива за всички освен за собственика.

POST /api/quizzes/{quizId}/score - използва се за извличането на оценка от съществуващ прогрес

GET /api/quizzes/mine - извлича всички собствени тестове в страници по 10

В настоящата глава бяха засегнати доста теми свързани с архитектурата и работата на сървърното приложение. Бяха разгледани

конфигурационни, имплементационни и сервизни специфики и беше дадена по-голяма яснота за това как функционират някои от използваните техники.

3.3 Уеб Приложение

3.3.1 NodeJS, npm и Webpack

(3) За изграждането на React приложението помага Node.js, който е базиран на двигателя ***V8*** **-** изключително мощен JavaScript двигател, писан е на C++, разработен от ***The Chromium Project*** и използван ***Google Chrome***. Node.js предоставя удобна среда за разработване и задвижване на full-stack JavaScript приложения. Node е с отворен код и работи под всички основни операционни системи. По отношение на React приложението Node се използва за изпълнението на някои подготвящи пакета скриптове, а неговия пакетен мениджър, ***npm,*** взима участие в инсталацията на необходимите пакети.

(1) Друг основен инструмент е Webpack, който се е превърнал в де факто стандарта за разработка на React приложения. През годините уеб програмирането еволюира от страници, картини и малко JavaScript до пълноценни, големи приложения с тонове комплексен JavaScript и големи дървета от зависими библиотеки и файлове. За да се справи с нарастващата комплексност общността измисля най-различни похвати и практики като:

- Използването на модули в JavaScript, позволяващо разделянето и организирането на програмната логика на отделни файлове.

- JavaScript пред-процесори (които позволяват използването на функционалности, които все още не се поддържат от браузъра или JavaScript версията) и компилирщи-до-JavaScript езици (какъвто е CoffeeScript, например). Но въпреки неизменните им ползи, тези похвати изискват добавянето на допълнителни процеси при разработка - необходимостта да бъдат трансформирани (предени / компилирани) и пакетирани файловете в нещо, което браузъра разбира. Точно тук идват

на помощ инструменти като Webpack. Webpack е модулен инструмент за пакетиране, който е способен да анализира структурата на проекта, да намери JavaScript модулите и да ги пакетира в готов за браузъра вид.

3.3.2 Create React App

(4) Друг особено полезен инструмент, който е използван и в приложението за тестове е ***Create React App***, който е проектиран директно от създателите на React и притежава следните характеристики:

- Единствена зависимост - Нужно е да бъде свален само този инстурмент. Към него са включени Webpack, Babel, ESLint, и други страхотни пакети, но се грижи за уредено и обединено изпълнение, а потребителя получава готова за работа платформа.
- Няма нужда от конфигурации - Потребителя не трябва да конфигурира нищо. Достатъчно надеждна конфигурация за разработъчни и продукционни билдове идва на готово, за да може разработчикът да се съсредоточи в писането на код.

- Няма ограничаване - По всяко време проекта може да бъде “разпънат”, за да може да бъде персонализирана конфигурацията. С изпълнението на една команда, всички конфигурации и зависимости биват преместени директно в проекта, за да може да бъдат контролирани изцяло.

Средата за разработка, която подготвя Create React App притежава всичко необходимо за изграждането на модерно приложение на една страница:

- React, JSX, ES6, и Flow синтактична поддръжка.

- Екстра добавки към ES6.

- Автоматично добавяне на специфичните за всеки браузър имена при CSS свойствата.

- Бърз, интерактивен двигател за ***unit*** тестове с вградена поддръжка за рапорт на покритието.

- Разработъчен сървър в реално време, който подсказва за често срещани грешки.

- Готов скрипт за пакетирането на JS, CSS, и изображенията за продукционна среда.

- Офлайн, сервизен работник и манифест за уеб приложението, който покрива всички критерии за създаването на прогресивни уеб приложения.

- Безпроблемно обновявания на всички инструменти заради единичната зависимост.

Единствения минус на посочения подход е, че всички тези инструменти са предварително конфигурирани да работят по точно определен начин. Ако проектът се нуждае от специфична конфигурация, винаги може да се “разпъне”, но конфигурацията ще трябва да се поддържа ръчно.

3.3.3 Babel

(3) Нужно е да се вметне още една особеност. За да може да се използва JSX (и някои похвати от ES2015) в кода, трябва да бъде инсталиран Babel. Първо трябва да бъдат изяснени проблемите, които той решава. Както беше споменато вече, причината да се използва Babel е, че той поддържа множество способности в JavaScript, които не се поддържат от браузъра. Тези допълнителни функционалности правят кода по-чист за разработчиците, но става така, че браузъра не ги разбира. Решението е скриптовете да бъдат написани с JSX и ES2015, и когато приложението е готово за доставка всички

скриптове биват компилирани в ES5, което е стандартната спецификация поддържана от почти всички браузъри. Babel е популярен JavaScript компилатор, широко приет в React общността. Babel да компилира ES2015 код в ES5 JavaScript, както и JSX в JavaScript функции. Процесът се нарича ***transpilation*** (термин означаващ компилацията на сорс код от един език в сорс код на друг).

3.3.4 Flux

Всяко приложение работи с някакъв набор от динамични данни, които се променят с времето. С нарастване на приложението и появата на отделни компоненти, които взаимодействат едни с други, се появява и нуждата от удобен начин за комуникация и пренос на данните. В случая на приложението за тестове ***UniQuizBit*** потребителите постоянно обработват данни когато създават групи, тестове, и др. Допълнително се обработват търсения по имена и тагове. За необходимостта от унифициран начин за контрол се грижат системи като ***Flux.***

(6) Следват основните концепции на високо ниво и принципите, които трябва да бъдат следвани при изграждането на приложения с Flux.

Flux е разработен от разработчици на Facebook и представлява шаблон, които предоставя начин за контрол на потока от данни в приложението. Най-важната концепция за потока на данни е, че той е едностранен.

Основни части на Flux:

- Dispatcher (Диспечер)
- Store (Хранилище)
- Action (Действие)
- View (Изглед)

Диспечер

Диспечерът приема действия и ги препраща към хранилищата, които са абонирани към него. Всяко хранилище приема всяко действие. За целта трябва да съществува само една единствена инстанция на диспечера за цялото приложение.

Хранилище

Хранилището се управлява директно данните на приложението. Хранилищата се регистрират в диспечера, за да могат да получават действия. Данните в хранилището трябва да се променят само чрез кореспондиращи действия. Не трябва да бъдат предоставени никакви публични методи за промяна, а само такива за извличане. Хранилищата решават на кои действия да отговарят. Всеки път когато данните в хранилището се сменят, то трябва да излъчи събитие за промяна. Трябва да бъдат предоставени отделни хранилища за отделните модели.

Действия

Действията дефинират вътрешния интерфейс на приложението. Те обхващат всички начини, по които нещо може да се случи в приложението. Те представляват обикновени обекти, които притежават поле “тип” и някакви данни. Действията трябва да бъдат семантични и описателни относно случващото се. Те не трябва да описват детайли за имплементацията. Типовете трябва да бъдат описвани с по общи имена като “изтрий-потребителя” вместо “изтрий-потребителя-по-име” или други подобни (описващи процеса), защото всички хранилища получават действията и могат да се възползват от това, да извършат необходимото почистване на данните при подобно “изтрий-потребителя-по-име” събитие.

Изгледи

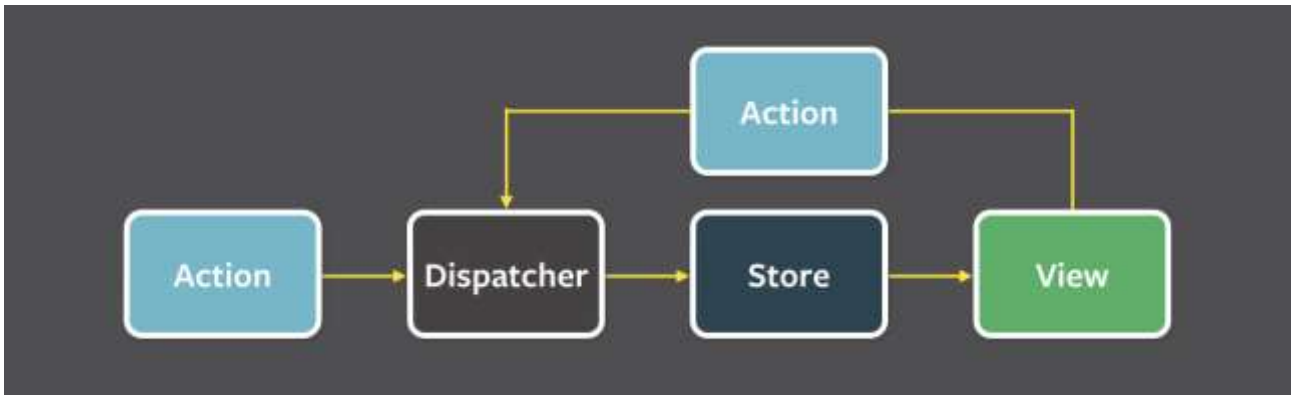
Данните от хранилищата се представят във изгледи. Изгледите могат да използват библиотека по-избор. (В този случай се комбинира идеално с React). Когато изгледа използва данни от хранилището, трябва да се абонира за

събитията на промяна, които идват от хранилището. Когато хранилището излъчи събитие за промяна, изгледа може направи необходимите промени и да се пренареди. Ако компонента използва хранилище без да се абонира за неговите събития, би могло да се прокрадне трудно проследима грешка. Събитията обикновено се изпращат от изгледите, когато потребителите взаимодействат с части от интерфейса.

Поток на данните

Всички елементи изброени до момента могат да бъдат сглобени с диаграма описваща цялостния поток на данните през системата.

1. Изгледите изпращат действия към диспечера.
2. Диспечера изпраща действията до всички хранилища.
3. Хранилищата изпращат данните към изгледите.



Фиг. 22 Диаграма за нагледно представяне на потока от данни

3.3.5 React-Bootstrap

React-Bootstrap е още една полезна добавка, която представлява пълна имплементация на библиотеката Bootstrap (от създателите на Twitter) чрез React компоненти. Няма зависимост от bootstrap.js или jQuery. Библиотеката може да се използва както с ***CommonJS*** модули така и чрез ES6 модули с Babel, AMD, или като глобален JS скрипт. Тъй като React-Bootstrap не зависи от конкретна версия на Bootstrap, не носи допълнителен CSS. Въпреки това, някои елементи се нуждаят от стиловете за да работят нормално. Как и кои стилове ще бъдат набавени зависи изцяло от потребителя, но най-лесния начин за това е

добавянето на последните версии чрез ***CDN*** (Content Delivery Network). За по сложните сценарии на използване е най-добре да се използва Webpack, за да могат да бъдат добавени всички стилизиращи файлове като част от процеса по пакетиране.

3.3.6 Потребителски интерфейс

Преди разглеждането на самия интерфейс, без впускане в голяма дълбочина е важно да се отбележи как се осъществява комуникацията между сървърното и клиентското приложение. Това става чрез библиотеката за HTTP заявки ***Axios***. Тя от своя страна е покрита от статичен сървис, който се използва за цялото приложение и през него минават всички заявки към сървъра. На фиг. 23 демонстрирана имплементацията на HTTP сървиса.

```
import axios from 'axios'
import Auth from './Auth'

const BASE_URL = 'http://localhost:5000/api'

class Http {
  static withOptions (secured = false) {
    let options = { headers: {} }

    if (secured) {
      options.headers['Authorization'] = `bearer ${Auth.getToken()}`
    }

    return options
  }

  static processResponse (axiosPromise) {
    return axiosPromise
      .then(res => res.data)
      .catch(err => {
        if (err.response) {
          return err.response.data
        }
        window.alert(err)
      })
  }

  static get (url, secured = false) {
    return Http.processResponse(
      axios.get(`${BASE_URL}${url}`, Http.withOptions(secured)))
  }

  static post (url, data, secured = false) {
    return Http.processResponse(
      axios.post(`${BASE_URL}${url}`, data, Http.withOptions(secured)))
  }

  static put (url, data, secured = false) {
    return Http.processResponse(
      axios.put(`${BASE_URL}${url}`, data, Http.withOptions(secured)))
  }

  static delete (url, secured = false) {
    return Http.processResponse(
      axios.delete(`${BASE_URL}${url}`, Http.withOptions(secured)))
  }
}

```

Фиг. 23 Имплементация на HTTP сървиса, който работи с React приложението

За да се осъществи сигурността на сървърното приложение и уеб клиента се използват хеширани ***token*** низове, които гарантират самоличността на потребителя (Фиг. 24)

```
{
    "success": true,
    "message": "You have successfully logged in.",
    "result": {
        "user": "Martin.Kotrulev",
        "userId": "7118ccc9-d7e9-4315-8152-60e4087f9439",
        "expires": "2018-03-05T08:51:45.6357187Z",
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
            .eyJ1bmlxdWVfbmFtZSI6Ik1hcnRpbi5LbS5Rydmxldi1aImVtYWlsIjoibWFydGluLmtvdHJ1bGV2QGV4YW1wbGUuYmciLCJuYW1laWQiOiI3MT
            E4YzNjOS1kN2E5LTQxMTUtODE3Mi0ZOWU0MDA3Zjk0MzkiLCJuYmYiOjE3MTk4MDc5MDUsImV4cCI6MTUyMDIzOTkwN5wiaWF0IjoxNTE5ODA3O
            TA1LCJpc3MiOiJVbmlxdWl6Yml0IiwiYXVkIjoiVW5pcXVpemJpdCJ9.ZUvhLgIgMeMTvjWPwtyJnjnKDmMLlOo8evlVee44Kx4"
    }
}
```


Фиг. 24 Примерен токен генериран от сървъра

За запазването и контрола на ***token***-а се грижи друг сървис, който записва всичката полезна информация в локалното хранилище на браузъра и го извлича когато е необходимо. Неговата имплементация е поместена на фиг. 25

```
class Auth {
  static saveUser (user) {
    window.localStorage.setItem('user', user)
  }

  static getUser () {
    return window.localStorage.getItem('user')
  }

  static saveUserId (userId) {
    window.localStorage.setItem('userId', userId)
  }

  static getUserId () {
    return window.localStorage.getItem('userId')
  }

  static setTokenExpiration (expires) {
    window.localStorage.setItem('expires', expires)
  }

  static authenticateUser (token) {
    window.localStorage.setItem('token', token)
  }

  static isAuthenticated () {
    let isValid = window.localStorage.getItem('token') !== null

    if (!isValid) return isValid

    isValid = Date.now() <= new Date(
      window.localStorage.getItem('expires'))

    if (!isValid) this.deauthenticateUser()

    return isValid
  }

  static deauthenticateUser () {
    window.localStorage.removeItem('token')
    window.localStorage.removeItem('user')
    window.localStorage.removeItem('userId')
  }

  static getToken () {
    return window.localStorage.getItem('token')
  }
}
```

Фиг. 25 Имплементация на Auth сървиса

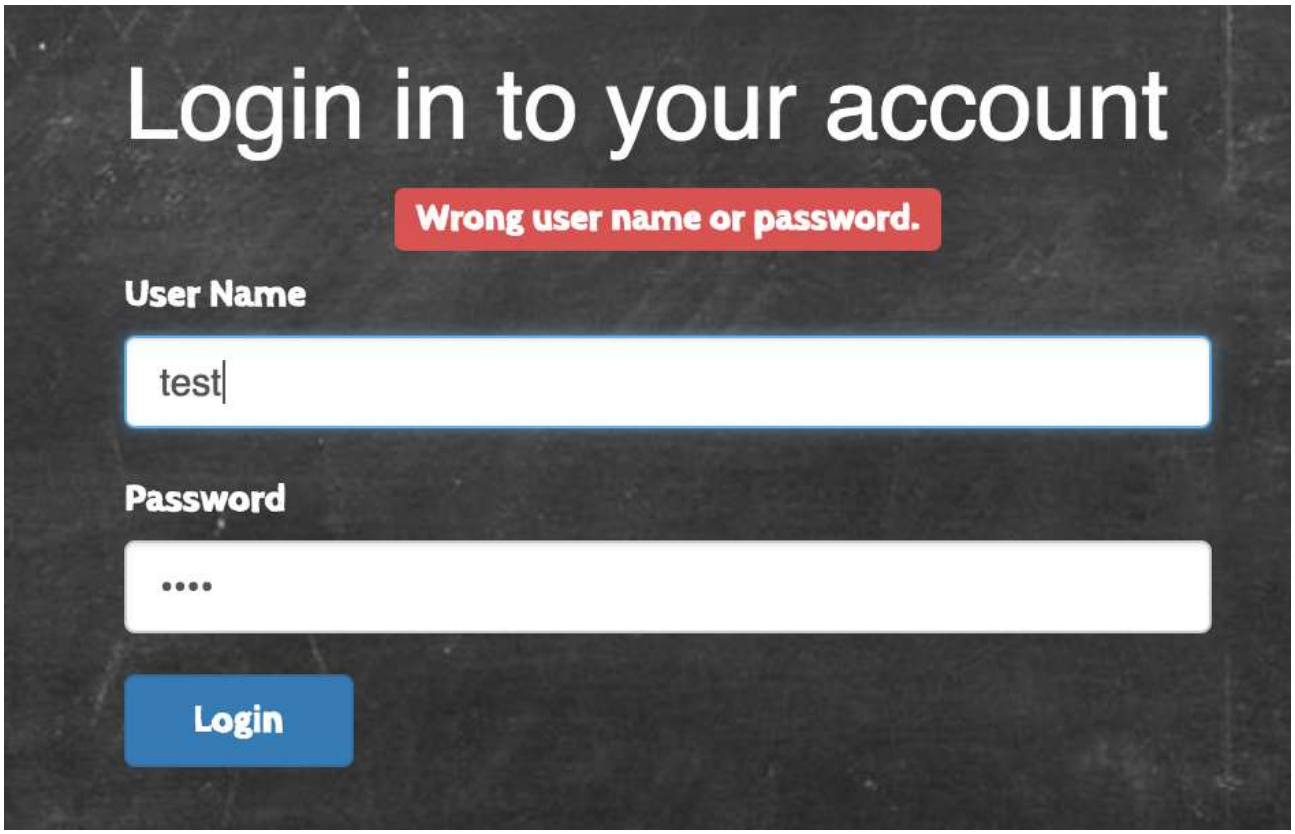
При създаването потребителския интерфейс основна двигателна сила са максимално олекотен, ненатоварващ дизайн и простота на работа с приложението. На основния екран (фиг. 26) могат да бъдат видени някои от основните белези на дизайна. Това са черната учебна дъска, тебеширения шрифт и жълтия шрифт на логото (и двата шрифта са от Google Fonts) както и основните менюта, видими през цялото време.

Фиг. 26 Основен екран

Основните менюта са Groups, Quizzes, потребителското меню с две подменюта водещи към собствените групи и тестове. Най-добрата част от използването на React е че всички компоненти се преизползват на много места, а чрез програмна логика се решава кога, кой, какво да вижда.

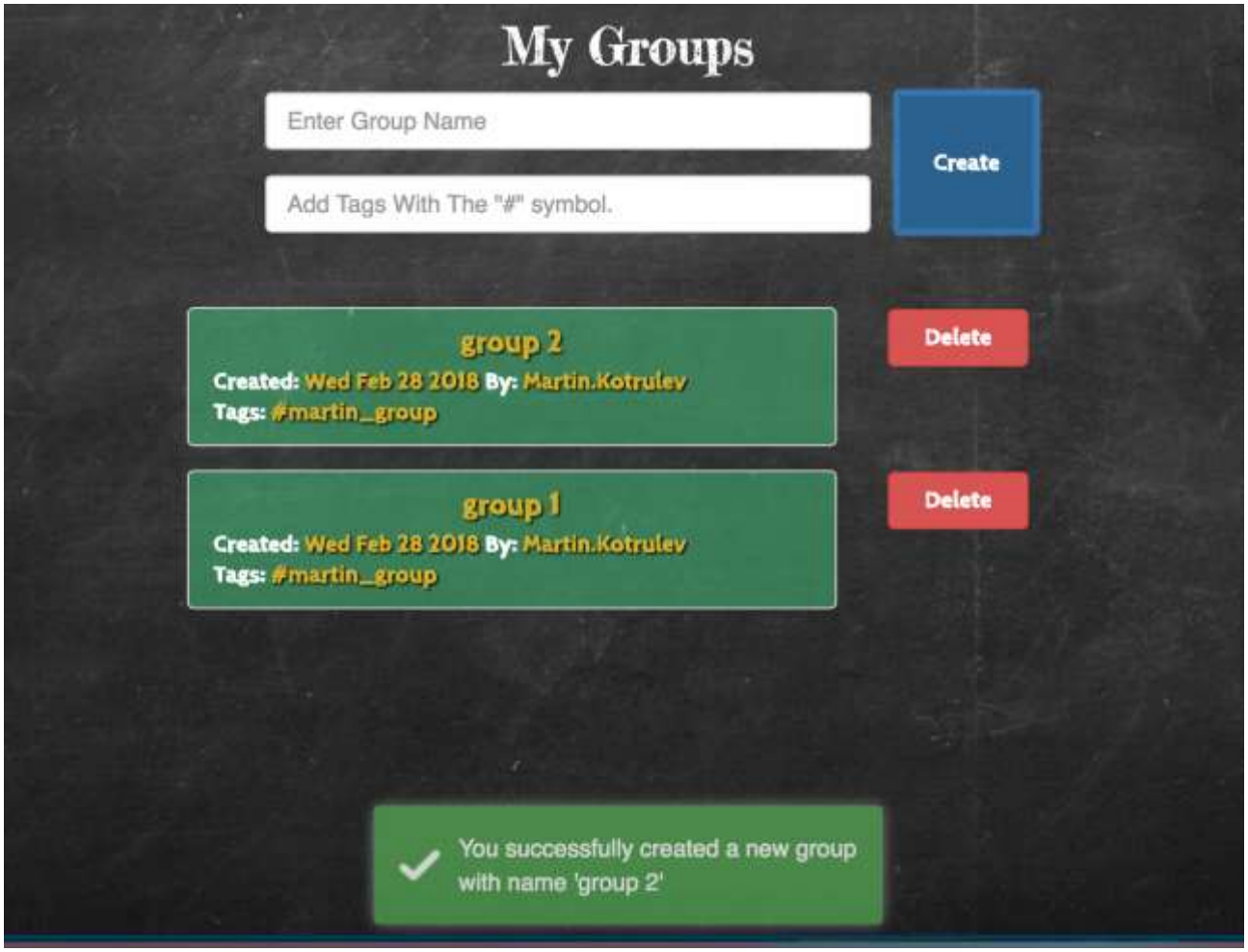
Регистрационната форма и формата за вписване (фиг. 27 и 28) като цяло нямат особено сложна логика. Притежават поле за грешка, което свети в червено в случай на грешки от клиентската валидация или от сървърната. Този вид грешки са консистентни през цялото приложение и се появяват при всяка форма, която притежава някакъв вид валидация.

Фиг. 27 Панел за регистрация



Фиг. 28 Панел за вписване с грешка

Панелите за създаване на групи и търсене отново се преизползват като в зависимост от пътя въведен на React рутера или специфики за потребителя се визуализират различен набор от данни и форми. Сините бутони служат за създаване на нови обекти, червените за триене, а зелените са в редките случаи когато е позволена някакъв вид модификация. (фиг. 30)



Фиг. 29 Екран за създаване на на Груп

Търсенето се осъществява по подобен начин и отново компонентите остават същите. Там където може да бъде създаван нов тест или група се появява формата за създаване, а там където се разглеждат се появява формата за търсене. Търсенето от своя страна става чрез въвеждането на тагове, слети със знака “#” пред името (фиг. 30 и 31).

Фиг. 30 Екран за "моите" тестове

Фиг. 31 Форма за създаване на тестове в група

При създаване на тестове нещата придобиват доста по интересен тон. Тук е възможно директното добавяне на нови въпроси, изтриването им, добавяне на отговори към въпрос и тяхното изтриване. Отново дизайна е напълно интуитивен и всички правила за бутоните са следвани изцяло. Интересното в случая е, че целия тест се обновява наведнъж със "Save Quiz", а с бутона "Publish Quiz" става достъпен от всички. Компонента за тестове също е преизползваем и изпълнява двойна функция. При собствен тест потребителя получава въпросите и отговорите в оригиналният им ред както и пълна свобода да ги редактира. Ако същия тест бъде достъпен от друг потребител, в роля

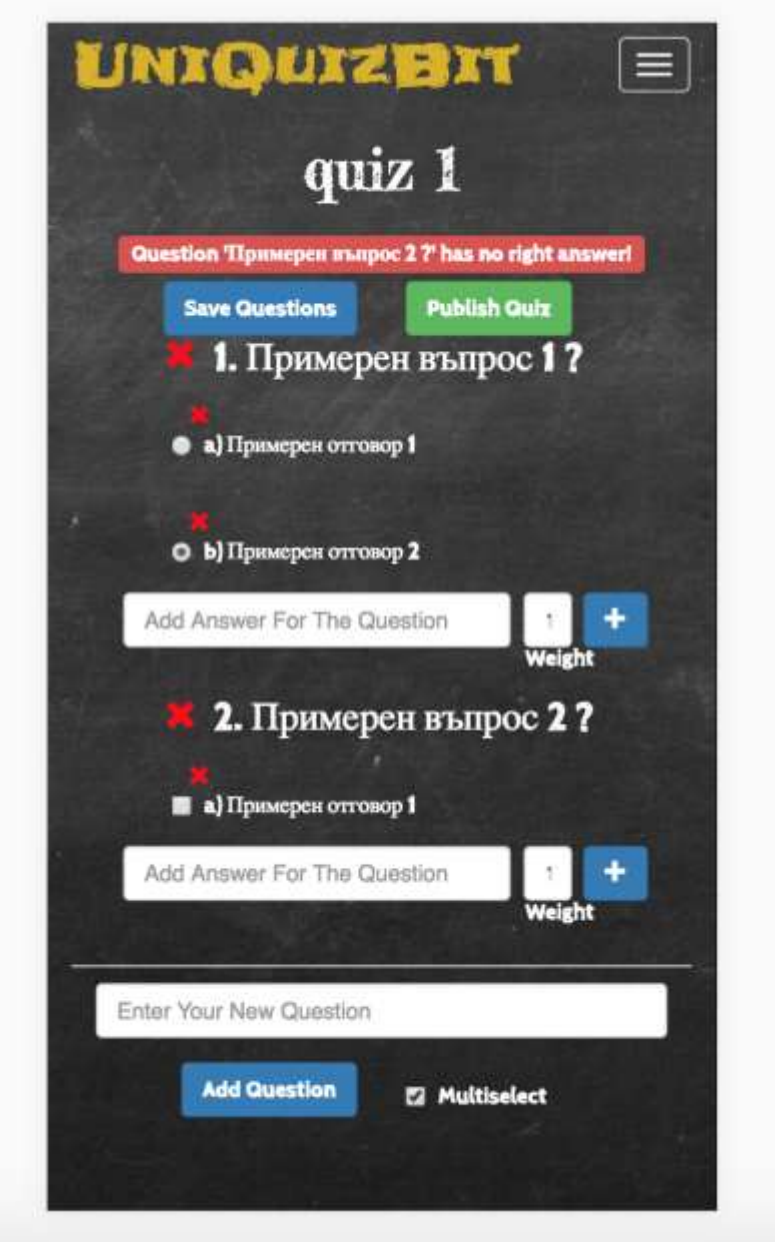
влизат всички правила за правене на тест и регистриране на прогрес. Целия интерфейс за редакция, също така, е забранен за достъп и не се вижда.

Фиг. 33 Форма за редактиране на въпросите по тест

Фиг. 34 Бутон за показване на повече резултати в края на всеки списък

За финал може да се добави, че приложението е напълно съвместимо и с мобилни телефони благодарение React Bootstrap (фиг. 35). Допълнително, самото клиентско приложения би могло да се доразвие чрез React Native, за да бъде реализирано естествено за мобилните платформи.

В тази глава беше представен потребителския интерфейс, който дава основния облик към приложението UniQuizBit.



Фиг. 35 Изглед на приложението през мобилен телефон

3.4 Възможности за надграждане

3.4.1 Ограничаване на достъпа и времевия диапазон на тестовете

Едно от архитектурните решения още по време на дизайна е да се въведе начин за ограничаването на достъпа и времевия диапазон на тестовете. Какво ще рече това? Всеки тест притежава полета за началната и крайната си дата и час плюс допълнително поле, посочващо дали теста може да бъде решаван само веднъж от даден потребител. Това позволява планирането и стриктното протичане на важни тестове (каквито са изпитите напр.). Допълнително ниво на сигурност е добавено с полето “парола”. С негова помощ може да се подсигури участието само на конкретни потребители, които са получили паролата предварително. Тази функционалност присъства в сървърната услуга и би могла да бъде лесно реализирана в клиентско приложение.

3.4.2 Автоматизирано приключване и оценяване на тестовете

Един от проблемите, който неминуемо може да възникне при решаването на тестове, е че понякога заради външни причини като липса на връзка, изтекло време, за което стана на въпрос по-рано, може процеса по приключване на теста да не завърши. Тогава освен липсата на оценяване на даден потребител за даден тест, особено когато става въпрос за специфични ситуации като ограничаване на времето, може да се появи и неконсистентност на данните в таблиците отговорни за прогреса. Решението на такъв вид проблем, най-често се решава от някакъв вид сървърен работник с разписание, който изпълнява задачи през определено време.

Популярно решение в .NET средите е ***Hangfire***. Той е лесен за работа и лесен за настройване. За периодичните, интервалови задачи използва класически ***cron*** низове. Не е обвързан с операционната система и не се налага инсталирането на допълнителен софтуер.

Задачите на заден план представляват статични или инстантни методи със стандартни аргументи, което означава, че не се наследяват базови и не се

имплементират интерфейси. Веднъж щом задачата е създадена, Hangfire поема щафетата и носи отговорността задачите да бъдат изпълнени “поне веднъж”.

При “падане” или терминиране на приложението, незавършилите задачи на заден ще бъдат отново изпълнени автоматично. За това е грижи автоматично създадената база данни от Hangfire. Въпреки че инсталацията по подразбиране SQL Server и използва избирателна техника при подбора на задачи, могат да се използват ***MSMQ*** (***Message Queing***) или ***Redis*** разширенията, за да се редуцира забавянето от процесите до минимум. Задачите могат да бъдат запазвани и в други бази като PostgreSQL и MongoDB, което идеално пасва на сегашната конфигурация на UniQuizBit.

Работните филтри позволяват добавяне на персонализирана функционалност към процесите на заден план по подобие на ASP.NET MVC ***action*** филтрите.

В контекста на UniQuizBit, Hangfire може да се настрои периодична задача, която да проверява за приключили тестове, които са ограничени от времева рамка и автоматично да ги оценява, затваряйки и запазения прогрес. Подхода към това може да е LINQ заявка или запазена процедура в базата.

Разбира се винаги може да се разчита и на ***cron*** решение, което е част от операционната система или от сървъра за база данни. Hangfire, обаче предлага свобода, гъвкавост и независимост както всяка друга технология, която е избрана в настоящия проект.

3.4.3 Система за абониране към групи и тестове

Една от основните услуги, които предоставят всички социални мрежи е абонирането. Когато човек иска да проследи дадено събитие, нищо не надминава абонамента. По този начин се предоставя възможност за автоматизирано “подсещане” на потребителя, за да не се налага неговият постоянен ангажимент. Съществуват множество начини за изпращане на напомнящи съобщения за абонираните, вариращи от изпращане на e-mail и sms до така наречените ***push-нотификации***, с които може би не съществува човек,

които да използва “умно” устройство и да не се е сблъсквал. За съжаление в рамките на този проект не беше предвидена подобна функционалност и именно за това е запазена за тези редове. Основното, което е необходимо в UniQuizBit е да се предостави възможност за абониране към дадена група от тестове или към потребител, за да могат потребителите да получават съобщения когато се добавят нови тестове. Това ще даде възможност за проследяването на тестовете на даден потребител или следването на цяла група отговаряща на интересите на абоната. Друга още по-важна способност би била съобщения за наближаващ тест (в случая на тестове, които са ограничени по времеви интервал), за да се предостави по-голяма сигурност на абонатите, че няма да пропуснат важните тестове. Отново една примерна реализация на тази функционалност би могло да бъде с помощта Hangfire, а потребителите да бъдат сигнализирани с изпращане на e-mail, чрез вградената услуга за изпращане на e-mail-и на .NET Core.

3.4.4 Генериране на статистически данни

Много потребители или институции биха желали да имат опцията за генериране на статистика. Такава статистика може да варира от списъци на правилни/грешни отговори, участници взели участие в даден тест и техните резултати, средни оценки на тестовете по групи и много други. За щастие всички тези статистики могат да бъдат извлечени от сегашната архитектура, защото са замислени предварително и заложени в първоначалния дизайн. Подобно надграждане би било сравнително лесно, след като бъдат поставени изискванията за набор от статистически справки. Допълнително могат да бъдат добавени, след това, изгледи за отделните статистики в уеб приложението с опции за печат както и изгледи на ниво база данни, където справките да бъдат съхранявани и по лесно достъпвани.

3.4.5 Интеграция с други системи

Тук идва момента да бъде спомената отново темата за цялостна учебна система. Както стана дума, в по ранните глави, сървърната част на UniQuizBit е разработена като REST API неслучайно. Освен възможностите за

разработка на различни уеб или мобилни клиенти, които да работят с това API, то дава възможност за интеграция и с цели системи. Отделената архитектура на сървърното приложение позволява директната му интеграция в други приложения или набор от приложения, част от цялостни решения. Учебните платформи са тези, които най-често са облагодетелствани от тестовото изпитване и като такива са идеална целева група, макари не единствена. Сравнително малкият му обхват и конкретиката на проблема който решава, може дори да го класифицира като ***microservice*** и да бъде самостоятелно разширявано, поддържано и доставяно отделно. Тези качества го правят идеален кандидат за интеграция във всякакви системи, които биха изисквали някакъв вид тестово изпитване.

ЗАКЛЮЧЕНИЕ

Настоящата работа изминава дългия, но задоволителен път от идеята до реализацията. Поставя си за цел не само да опише изградения софтуер, който стои като основа, но и да представи технологиите, които помагат за неговото изграждане. За никой не е тайна, че в днешно време ИТ средата е твърде динамична. Нови технологии идват и си отиват в рамките на няколко месеца. Постоянната надпревара в създаването на все по-модерен и все по-мощен хардуер, изисква и модерен софтуер, който да го управлява. В основата на всички софтуери, пък е заложено бързодействието. Вече никой не иска да чака, защото за вниманието на потребителя се борят още много други. Освен скоростната работа, вниманието приковава и неангажиращият, интуитивен интерфейс. Не само, че за това са написани множество книги, но и всеки целящ да предостави или продаде даден софтуерен продукт трябва да обърне особено внимание в покриването на този вид изисквания. Интересен дизайн, лесна навигация, поддръжка за мобилен телефон. Все неща, които вече възприемаме за даденост, но зад които се трудят хора, целящи да доставят възможно най-добрия си продукт.

В този дух и ред на мисли, първата глава разглежда и сравнява множество приложения, платформи и системи с отворен код, за да съпостави изискванията, които стоят пред обучителния софтуер.

За доставяне на добри продукти в днешно време, разработчиците имат нужда и от максимално модерни и мощни инструменти за разработка и доставка. В много случай новите технологии не само подобряват начина на работа, но съчетават и изцяло нови методологии за работа. DevOps инструменти като контейнеризация с продукти като Docker, даващи изулирана среда и лесен начин за доставка и контрол на софтуера и инструменти за продължителна интеграция и продължителна доставка като Jenkins са само два примера от много други, доказващи че създаването на софтуер в момента е един много сериозен процес, в който разработчиците се стремят да подсигурят и автоматизират, така че да си осигурят бърз и надежден начин за доставката на перфектната услуга.

Именно поради гореспоменатите причини за избора на технологии беше посветена цялата втора глава на настоящата работа. В нея бяха разгледани обстойно технологиите използвани за изграждането на проекта UniQuizBit. Първо беше засегната темата за базата данни в лицето на PostgreSQL. Разгледани бяха предимствата с цел да се аргументира използването ѝ. След това представен инструмента за мениджмънт на бази данни DBeaver. Много полезен софтуер позволяващ манипулацията на бази от голям брой различни доставчици. Третата засегната тема беше избора на сървърен език в лицето на C#. Направен е исторически разбор и са подчертани силните му качества. Сървърният език сам по себе си не стига. За това следващата тема засягаше технологията за сървърни приложения .NET Core и начините, по които улеснява живота на разработчика. Втора глава, след това, обръща внимание на технологиите за разработка на уеб приложението. Първо беше представен JavaScript и най-новите му допълнения, а след това - една от горещите технологии, която работи с него, в изграждането на клиентски приложения - React. В края на главата беше засегната системата за контрол на версиите GitHub.

Третата глава е посветена на практическото изграждане и в началото запознава с базата данни по таблици и техните връзки. Описани бяха подробно моделните свойства и целите, които покриват от към програмна логика. След това беше представена архитектурата на сървърното приложение и похватите на работа с .NET Core. Описани бяха REST контролерите и по-интересните моменти в логиката. По нататък беше разгледано уеб приложението с допълнителните технологии подпомагащи процеса на разработка, след което беше представена и самата функционалност. В края на главата беше обърнато внимание на възможно бъдещо доразвиване и интеграция.

Настоящата работа си постави за цел да разгледа и проследи проблематиката и решенията при разработката на приложението UniQuizBit и неговата сървърна услуга като предостави широк поглед над проблемите, които се налага да бъдат решени, инструментите необходими за тяхното решение и самата му реализация. За финал бяха разгледани евентуалните перспективи, тъй като Интернет базираните услуги никога не живеят в изолирана среда и възможностите пред тях са неограничени.

ИЗПОЛЗВАНИ ИЗТОЧНИЦИ

1. Antonio, PacCassio de Sousa. Pro React - Apress; 1st ed. edition (December 24, 2015)
2. Bell, Peter, Beer, Brent. Introducing GitHub A Non-Technical Guide - 2015, O'Reilly Media, Inc.
3. Bertoli, Michele. React Design Patterns and Best Practices - Packt Publishing, 2017
4. Create React App - https://github.com/facebook/create-react-app 15.02.2018
5. DBeaver Overview - https://dbeaver.com/dbeaver-overview/ 15.02.2018
6. Flux Concepts https://github.com/facebook/flux/tree/master/examples/flux-concepts 15.02.2018
7. LINQ: .NET Language-Integrated Query - https://msdn.microsoft.com/en-us/library/bb308959.aspx 15.02.2018
8. New features in EF Core - https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-2.0 15.02.2018
9. Obe, Regina O., Hsu, Leo S. PostgreSQL Up & Running - 2nd edition, 2014, O'Reilly Media, Inc
10. The Top 8 Open Source Learning Management Systems - https://elearningindustry.com/top-open-source-learning-management-systems - 15.02.2018
11. The Top 10 Educational Websites for Taking Online Courses https://www.lifewire.com/educational-websites-for-taking-online-courses-4072166 15.02.2018
12. Top 18 Online Quiz Makers For Teachers and Educators - https://myelearningworld.com/top-10-free-online-quiz-makers-for-teachers-and-educators/ 15.02.2018
13. Troelsen, Andrew. Pro C# 7: With .NET and .NET Core - Apress; 8th ed. edition (November 21, 2017)
14. Why React? - https://reactjs.org/blog/2013/06/05/why-react.html 15.02.2018
15. Why VSCode? - https://code.visualstudio.com/docs/editor/whyvscode - 15.02.2018