

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА
ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ



Катедра „Компютърни науки и технологии“

КУРСОВА РАБОТА

Тема:

Проектиране и реализиране на система за
намиране на фитнес инструктори

Изготвил: Мариян Веселинов Желязков, Яна Галинова Илиева

Специалност: Софтуерни и интернет технологии

Факултетен номер: 18621641, 18621762

ТУ Варна, 2021 г.

Ръководител:

/х. ас. Д. Димитров/

Съдържание

Съдържание	2
Увод	4
Анализ на проблема	6
Защо е необходимо такова приложение и какви проблеми решава?	6
Проучване	8
Анализ на конкуренцията	8
Анализ на бариерите за навлизане на пазара	9
Допитване	10
Използвани технологии	14
Vue JS	14
Model-View-Viewmodel	14
Single-Page Application	15
Declarative Rendering	15
Routing	16
State-management	16
Tailwind CSS	18
Защо Tailwind Css?	19
Предимства	19
Java	19
Предимства	20
Защо Java за бекенд?	20
Spring Framework	22
Контейнер на зависимостите (инжектор)	22
Аспектно-ориентирано програмиране	23
Достъп на данни	23
WebSocket	24
Model-View-Controller	25
PostgreSQL	26
Проектиране	28
Персони	28
Storyboard	29
Use-case диаграма	31
Essential use case диаграма	32
Sitemap	33

Core model	34
Moodboard	36
Реализация	37
Концептуален модел	37
Back end разработка	37
Front end разработка	41
Тестове и резултати	44
WebSockets и Spring Security	44
Препращане на данни през компонентите в VueJS	44
Интеграцията на SockJS и Spring WebSocket	44
Заключение	45
Приложение	46
Приложение № 1	46
Приложение № 2	48

Увод

Всеизвестен факт е, че тренировките и упражненията за тяло значително подобряват здравето на човека по много различни начини. Типът на тренировката може да е различен с многообразието от цели, които може човек да си постави. Може да се тренира за сила, за тонус, за професионални състезания в различни спортове и т.н. Ето и някои от предимствата, които носи подобен тип дейност:

Помага да се контролира теглото. Заедно с диетата, упражненията играят важна роля в контролирането на теглото и предотвратяването на затлъстяването. За да се поддържа теглото, калориите, които се приемат, трябва да се равняват на енергията, която се изгаря. За да се отслабне, трябва да се изгарят повече калории, отколкото се приемат.

Намаля риска от сърдечни заболявания. Упражненията укрепват сърцето и подобряват кръвообращението. Увеличеният приток на кръв повишава нивата на кислород в тялото. Това помага за намаляване на риска от сърдечни заболявания като висок холестерол, коронарна артериална болест и сърдечен удар. Редовните упражнения също могат да понижат кръвното налягане и нивата на триглицеридите.

Помага на тялото да управлява нивата на кръвната захар и инсулина. Упражненията могат да понижат нивото на кръвната захар и да помогнат на инсулина да работи по-добре. Това може да намали риска от метаболитен синдром и диабет тип 2. И ако вече е налично едно от тези заболявания, упражненията могат да помогнат за управлението му.

Да помогне да се спрат цигарите. Упражненията могат да улеснят спирането на тютюнопушенето, като намалят апетита и симптомите на отнемане. Освен това може да помогне за ограничаване на теглото, което може да се натрупа, когато се спре тютюнопушенето.

Подобрява психическото здраве и настроение. По време на тренировка тялото отделя химикали, които могат да подобрят настроението и да накарат упражняващият се да се чувства по-спокоен. Това може да помогне със справянето със стреса и да се намали риска от депресия.

Помага да се поддържат уменията за мислене, учене и преценка остри с напредване на възрастта. Упражненията стимулират тялото да отделя протеини и други химикали, които подобряват структурата и функцията на мозъка.

Да се укрепят костите и мускулите. Редовните упражнения могат да помогнат на децата и тийнейджърите да изградят здрави кости. По-късно в живота може също да забави загубата на костна плътност, която идва с възрастта. Правенето на упражнения за укрепване на мускулите може да помогне с увеличаването или поддържането на мускулна маса и сила.

Намалява риска от някои видове рак, включително рак на дебелото черво, гърдата, матката и белия дроб.

Намаля риска от падане. За възрастните хора изследванията показват, че извършването на упражнения за баланс и укрепване на мускулите в допълнение към умерено интензивна аеробна активност може да помогне за намаляване на риска от падане.

Подобрява съня. Упражненията могат да помогнат за по-бързо и по-дълго заспиване.

Подобрява сексуалното здраве. Редовните упражнения могат да намалят риска от еректилна дисфункция (ЕД) при мъжете. За тези, които вече имат ЕД, упражненията могат да помогнат за подобряване на сексуалната им функция. При жените упражненията могат да повишат сексуалната възбуда.

Увеличава шансовете да се живее по-дълго. Проучванията показват, че физическата активност може да намали риска от ранна смърт от водещи причини за смърт, като сърдечни заболявания и някои видове рак.

В последните две години възможността на хората да практикуват спорт във фитнес студия или на открито е силно ограничена вследствие на противоепидемичните мерки. От това следва, че тяхното здраве се влошава. Необходимо е да се вземат навременни мерки за предотвратяване на рисковете, свързани с продължително обездвижване.

Анализ на проблема

Защо е необходимо такова приложение и какви проблеми решава?

По време на световна пандемия и противоепидемични мерки, предимствата на фитнес заниманията се губят, поради факта, че хората тренират по-рядко или изобщо не го правят. Те са принудени да прекарват повече време вкъщи. Една от мерките за опазване здравето на населението включват затваряне на фитнес салони. Цялата ситуация около пандемията влияе психологически на хората - нарастващо безпокойство, нарастваща несигурност в бъдещето, доходи, неотговарящи на нарастването на цените, изобщо един куп причини да се остави на заден план тренирането за здраве.

Това налага търсенето на решение, което ще се справи с този проблем. Едно такова решение би било софтуер, който да премахва трудностите, пряко или непряко свързани с физическите упражнения, като например:

- трудността с намиране на добър инструктор;
- трудността с изготвяне на тренировъчна и диетична програма от специализирано лице;
- затрудненото пътуване;
- събиране на повече хора и срещи на лично и много други подобни причини;

Необходимостта от улеснение на процесите, свързани с тренировките се наблюдава по времето на пандемията, но тя не е единствената причина за създаване на такъв продукт. В съвременното много процеси се дигитализират за удобството на потребителя - плащане на битови сметки, умни телевизори, умни апартаменти, поръчване на бързо хранене, дори се създава виртуален 3D свят от технологичните гиганти, следователно защо да не се направи софтуерен продукт, който да облекчи и да помогне на потребителя:

- намирането и избора на фитнес треньор;
- общуването между двамата;
- следенето на прогреса;

и от друга страна (страната на треньора):

- управлението на личния треньорски бизнес;

- осъществяването на парични транзакции;
- изготвянето на тренировъчни и диетични програми;

Проучване

Анализ на конкуренцията

В анализа на конкуренцията се разглеждат две други приложения, които се занимават с онлайн фитнес консултиране - Transform и Trainerize. И двете представляват преки конкуренти за FitBook, защото са в същата ниша. В следната таблица са представени техните качества.

Правило	Transform	Trainerize
Информация за състоянието на системата	Системата не предоставя функционалност за известяване	Системата предоставя функционалност за известяване, която е и лесно достъпна за потребителя
Съвпадение на нуждите на реалния свят с предоставените от приложението	Информацията в платформата е предоставена на разбираем за таргетираните потребители език	Приложението съвпада с нуждите на таргетираната аудитория
Потребителски контрол и свобода на действията	Мобилното приложение съдържа бърз на началния екран и не може да се навигира извън него	Платформата има обособени роли за различните потребители с различно ниво на достъп до нейните функционалности
Консистентно представяне на информацията и стандарти	Сайтът има консистентно представяне на информацията. Интерфейсът на менюто е интуитивен.	Нестандартно, но удобно и консистентно представяне на информация

Предотвратяване на грешки и тяхното прихващане	Невъзможност да се регистрира или логне в мобилното приложение	Системата известява потребителя чрез диалогови прозорци в случай на грешка
Осигуряване на интерфейс вместо действия по памет	Удобно разделение на навигационното меню, което позволява лесно използване на функционалностите	Изградена навигация
Гъвкавост и ефективност	Платформата предлага преки пътища за удобство на запознатите с нея потребители, но не предлага възможност за персонализиране	Платформата не среща нуждите на потребителите за бърз достъп до основните функционалности
Минималистичен дизайн	Сайтът има минималистичен дизайн	Уеб приложението няма минималистичен дизайн
Обратна връзка	Системата не предоставя лесен начин за обратна връзка	Приложението не предоставя интерфейс за обратна връзка
Помощ и документация	Системата предоставя FAQ секция, която не е с добър UX дизайн	Системата предоставя помощ под формата на видеоклипове на разположение за потребителя

Анализ на бариерите за навлизане на пазара

При анализа на бариерите за навлизане на пазара, бяха идентифицирани следните точки:

- Законови съображения;

Въпросът дали фитнес инструкторите, които няма образование за диетолог, т.е. не са медицински лица, имат право да създават хранителни режими за своите клиенти. Съображенията са здравни,

защото без нужните компетенции, един фитнес инструктор може да навреди на човек.

- Тренъори;

Възможно ли е да се привлекат достатъчно тренъори, за да може да просъществува платформата. Считаме, че в сегашните времена, много тренъори биха се възползвали от това приложение.

- Клиенти;

Доколко клиентите имат желание да бъдат инструктирани онлайн, а не на живо. Отново заради сегашната противоепидемична обстановка - изглежда, че много хора биха харесали идеята, да могат онлайн пак да останат активни.



Допитване

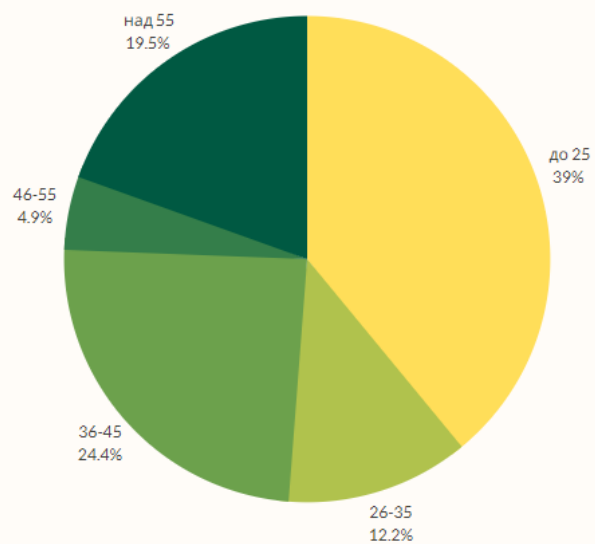
За целите на проучването са попитани 31 човека относно тяхното отношение към тренирането и към ситуацията с фитнес инструкторите. Техните отговори

са резюмирани в следващите графики. Изводът от тях е, че повече от половината тренират поне един на седмица и около половината биха използвали услугите на онлайн фитнес инструктор особено в условията на пандемия и наложени ограничения в следствие на противоепидемичните мерки.

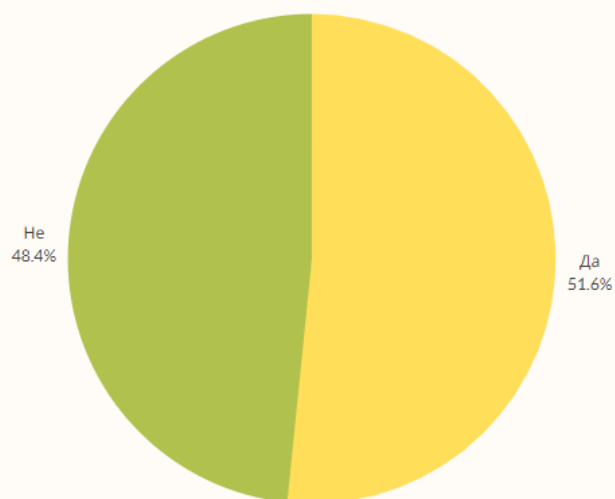
На базата на това проучване може да се счита, че приложение би се справило добре на пазара за онлайн фитнес платформи.



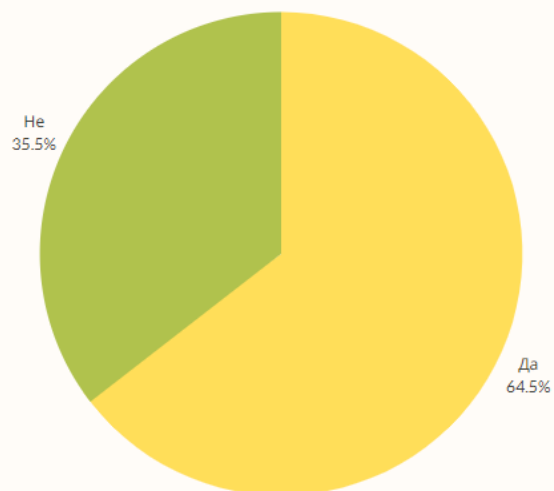
НА КОЛКО ГОДИНИ СТЕ?



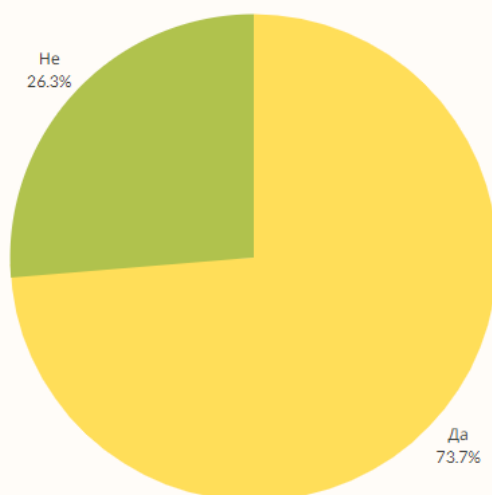
БИХТЕ ЛИ ИЗПОЛЗВАЛИ УСЛУГА КАТО ОНЛАЙН ФИТНЕС ИНСТРУКТОР?



**ЧУВСТВАТЕ ЛИ, ЧЕ
ПРОТИВОЕПИДЕМИЧНИТЕ МЕРКИ СА
ПОВЛИЯЛИ НЕГАТИЧНО НА ФИЗИЧЕСКАТА
ВИ АКТИВНОСТ?**



**ЛЕСНО ЛИ ОТКРИВАТЕ ДОБЪР ФИТНЕС
ИНСТРУКТОР?**



Използвани технологии

Vue JS



VueJS (или само Vue) е фронт енд JavaScript фреймуърк за построяване на потребителски интерфейс и Single-Page Applications с отворен код. Архитектурата му е Model-View-Viewmodel и се фокусира на declarative rendering и композиция на компонентите. Ядрото на Vue се фокусира върху View слоя от MVVM архитектурата като по-сложните функционалности (routing, state-management и т.н.) се предоставят чрез официално поддържани съпорт библиотеки и пакети. Първата версия на фреймуърка е пусната през Февруари месец 2014 г. от Evan You.

Model-View-Viewmodel

Model-view-viewmodel (MVVM) е софтуерен архитектурен модел, който улеснява отделянето на разработването на графичния потребителски интерфейс (View) от разработването на бизнес логиката (Model), така че изгледът да не зависи от конкретна платформа на модела. Viewmodel частта от MVVM е отговорен за преобразуването на обектите с данни от модела по такъв начин, че обектите да се управляват и представят лесно. В това отношение viewmodel е по-скоро модел, отколкото изглед, и обработва повечето, ако не и цялата логика на показване на изгледа.

Viewmodel може да реализира модел на посредник, организиращ достъп до бизнес логиката около набора от случаи на използване, поддържани от изгледа. MVVM е проектиран да премахне практически целия GUI код от изгледния слой, като използва функции за data-binding, за да облекчи по по-добър начин отделянето на разработката на изгледния слой от останалата част от шаблона. Вместо да изискват от разработчиците на UX да пишат GUI код, те могат създават data-binding към модела на изглед, който се пише и

поддържа от разработчиците на приложения. Разделянето на ролите позволява на интерактивните дизайнери да се съсредоточат върху нуждите на UX, а не върху програмирането на бизнес логиката. Така слоевете на приложението могат да бъдат разработени в множество работни потоци за по-висока производителност. Дори когато един разработчик работи върху цялата кодова база, правилното отделяне на изгледа от модела е по-продуктивно, тъй като потребителският интерфейс обикновено се променя често и късно в цикъла на разработка въз основа на обратната връзка с крайния потребител.

Моделът MVVM се опитва да спечели и двете предимства на разделянето на функционалната разработка, предоставено от MVC, като същевременно използва предимствата на data-binding и фреймуърка. Той използва свързващото устройство, модела на изглед и функциите за проверка на данни на всички бизнес слоеве, за да потвърди входящите данни. Резултатът е, че моделът и фреймуоркът управляват възможно най-много от операциите, елиминирайки или минимизирайки логиката на приложението, която директно манипулира изгледа.

Single-Page Application

SPA е вид реализация на уеб приложение, което зарежда само един уеб документ и след това актуализира основното съдържание на този отделен документ чрез JavaScript APIs като Fetch и XMLHttpRequest. Това позволява на потребителите да използват уеб сайтове, без да зареждат цели нови страници от сървъра, което може да доведе до повишаване на производителността и по-динамично изживяване, с някои недостатъци като SEO, повече усилия, необходими за поддържане на state-a, имплементиране на навигация и др.

Популярни SPA фреймуърци са Angular, React, Vue.

Declarative Rendering

Declarative Rendering може да се дефинира в сравнението му с противоположното Imperative Rendering по следния начин (даден е пример извън областта на програмирането – майка иска от детето си да си почисти стаята):

- Imperative – дава точни инструкции за начина на постигане на цел

Пример: „Събери си играчките, изхвърли боклука, почисти праха и мини с прахосмукачката.“

- Declarative – дава целта която трябва да бъде достигната, но не и инструкции как да се случи това

Пример: „Ето ти снимка на чиста стая. Искам да изглежда по този начин твоята стая за 30 минути.“

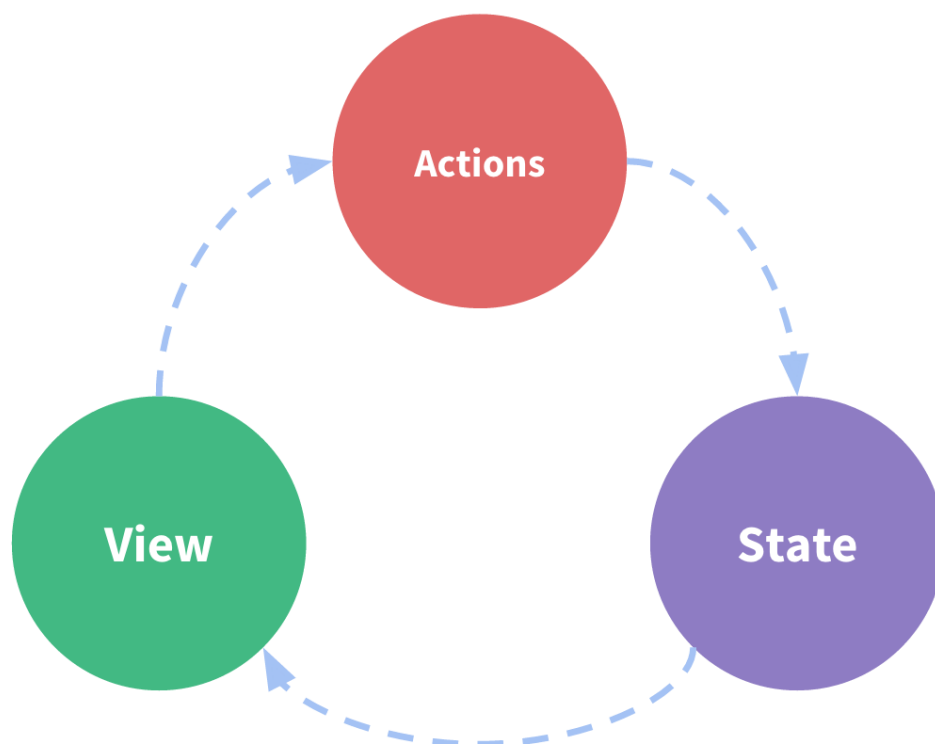
Declarative Rendering във Vue ни позволява да изобразяваме данни в DOM, използвайки темплейтен синтаксис. Използват се двойни фигурин скоби и се нарича интерполиране.

Routing

Routing е механизмът, чрез който заявките се свързват с някакъв код. По същество това е начинът, по който навигирате през уебсайт или уеб приложение. При кликане върху някаква връзка URL адресът се променя, което предоставя на потребителя някои нови данни или нова уеб страница. Routing-ът не е функция от основната библиотека на Vue, затова се използва съпорт библиотеката Vue Router.

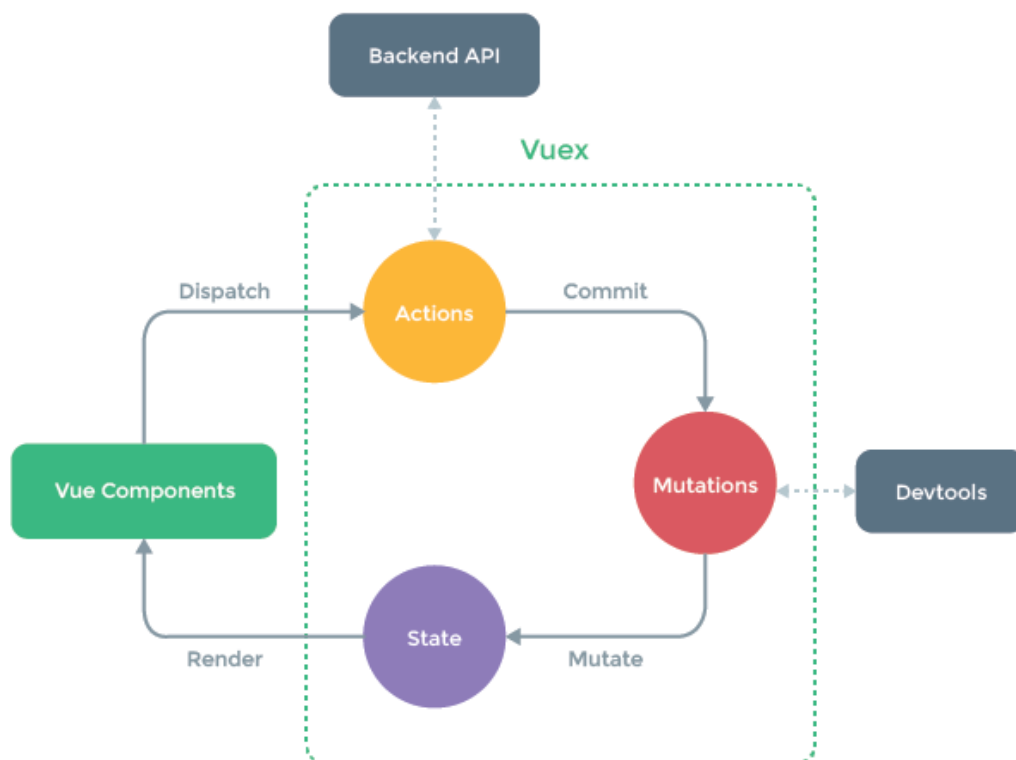
State-management

State-management е процесът на управление на състоянието на едно приложение. Както Routing-ът, така и State-management не е функция от основната библиотека на Vue и се налага да използваме съпорт библиотеката Vuex. Vuex служи като централизирано хранилище за всички компоненти в приложението, с правила, гарантиращи, че състоянието може да се променя само по предвидим начин.



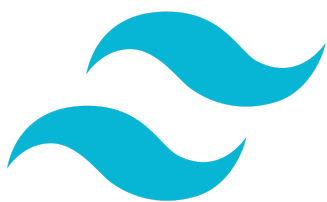
Фиг. 1. Репрезентация на концепцията "one-way data flow"

Идеята на Vuex е да се извлече споделеното състояние от компонентите и да се управлява в глобален сингълтън. По този начин дървото на компонентите се превръща в голям "изглед" и всеки компонент може да има достъп до състоянието или да задейства actions, независимо къде се намират в дървото. Чрез дефинирането и разделянето на концепциите, свързани с управлението на състоянието и прилагането на правила, които поддържат независимост между възгледите и състоянията, ние даваме на нашия код повече структура и поддръжка.



Фиг. 2. Схема на структурата на Vuex

Tailwind CSS



Tailwind CSS по същество е utility-first CSS фреймуърк за бързо изграждане на персонализирани потребителски интерфейси. Това е много адаптивен CSS фреймуърк на ниско ниво, който ни дава всички градивни елементи, от които се нуждаем, за да изградим дизайн, без никакви досадни догматични стилове, които трябва да се борим, за да премахнем. Красотата на Tailwind CSS е, че не налага спецификация на дизайна или как трябва да изглежда един сайт, а просто събира малки компоненти заедно, за да се изгради потребителски

интерфейс, който е уникален. Това, което Tailwind прави, е да вземе „суров“ CSS файл, да обработи този CSS файл върху конфигурационен файл и да произведе изходен CSS.

Защо Tailwind Css?

- По-бързо построяване на потребителски интерфейс;
- Използване на utility класове за изграждане на дизайн без писане на CSS в традиционния вариант;

Предимства

- Няма нужда постоянно да се мислят имена на класове на HTML елементите;
- Минимално количество код в CSS файла;
- Добро ниво на персонализиране;
- Лесно построяване на responsive страници;
- Промените се правят локално, което означава, че се отстранява постоянния;
- Мониторинг на глобален CSS файл;

Java



Java е обектно-ориентиран език за програмиране, разработен от Sun Microsystems и пуснат в употреба през 1995 година като част от Java платформата. Изходният код, написан на Java не се компилира до машинен код за определен микропроцесор, а до междинен език – байткод. Байткодът не

се дава за директно изпълнение от процесора, а се изпълнява от негов аналог – виртуален процесор, наречен JVM (Java Virtual Machine). Java е език за програмиране от високо ниво с общо предназначение. Синтаксисът му е подобен на C и C++, но не поддържа много от неговите възможности с цел опростяване на езика, улесняване на програмирането и повишаване на сигурността.

Предимства

- Лесна преносимост между различни платформи – веднъж написана и компилирана, една Java програма може да бъде стартирана на различни машини, независимо от архитектурата или операционната им система;
- допълнителни действия, извършвани от виртуалната машина, като освобождаване на паметта от обекти, които не се използват (garbage collector), проверка за размерността на масивите;
- възможността за контрол на правата на потребителя на ниво виртуална машина;
- първоначално заделяне на heap, част от паметта, резервирана за Java;
- висока степен на сигурност, поради факта, че програмистите не работят директно с паметта;

Защо Java за бекенд?

- Мащабируемост;

В Java separation of concerns позволява по-добро мащабиране. Когато обработването или входно-изходните операции (IO) се увеличават, може лесно да се добавят ресурси и да се преразпредели натоварването. Separation of concerns прави това прозрачно за приложението.

Компонентите на Java са лесно достъпни, което прави мащабирането на големи уеб приложения лесно. Езикът е гъвкав и не трябва да се пише толкова много код, за да се подобри мащабируемостта.

- Многонишковост;

Независимо от времето, което процесът отнема, сървърът остава responsive и следователно има по-малко проблеми. Приложението на

Java обработва бързо входа на потребителя, независимо от броя на едновременните потребители. Приложението използва оптимално кеша и ресурсите на процесора, като по този начин осигурява по-добра производителност.

- Богата Екосистема;

- Maven;

Apache Maven е инструмент за управление на проекти, който може да менажира различни аспекти на жизнения цикъл на проекта като изграждане, отчитане, документация. С него лесно се решават конфликти на версиите на библиотеки от различни пакети. А самите библиотеки, т.н. jar файлове, могат да се вземат от централното хранилище на Maven и да се поставят в pom.xml файла на проекта.

- Spring Framework;

Разглежда се в следващата точка.

- Tomcat;

Tomcat е уеб сървър за Java приложения. Той е безплатен и с отворен код и имплементира спецификациите на Sun Microsystems за JSP и Servlets. Представява контейнер за Java уеб приложения, който осигурява среда, в която да се изпълняват те. Spring използва embedded Tomcat уеб сървър. Заради това не е необходимо да се прави отделна инсталация и конфигурация.

- Управление на паметта;

Автоматичното управление на паметта на Java е значително предимство. На програмен език, ние разделяме паметта на две части, т.е. „стека“ и „хийпа“. Като цяло, хийпа има много по-голяма памет от стека. Java разпределя стекова памет на нишка. Хийпа съхранява действителните обекти, а променливите на стека се отнасят към тях. Хийп паметта е само една във всяка JVM, следователно се споделя между нишки. Самият heap обаче има няколко части, които улесняват garbage collecting-a в Java. Размерите на стека и хийп-а зависят от JVM.

Spring Framework



Spring Framework е фреймуърк с отворен код за Java платформата. Spring framework предоставя много функции, които улесняват разработването на Java-базирани enterprise приложения. Spring framework включва различни модули, които предоставят широк набор от функционалности:

Контейнер на зависимостите (инжектор)

Основното ядро в Spring Framework е неговият IoC (Inversion of Control) контейнер, който осигурява логически средства за конфигуриране и управление на Java обекти с помощта на reflection. Контейнерът е отговорен за управлението на жизнения цикъл на специфични обекти: създаването на тези обекти, извикването на инициализиращите им методи и конфигурирането им свързвайки ги заедно. Обектите, създадени от контейнера се наричат още управлявани обекти или бийнове (beans). Контейнерът може да бъде конфигуриран чрез зареждане на XML файлове или засичането на специфични Java анотации от конфигуриращите класове.

Обектите могат да бъдат получени чрез dependency lookup или dependency injection. Dependency injection е софтуерен шаблон за дизайн, който опростява Обектно- ориентираното програмиране (ООП) чрез намаляване на зависимостите между отделните класове. Така при създаването на един проект, използвайки някой обектно-ориентиран език се избягва зависимостта от конкретни имплементации, и се набляга повече на абстракции. Това се нарича още обръщане на контрола (inversion of control) – когато модулите от високо ниво не зависят от модули от по-ниско ниво. При dependency injection обектите се проектират по начин, при който те получават инстанции на обекти от различен външен модул код, вместо да са конструирани вътре в него. Dependency injection включва най-общо три елемента:

- зависим елемент;

- декларация на зависимостите на елемента, дефинирани като интерфейси, абстрактни класове;
- контейнер на зависимостите (инжектор), който създава инстанции на класове, които реализират дадена интерфейсна зависимост при заявка;

Зависимият елемент определя на какъв софтуерен компонент зависи за да си свърши работата, а контейнерът решава какви конкретни класове отговарят на изискванията на зависимия обект, и ги предоставя като зависимости. По този начин контейнерът по свой избор, може да променя конкретните имплементации, които се използват по време на изпълнение на програмата (run-time), а не по време на компилация (compile-time). По време на изпълнение (run-time) могат да бъдат създадени множество различни инстанции на даден софтуерен компонент, без значение, каква конкретна е била инжектирана. Това позволява кодът, който пишем да е много по модуларен и разкачен, което го прави в същото време и много по-лесен за тестване.

Аспектно-ориентирано програмиране

Аспектно-ориентираното програмиране (АОП) е относително нова парадигма в програмирането която цели да се увеличи модулността на софтуера и включва методи и инструменти които помагат за това. Почти всички програмни парадигми поддържат някакво ниво на групиране и капсулиране на проблеми в отделни, независими части чрез представянето им чрез абстракции (например, функции, процедури, модули, класове, методи), които могат да бъдат използвани за изпълнение, абстрахиране и композиране на отношения. Някои отношения „пресичат“ множество абстракции в програмния код, и се наричат междусекторни отношения. Като пример, може да се посочи „логването“ в дадена система, тъй като в зависимост от избрания вход в системата, поведението на всички методи и класове може да е различно.

Достъп на данни

Така наречените Spring frameworks за достъп на данни се отнасят за често срещани проблеми при работа с база данни, които разработчиците срещат. Както следват всички най-известни frameworks за достъп на данни в Java се

поддържат: JDBC, iBatis/MyBatis, Hibernate, JDO, JPA, Oracle TopLink, Apache OJB, и Apache Cayenne и други.

За всички изброени дотук frameworks, Spring поддържа следните характеристики:

- Resource management – Автоматично поискване и получаване на ресурси от базата данни;
- Exception handling – превежда уникалните данните за достъп в Spring йерархия;
- Transaction participation – прозрачно участие в настоящи транзакции;
- Resource unwrapping – извличане на база данни обекти от pool wrappers към който е свързан;
- Abstraction – абстракция за BLOB и CLOB манипулации;

Всички тези възможности са достъпни когато се използват шаблонни класове (template classes) предоставени за всеки framework който Spring поддържа.

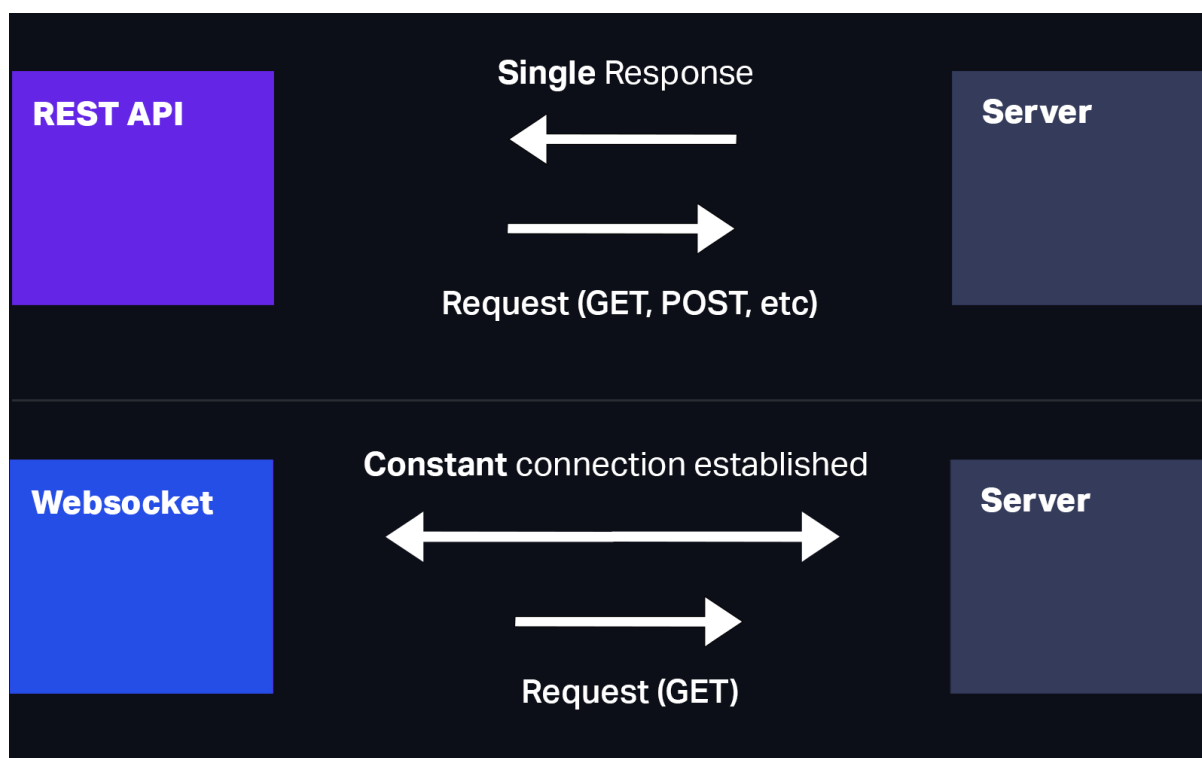
WebSocket

WebSocket е протокол, който установява двупосочна интерактивна комуникация между клиент и сървър върху TCP връзка.

Този протокол е отделен от HTTP, но съвместим с него. За разлика от HTTP, WebSocket е протокол със състояние, което означава, че връзката между клиента и сървъра остава да съществува, докато не бъде терминирана от някоя от двете страни.

Този протокол се използва при имплементиране на:

- Уеб приложения в реално време - например уеб сайт за bitcoin търгуване;
- Игри;
- Чатове;



Фиг. 3. Съпоставка на HTTP комуникация с WebSocket комуникация

Model-View-Controller

Model-View-Controller (MVC) е архитектурен модел за изграждане на потребителски интерфейси. Той разделя дадено софтуерно приложение на три взаимосвързани части, така че да се отделят вътрешни представяния на информация от начините, по които информацията се представя на потребителя/или от страна на потребителя: контролери, изгледи, модели. Както и при други софтуерни модели, MVC изразява „core of the solution“ на проблем, докато позволява да бъде адаптиран за всяка система. Специфични MVC архитектури могат да се различават значително.

Елементи: Централният елемент на MVC – моделът, улавя поведението на приложението по отношение на своя домейн, независимо от потребителския интерфейс. Моделът директно управлява данните, логиката и правилата на приложението. The view изгледът може да бъде всеки изход за представяне на информация, например чрез графика или диаграма; множество изгледи към една и съща информация са възможни, като стълбчета за управление и табличен изглед за счетоводители.

Третата част Controller контролера, приема входа от потребителя и го конвертира в команди за модела или изгледа.

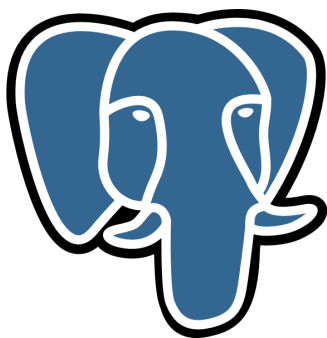
Взаимодействия: В допълнение за разделяне на приложението на трите вида елементи, дизайн на model–view–controller определя взаимодействието между тях.

The Controller – Контролерът може да изпраща команди към модела за актуализиране състоянието на модела (например, редактирането на документ). Той също може да изпраща команди към свързаната с изгледа част, за да се промени представянето на модела (например, чрез скролиране на документа).

The model – Моделът уведомява свързаните части с изгледа и контролера, когато е налице промяна в състоянието му. Това уведомяване позволява на изгледа да изведе на изхода актуалната информация и контролера да променя разположението на наборите от команди.

The view – Изгледа изисква информация от модела, който използва, за да се генерира изходната информация за представяне на потребителя. Използване в уеб приложения: Въпреки че първоначално е разработен за настолните компютри Model-View-Controller е широко приет като архитектура за World Wide Web приложения в голяма част от езици за програмиране. Няколко търговски и нетърговски уеб приложения са създадени, които прилагат тази схемата. Тези приложения се различават по своите интерпретации, основно в начина, по който отговорностите им Model-View-Controller са разделени между клиента и сървъра.

PostgreSQL



PostgreSQL, известна също като Postgres, е безплатна система за управление на релационни бази данни с отворен код (RDBMS), която набляга на разширяемостта и съответствието с SQL. Първоначално е наречен POSTGRES, позовавайки се на произхода му като наследник на базата данни Ingres, разработена в Калифорнийския университет, Бъркли. През 1996 г. проектът е преименуван на PostgreSQL, за да отрази неговата поддръжка за

SQL. След преглед през 2007 г. екипът за разработка реши да запази името PostgreSQL и псевдонима Postgres.

PostgreSQL включва транзакции със свойства на атомарност, последователност, изолация, издръжливост (ACID), автоматично обновяеми изгледи, материализирани изгледи, тригери, външни ключове и съхранени процедури. Той е проектиран да обработва редица натоварвания, от единични машини до складове за данни или уеб услуги с много едновременни потребители. Това е базата данни по подразбиране за macOS Server и е достъпна и за Windows, Linux, FreeBSD и OpenBSD.


Проектиране

Персони

Персоните са измислени персонажи на хора, които се създават при фазата на проучването, за да се представят различни видове потребители, които биха използвали дадената услуга, продукт, сайт или бранд. Създаването на персоните помага за по-доброто разбиране на нуждите, опита, поведението и целите на потребителите.

Персоните са резюме на откритията, които са получени при фазата на проучването и се базират на тях.

За целите на проекта са създадени две персони. Първата е за човек, който е идеален кандидат за използване на приложението. Втората персона е на човек, за който е много малко вероятно да прояви интерес и да използва приложението за личен онлайн инструктор.



Tech Savvy **Trusting** **Adventurous**

Active

Goals

- Gain muscle mass.
- Maintain healthy life.
- Cliff jumping.

Frustrations

- Limitations due to Covid pandemic.
- Busy lifestyle.

Bio

Ivan is an online trader from Sofia, Bulgaria. He is 26 years old and has a cat called Toro. He grew up in a privileged family with no worries. He has made a lifestyle out of his occupation as trader and is always following the latest market news. He spends a lot of time in front of computers and likes things to be automated and easy to get done. He is moderately active - goes jogging and hiking sometimes. Recently he went out with a girl on a date and it seems like the spark flew between them. This is his motivation to get in better shape.

"Time is money."

Age: 26
Work: Online Trader
Family: Single
Location: Sofia, Bulgaria
Character: Energetic

Personality

Introvert ☐ Extrovert ☐

Thinking ☐ Feeling ☐

Lazy ☐ Hardworking ☐

Motivation



Fear ☐

Growth ☐

Power ☐

Social ☐

Brands & Influencers

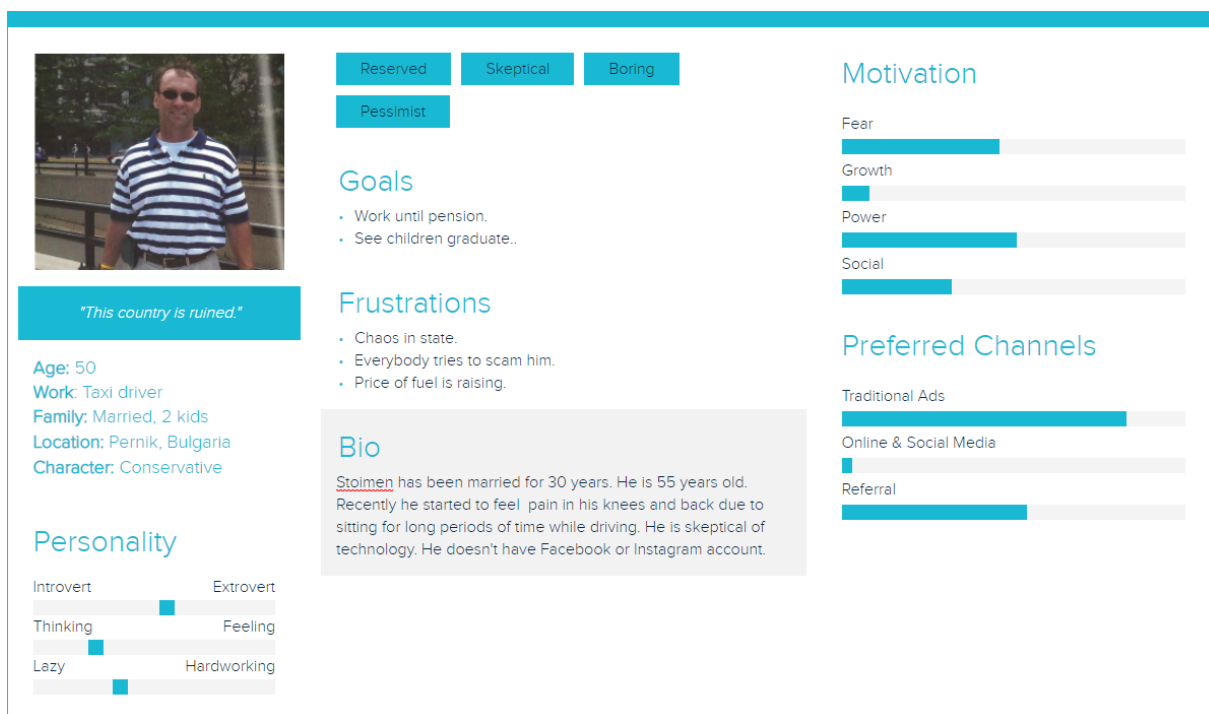
Preferred Channels

Traditional Ads ☐

Online & Social Media ☐

Referral ☐

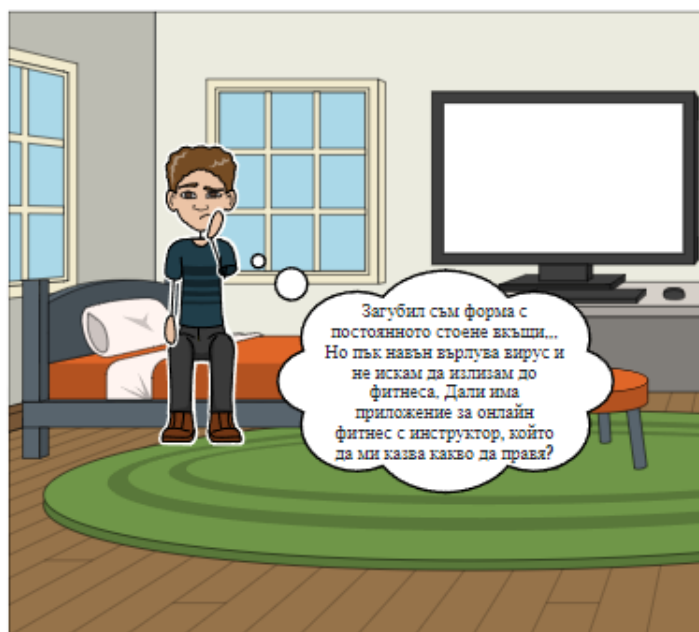
Influencers ☐



Storyboard

Storyboard-ът е инструмент за споделяне на една идея графично вместо в текстов вид. Той представлява определена поредица от изображения, която има логическа връзка и представя някаква история-предпоставка. Този начин на предаване на идеи се наблюдава, че е по-ефективен при комуникацията между хора, тъй като явно човешкият мозък възприема картини по-лесно отколкото абстрактен текст.

Storyboard-ът, изработен във връзка с текущия проект представя персонаж - млад мъж, който вследствие на противоепидемичните мерки прекарва по-голямата част от деня си у дома и не извършва почти никаква физическа дейност. Той много би искал да отиде във фитнес студио и да се възползва от услугите на фитнес инструктор, за да разбере как правилно да тренира така, че да влезе във форма по правилен начин. Следващият най-добър вариант за него е да потърси онлайн инструктор, с когото да комуникира чрез Интернет и който да му казва, какви упражнения и хранителен план са най-подходящи за него. Мъжът прави проучване в интернет и намира приложение за личен онлайн инструктор и следва инструкциите в него.



Use-case диаграма

Use-case диаграмата графично изображение на функционалностите, които един потребител може да достъпва в една система. Тя е първичната форма на изисквания към един софтуер. Този вид диаграма има задачата да отговори на въпроса “Какво?” (какво трябва да се имплементира), а не “Как?” (как трябва да се имплементира).

В текущата система има двама актьори - клиент и треньор. Някои от функционалностите са общи, докато други са свързани само с един от актьорите. Изображението по-долу представя графично функционалностите на системата за търсене на личен онлайн фитнес инструктор.

Налични са следните елементи на use-case диаграмата:

- Актьор - субект, който общува със системна функционалност. Може да е реален човек или външна система. Изобразява се с фигурна на човек. Актьорите в тази система за двама - клиент и треньор.;
- Use case - представлява една функционалност. Изобразява се с елипса. Пример за use case е търсене на треньор;
- Комуникационна връзка - Показва участието на един актьор в даден use case. Изобразява се с обикновена стрелка. Например актьорът “клиент” използва функционалността “търсене на треньор”;
- Връзка <<includes>> - указва, че функционалността в един use case е задължителна част от процеса на друг use case. Изобразява се със стрелка с пунктирна линия, надписана с “<<includes>>”. Пример за това в текущата диаграма е функционалността “промяна на клиент”, която включва “промяна на хранителен режим” или “промяна на тренировъчна програма”;



Essential use case диаграма

Essential use case диаграмата е разновидност на use case диаграмата. Нейната особеност е, че е тя има доста по-сбит вид - тя е симплифицирана, абстрактна и генерализирана use case диаграма. При системи, които са богати на функционалности, есенциалната use case диаграма може да е по-удачен избор, защото е по-лесна за възприемане от човешкото око, отколкото една голяма диаграма с преплитачи се линии.

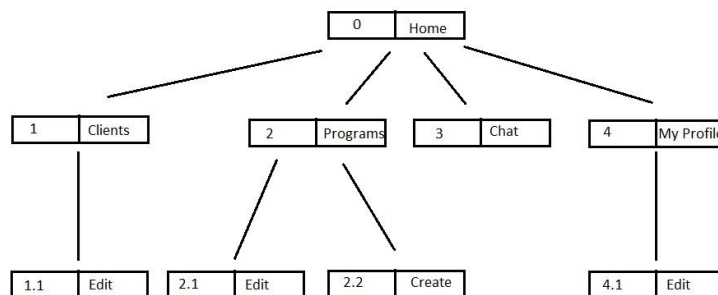
Дейност	Клиент	Треньор
Пращане заявка на треньор	Да	Не
Търсене на треньор	Да	Не
Въвеждане на прогрес	Да	Не
Преглед списък с клиенти	Не	Да
Промяна на клиент	Не	Да
Изготвяне на програми	Не	Да
Редактиране на личен профил	Да	Да
Чат	Да	Да
Вход в системата	Да	Да
Получаване на известие	Да	Да

Sitemap

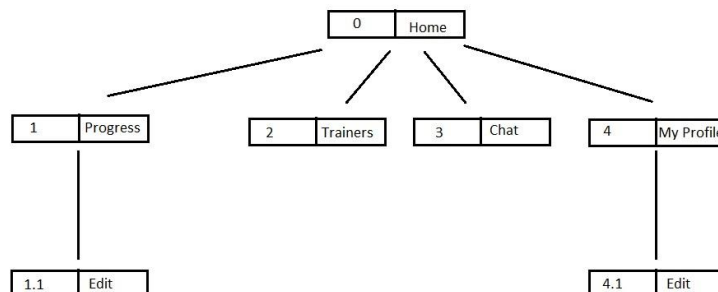
Sitemap-ът представлява йерархична диаграма на уеб сайт или приложение, която показва как страниците са приоритизирани, свързани и именувани. Sitemap-ът е един вид птичи поглед над структурата на уеб сайта.

По-долу са представени два sitemap-а. Първият е за това, което треньорите виждат, а вторият за гледната точка на клиентите.

Trainers



Clients



Core model

Core model представлява презентация на най-съществените функционалности на една система - тези, които потребителите основно ще използват, за да постигнат целта си. Използва се, за да се разбере как може да се помогне на потребителите да получат това, от което имат нужда от сайта.

Необходимо е да се определи бизнес целта и задачите на потребителя.

За целите на текущото приложение са изработени две core страници - за треньора и за клиента. Основната функционалност за един треньор е задаване и редактиране програма и хранителен план за клиент. Основната функционалност за един клиент е търсене на треньор и изпращане на заявка за да стане клиент на съответния треньор.

Core page: FitBook (Редактиране на клиент)

Netlife
Research

Business goals

Интерфейс за промяна диетата или програмата на клиент.

User tasks

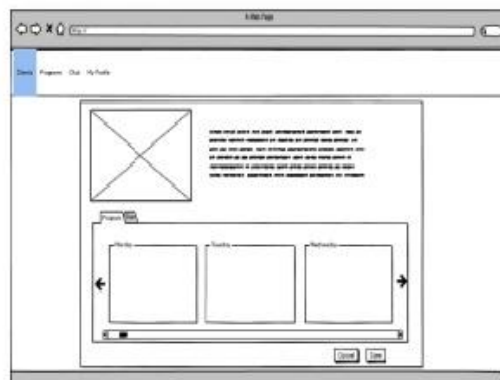
Да се промени диетата на клиент.

Да се промени програмата на клиент.

Inward paths

Екран Clients

Core content



Forward paths

Успешно редактиран клиент

Запис на промените

Core page: FitBook (Търсене на треньор)

Netlife
Research

Business goals

Предлага удобен и прост начин за търсене и свързване с треньор от страна на клиента

User tasks

Потребителят търси треньори като може да филтрира по име, адрес и пол.

Потребителят се свързва с треньора през чат функционалността или му изпраща заявка.

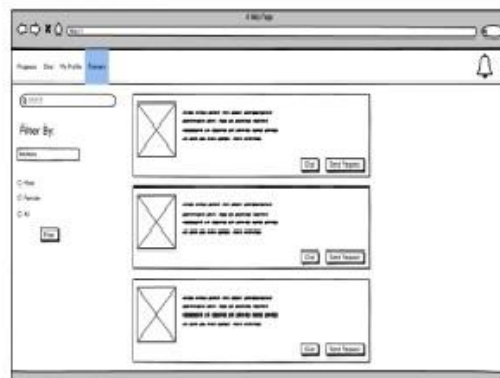
Inward paths

Начална страница

Регистрация

Подходящ Google search

Core content



Forward paths

Чат секция

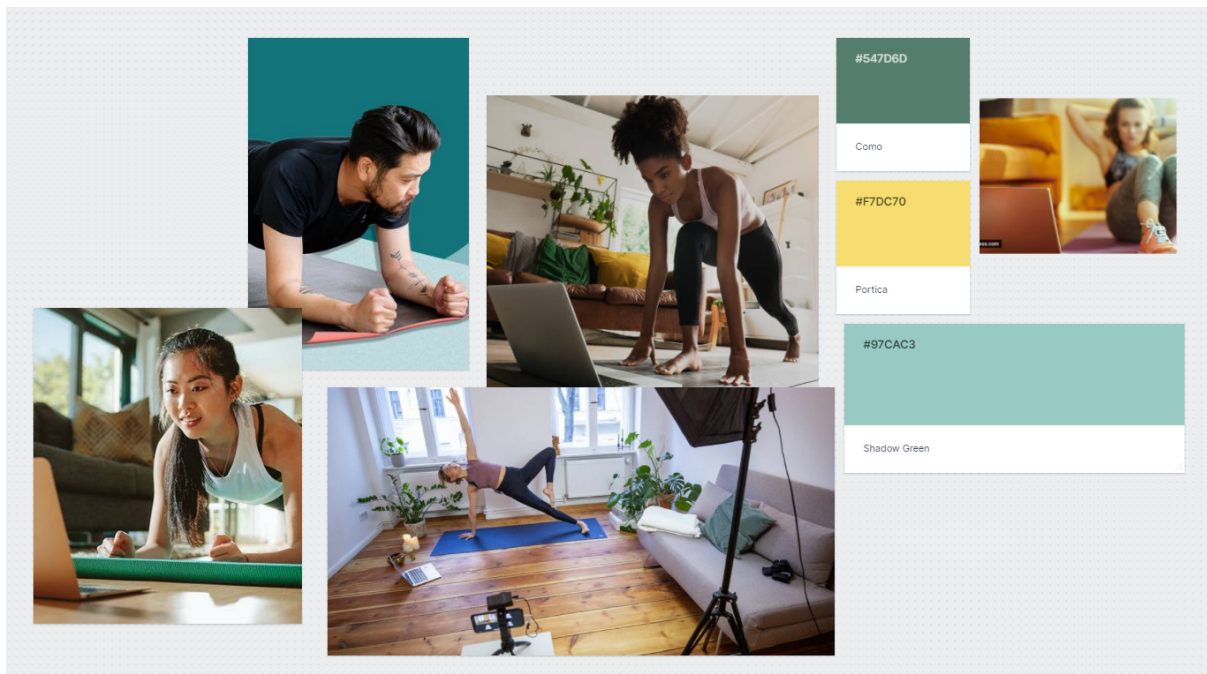
Успешно изпратена заявка

Профил на треньора

Moodboard

Moodboard е друг инструмент за визуално представяне на определена идея или дизайн. Този инструмент е един вид колаж, който може да съдържа изображения, текст, линкове и други подобни елементи.

Тази дъска се изработва от дизайнера за проекта и съдържа неговата идея за визията и излъчването на крайния продукт. Основна е цветовата гама. Различните цветови гами са подходящи за различни видове продукти. Например за продукти свързани с fitness и wellness е подходящо да се използват син, зелен, жълт цвят, тъй като те са витални, енергични и придават усещане за жизнерадост.

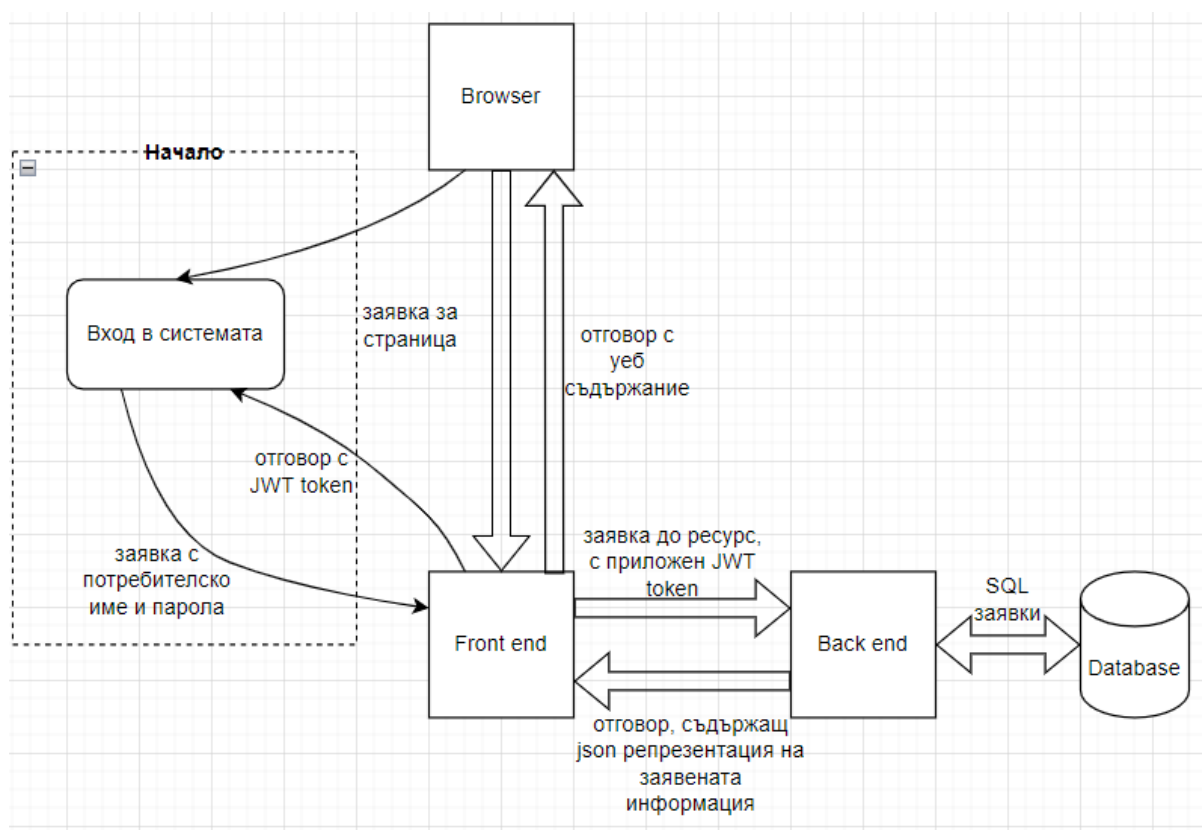


Реализация

Системата е разработена в две части - front end и back end.

Концептуален модел

Концептуалният модел е репрезентация на система и на начина, по който тя работи. Концептуалният модел на текущата система се състои от четири основни елемента - браузър, front end приложение, back end приложение и база данни. В правоъгълник с пунктирна линия е ограден важният момент от използването на системата, при който потребителят се автентичира и получава token за достъп, който се прилага при всяка последваща заявка. Този token се използва за определяне идентичността и правата на логнатия потребител.



Back end разработка

Сървърното приложение предоставя REST API на разположение за front end приложението. Вътрешната структура е на слоеве - презентационен слой, слой за бизнес логика и даннов слой. Чрез последния back end приложението се свързва с базата данни, където се съхраняват данните, и запазва, заявява и изтрива записи.

TrainerController:

GET

- /api/trainer/{id};
- /api/trainer/user/{id}/clients;
- /api/trainer/{trainer_id}/request/user/{client_id};
- /api/trainer/user/{trainer_id}/handle_request/{client_id};
- /api/trainer/chat_mates;
- /api/trainer/{id}/programs;
- /api/trainer/user/{id};

POST

- /api/trainer;

PUT

- /api/trainer/{id};

DELETE

- /api/trainer/client/{client_id};
- /api/trainer/program/{program_id};

ClientController:

GET

- /api/client/{id}/progress;
- /api/client/{client_id}/program/{program_id};
- /api/client/{client_id}/nutrition_plan/{program_id};
- /api/client/chat_mates;
- /api/client/user/{id};
- /api/client/user/{id}/program;

- /api/client/user/{id}/nutrition_plan;

POST

- /api/client;

PUT

- /api/client/{id};

LoginController:

POST

- /api/login;

RegistrationController:

POST

- /api/registration;

NutritionPlanController:

GET

- /api/nutrition_plan/{id};

POST

- /api/nutrition_plan;

PUT

- /api/nutrition_plan/{id};;

ProgramController:

POST

- /api/program/user/{id};

PUT

- /api/program/{id};

ProgressController:

POST

- /api/progress;

PUT

- /api/progress/{id};

ChatController:

GET

- /api/chat/{id}/messages/{page};

POST

- /api/chat/message;
- /api/chat;

ExerciseController:

GET

- /api/exercise;

NotificationController:

GET

- /api/notification/{id};

В приложението има два типа роли - треньор и клиент. Endpoint-ите, които са позволени само за една от тях, са защитени чрез способите за сигурност на Spring. Автентикацията се базира на JWT токени.

Това е начин за споделяне на информация, свързана със сигурността, между две страни - клиент и сървър. Един токен се състои от заглавна част, където се указва типа алгоритъм за криптиране. След това от тяло, където могат да се слагат всякакви двойки данни ключ-стойност, които служат за автентикация и идентифициране на потребителя.

Автентикацията при комуникация по WebSockets протокола също е конфигурирана да използва JWT токени и използва Principal за мапване на потребител със сесиен идентификатор при свързване по този канал.

В Приложение № 1 е описан алгоритъмът за попълване на липсващи ProgramParts от един Program обект. Ако при създаване на програма някой ден от седмицата бива пропуснат, защото няма упражнение на дадения ден, то тогава празниците се попълват така, че винаги списъкът от ProgramParts на обекта Program да съдържа 7 елемента - по един за всеки ден от седмицата. Това значително улеснява изобразяването на информацията във front end-а.

Front end разработка

Front end частта е разработена на Vue с използване на библиотеките Vuex и Vue Router съответно за state-management и навигация.

- Vuex - използван е именуван store, който е разделен на модули за по-добро управление. Най-важният модул е auth, който пази сесиинна информация(userId, JWT token). Другите модули са clients и trainers, които пазят информация съответно за клиенти и треньори;
- Vue Router - една от по-важните функционалности на vue-router е че позволява лесно да се показва нужната навигация. Имплементиран е филтър, с който не се позволява на неоторизирани потребители да достъпват изгледи, които са предназначени за регистрирани потребители;

“Страниците” на приложението са поместени в /src/pages, а компонентите в /src/components

- Страници;

Приложението предоставя следните страници на потребителите:

- Chat;
- Clients;
- EditClient;
- MyProfile;
- Programs;
- Progress;
- Trainers;
- Login;
- Registration;
- Компоненти:
 - AddProgram;
 - ChatCard;
 - ClientCard;
 - EditProgram;
 - NotificationCard;
 - Notifications;
 - ProgramCard;
 - ProgressCard;
 - TrainerCard;
 - TheHeader;
 - App;

Компонентът “корен” е App, където с помощта на Vue Router се събират всички останали страници и компоненти. Свързването на Vue инстанцията с главния HTML файл се случва в main.js, където също се указва на

приложението, че трябва да използва Vuex и Vue Router. За стилизиране се използва технологията Tailwind CSS, чийто конфигурационен файл е `tailwind.config.js`.

Тестове и резултати

В процеса на интеграция между front end и back end напълно естествено възникнаха проблемни ситуации, които изискваха тяхно сътрудничество между двете страни. Основните трудности, които бяха срещнати и тяхното решение са разгледани в следните две подглави.

WebSockets и Spring Security

Съчетаването на тези две технологии се оказа не толкова лесно. За да може заявките по WebSockets да бъдат оторизирани, трябваше да се имплементира custom interceptor (Приложение № 2), който да извлича JWT токена от header-ите на заявката и да слага потребителя в контекста на сигурността.

Решението беше достигнато на принципа проба - грешка. Като резултат само оторизирани потребители могат да се свързват с приложението чрез WebSockets протокол.

Препращане на данни през компонентите в VueJS

Когато се разделят елементите на компоненти с цел атомизация и използване, има нужда по добър начин да се управлява препращането на данни през компонентите. Това става по-трудно, когато има няколко вложени нива на компоненти.

Едно решение е “prop drilling”, което не е най-добра практика, но е просто за имплементация. Това е експлицитно предаване на данни през компонентите.

Интеграцията на SockJS и Spring WebSocket

Получаването на съобщения през WebSocket не се получи от първия път. Трябваше да се пробват много различни варианти във front end и в back end, докато една от комбинациите не проработи. Решението беше достигнато, чрез гледане на примери от примерни проекти в Интернет.

Заклучение

В заключение може да се каже, че системата за намиране на онлайн фитнес инструктори би намерила приложение в реалния свят.

Такова приложение се търси от потребителите, заради ситуацията на световна пандемия и последиците от нея, като ограничения на посетители на закрити места, в това число - фитнес зали, танцови студия и т.н. Липсата на физическа активност води до много лоши последици за здравето. Затова решението за фитнес у дома, под надзора на компетентно лице - фитнес инструктор, изглежда, че е перфектно с оглед обстоятелствата.

Тази система е насочена към по-младата част на населението, защото точно тя е тази, която има повече желание и енергия за спорт, също така и борави по-добре с технологиите.

Впечатлението от един уеб сайт за онлайн фитнес инструктори трябва да положително - за това помага избраната цветова гама от светли, жизнерадостни цветове. С това и с други инструменти за дизайн на потребителски интерфейс, се постига резултата на едно задоволително приложение, което помага на потребителите да изпълнят целта, която имат - а именно да спортуват.

Въпреки, че имплементацията на системата е предизвикателна, тя има още място за доработване, подобрене и добавяне на нови функционалности, като например поточно видео предаване на живо, качване на видео, възможност за рейтинг на треньорите, онлайн разплащания и т.н.

С тези функционалности, системата има потенциал да се наложи на пазара, и да привлече треньори и трениращи не само в България, но и в останалите страни в Европа и извън нея.

Приложение

Приложение № 1

```
@Component
public class ProgramPartUtil {

    public List<ProgramPart>
transformProgramParts(List<ProgramPart> programParts) {
        if (programParts.size() > 7) {
            throw new RuntimeException("A program cannot
have more than seven parts");
        }

        programParts =
programParts.stream().sorted(Comparator.comparing(ProgramP
art::getWeekDay)).collect(Collectors.toList());

        ProgramPart[] programPartsArr = new ProgramPart[7];

        programParts.forEach(programPart -> {
            switch (programPart.getWeekDay()) {
                case MONDAY: programPartsArr[0] =
programPart; break;
                case TUESDAY: programPartsArr[1] =
programPart; break;
                case WEDNESDAY: programPartsArr[2] =
programPart; break;
                case THURSDAY: programPartsArr[3] =
programPart; break;
                case FRIDAY: programPartsArr[4] =
programPart; break;
                case SATURDAY: programPartsArr[5] =
programPart; break;
                case SUNDAY: programPartsArr[6] =
programPart; break;
            }
        });
    }
}
```

```

        for (int i = 0; i < programPartsArr.length; i++) {
            if (programPartsArr[i] == null) {
                programPartsArr[i] =
programPart(weekDay(i));
            }
        }

        return Arrays.asList(programPartsArr);
    }

    private ProgramPart programPart(WeekDay weekDay) {
        ProgramPart programPart = new ProgramPart();
        programPart.setWeekDay(weekDay);
        return programPart;
    }

    private WeekDay weekDay(int i) {
        switch(i) {
            case 0: return WeekDay.MONDAY;
            case 1: return WeekDay.TUESDAY;
            case 2: return WeekDay.WEDNESDAY;
            case 3: return WeekDay.THURSDAY;
            case 4: return WeekDay.FRIDAY;
            case 5: return WeekDay.SATURDAY;
            case 6: return WeekDay.SUNDAY;
            default: throw new RuntimeException("Invalid
number of week day");
        }
    }
}

```

Приложение № 2

```
@Override
public void
configureClientInboundChannel (ChannelRegistration
registration) {
    registration.interceptors(new ChannelInterceptor() {
        @Override
        public Message<?> preSend(Message<?> message,
MessageChannel channel) {
            StompHeaderAccessor accessor =
MessageHeaderAccessor.getAccessor(message,
StompHeaderAccessor.class);
            if
(StompCommand.CONNECT.equals(accessor.getCommand())) {

Optional.ofNullable(accessor.getNativeHeader("Authorizatio
n")).ifPresent(ah -> {
                String token = ah.get(0);
                if (token != null) {
                    token = token.substring(7);
                    String email =
jwtTokenUtil.getEmailFromToken(token);
                    User user =
userService.findByEmail(email);
                    UsernamePasswordAuthenticationToken
authentication =
                        new
UsernamePasswordAuthenticationToken(user,
user.getPassword(),
Collections.singletonList(user.getRole()));
                    accessor.setUser(authentication);
                }
            });
        }
        return message;
    }
});
}
```