ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

по професия код 481020 „Системен програмист“
специалност код 4810201 „Системно програмиране“

Тема: Уеб сайт за следене на личните разходи

Дипломант:
Александра Веселинова
Александрова

Дипломен ръководител:
Венко
Николов

СОФИЯ

2 0 2 3

Отзив на дипломен ръководител

В настоящата дипломна работа е реализиран уеб сайт за следене на лични разходи, като са спазени всички функционални изисквания от заданието.

Потребителският интерфейс е удобен и подходящ за задачата на дипломната работа, а сървърната част на проекта е добре организирана и логически построена, следвайки добрите практики. Базата от данни е съобразена с нуждите на приложението и добре структурирана. Технологичният стек за изграждане на уеб сайта се състои от модерни, но едновременно с това доказали се и широко използвани технологии, които са и много подходящи за проекта (Spring Boot 3.0.0, Spring Core 6.0.2, Spring Data JPA 3.0.0, Spring ORM 6.0.0, Spring JDBC 3.0.0, Hibernate core 6.1.5.Final, HIbernate Validator 8.0.0, Spring Security 6.0.0, Tomcat core 10.1.1, H2 Database 2.1.214, Thymeleaf 3.1.0, Mapstruct 1.5.2.Final, Spring Boot Starter-Mail 3.0.0).

Дипломантът подходи сериозно към задачата си и се справи с реализацията на дипломната работа своевременно, като по време на разработката получи нови знания и приложи вече придобити такива в училище.

Дипломната работа е написана професионално, добре документирана и оформена е, а с проекта е реализирана напълно работеща система, която може да се използва в реалния живот.

Убеден съм, че дипломната работа е много добра и може да бъде защитена пред комисията.

Предлагам Мария Камбурова за рецензент на дипломната работа.

Увод

Развитието на консуматорската култура започва още през XV-XVI век в Англия. По това време се заражда разбирането, че човек трябва да си купува не само това, което му е необходимо, за да преживее, но и неща, които правят живота по-„хубав“.

През XVIII век населението купувало онова, което не можело да си произведе, на ежеседмични или ежегодни пазари. Последвалите успешно развитие на индустрията и увеличаване на работните места в Англия били причините за увеличението на заплатите, което било предпоставка за бързото скачане на интереса на гражданите към стоки от типа на чай, сапун и нови дрехи.

След Втората световна война дотогава рядко срещаните и считани за „луксозни“ продукти станали достъпни за масите. Пред 50-те години на XX век на европейския и американския пазар се появили стоки от всички краища на света, а тогава започва и глобализацията в продажбите. Така в следващите десетилетия станал бум на търсенето на електрически уреди, мебели с художествена стойност, който бил последван от мания към произведения на изкуството. Богатството и красотата придобиват все по-голямо значение дори и в наши дни. Много от стоките, които се купуват по внушението на рекламата на медиите, не намират непосредствено приложение в бита и всъщност са излишни [1].

Но ново измерение на консуматорското общество се открива след навлизането на интернет поради облекчаването и ускоряването на пазаруването онлайн. Така стигаме до настоящия момент, когато всеки един от нас бива заливан ежедневно от реклами за най-разнообразни продукти и услуги и е на няколко клика разстояние от покупка, където и да се намира - вкъщи, на работното място, на почивка. С дигитализацията на парите, банкови карти и дори онлайн валути, онлайн пазаруването е по-лесно

отвсякога. Това се дължи на психическа “заблуда” - когато купуваме нещо с физически банкноти, имаме усещането, че ги “губим”, защото ги даваме с ръката си. Вече не са в нас. Но при разплащането с карта, тази физическа липса изчезва. Затова хората са много по-склонни да харчат по-често и по-големи суми онлайн. В последните години все повече учени наблюдават пристрастяване към онлайн пазаруването и се обмисля записването на това състояние като отделна пхисическа болест BSD: Buying-Shopping Disorder.

Така важно умение за модерния човек е той да бъде финансово грамотен и отговорен. Не е навик, лесен за играждане, но е изключително полезен. Тъй като всеки от нас има устройство в себе си през целия ден, независимо дали това е телефон, лаптоп, таблет или нещо друго, то най-лесно и интуитивно за нас би било да следим своите покупки с негова помощ.

Именно това е целта на настоящата дипломна работа - да бъде разработен уеб сайт, в който съвременният човек да може да записва своите разходи и да получава статистика за тях, с което да изгради финансова дисциплина.

Първа глава
Методи и технологии за реализиране на уеб приложения

1.1. Основни принципи и понятия в изграждането на уеб приложения

1.1.1 Уеб страница и уеб сайт

Уеб страницата е начин за показване на информация в интернет. Една уеб страница се състои от елементи като текст, изображения, хипервръзки, видеоклипове, бутони. Въз основа на информацията, която тези страници съдържат, те са организирани в информационна йерархия – това позволява навигация от една страница към друга. Цялостната колекция от тези свързани помежду си уеб страници е уеб сайт. Уеб сайтът, в основата си, е просто начин за публично събиране и показване на информация. Без значение колко сложен става един уебсайт и колко страници съдържа в себе си, всичко се свежда до тази основна цел. Ако направим аналогия с книга: точно както различните книги могат да имат различен брой страници, различните уеб сайтове също имат различен брой уеб страници, засягат различни теми и се използват за информиране на хората.

1.1.2 Уеб хост и хостинг сървър

За да може един уеб сайт да бъде достъпен в интернет, то той трябва да съществува някъде на компютър. Това води до задължителното поддържане на устройства с постоянна интернет връзка, където да се “пази” сайтът и данните, които са му нужни, за да работи. Но тъй като да се съхранява уеб сайт и информацията за него на нашия компютър, който да е постоянно свързан с интернет, е доста скъпо и непрактично, са създадени хостинг услугите. Хостинг услуга е предоставянето на място за уеб сайт,

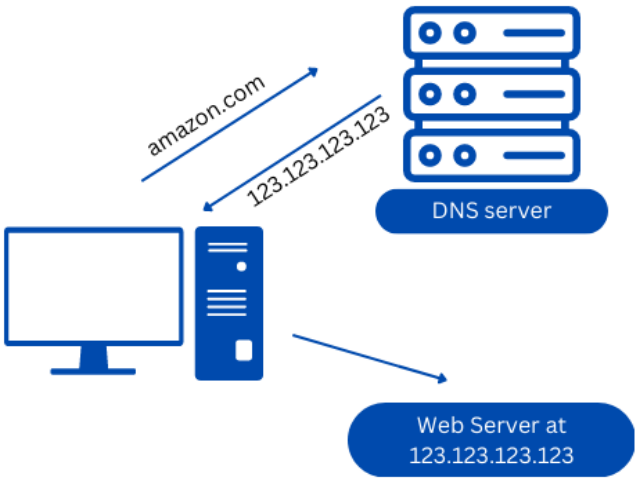
където да съществува и да бъде достъпен през 24-часов интервал от време всекидневно.

1.1.3 Домейн

След като един уеб сайт съществува някъде, то той трябва да бъде и достъпван. За тази цел се използват домейн имена. Всеки уебсайт има адрес, поредица от цифри в даден формат, известен като IP адрес. Тъй като не е възможно да се запомнят адреси в цифровия им формат, по които всеки от нас да достъпва сайтове, са измислени домейн имената. Правата над тях се държат от собственика на уеб сайта и се заплащат. Всички домейни се регистрират при регистратор на домейни.

1.1.4 DNS сървър

DNS съкращението идва от Domain Name System и означава система за домейн имена. DNS сървърът е именно “преводача”, който отговаря за транслирането на домейн името към съответния IP адрес. На фигура 1.1 е показана схема на принципа на действие на DNS сървърите.



Фиг. 1.1

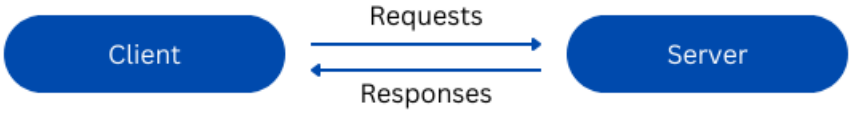
1.1.5 Браузър

Уеб браузърът е инструмент за посещаване на различните сайтове в интернет. Той извлича информация от други части на мрежата и я показва на работния плот или на мобилно устройство. Информацията се прехвърля с помощта на Hypertext Transfer Protocol, който определя как текст, изображения и видео се предават в мрежата. Тази информация трябва да се споделя и показва в последователен формат, така че хората, използващи всеки браузър, навсякъде по света, да могат да я видят. Не всички браузъри обаче представят еднаква информация по един и същи начин. За потребителите това означава, че един уебсайт може да изглежда и функционира по различен начин в зависимост от браузъра, през който се извършва достъпването. За да стане по-лесно използването на интернет, съществуват уеб стандарти, които дефинират допустимите разлики между браузърите.

1.1.6 Клиент - Сървър модел

Световната мрежа е клиент-сървър модел, което означава, че работният принцип на всички уеб сайтове се уповава именно на тази норма. Моделът клиент-сървър е разпределена структура на приложение, която разделя задачата или работното натоварване между доставчиците на ресурс или услуга, наречени сървъри, и рикуестъри на услуги (поискващи услугите), наречени клиенти. В архитектурата клиент-сървър, когато клиентският компютър изпрати заявка за данни до сървъра през интернет, сървърът приема заявения процес и доставя заявените пакети данни обратно на клиента. Клиентите не споделят никакви свои ресурси.

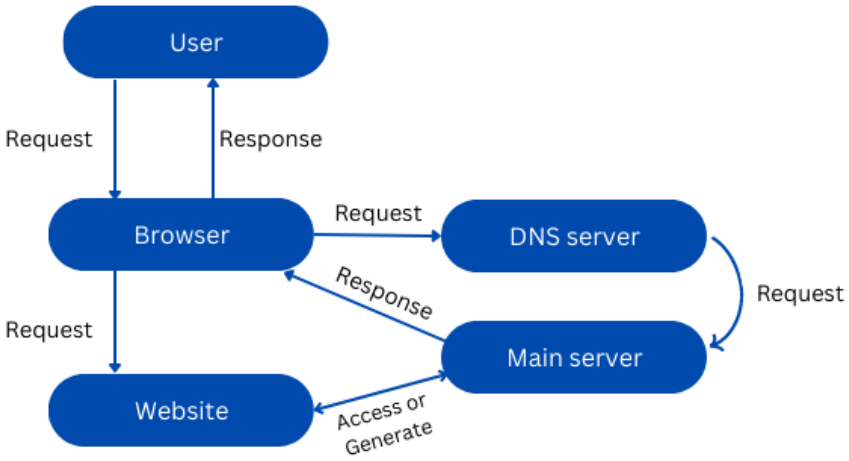
Всеки компютър в мрежата изпълнява функцията или на клиент, или на сървър. Опростена диаграма за това как те взаимодействат е показана на фиг. 1.2:



Фиг. 1.2

Клиентите са свързаните с интернет устройства на типичния уеб потребител. Например компютър, свързан към Wi-Fi мрежа, или телефон, свързан с мобилна мрежа, и софтуер за уеб достъп, наличен на тези устройства, който обикновено представлява браузър за даденото устройство - Safari, Firefox, Chrome и т.н.

Сървърите са компютри, които съхраняват уеб страници, сайтове или приложения. Когато клиентско устройство иска достъп до уеб страница, копие на уеб страницата се изтегля от сървъра на клиентската машина, за да се покаже в уеб браузъра на потребителя. Цялостната схема е представена на фигура 1.3.



Фиг. 1.3

1.2. Съществуващи решения

1.2.1 Expensify

Expensify е облачно-базиран (cloud-based) инструмент за проследяване на бизнес финанси, който помага на компаниите да управляват разходите си и да рационализират процесите си за отчитане. С Expensify служителите могат лесно да подават отчети за разходите си, да проследяват таксуваните часове и да управляват пътните разходи от своя настолен компютър или мобилно устройство. Expensify предлага разнообразие от функции, предназначени да опростят процеса на управление на разходите. Потребителите могат лесно да сканират и качват разписки с помощта на камерата на своя смартфон, тъй като Expensify използва SmartScan за документи (умно сканиране на документи, например касови бележки), елиминирайки необходимостта от ръчно въвеждане на данни. Софтуерът може също така автоматично да категоризира разходите, въпреки че се дава възможност и ръчно да се определи категорията на даден разход, и да ги присвои към правилната потребителска сметка, спестявайки време и намалявайки значително грешките. Expensify се интегрира с различни счетоводни и платежни системи, включително QuickBooks, Xero и PayPal, което улеснява синхронизирането на данни за разходите между платформите. Софтуерът също така предлага функции като автоматично конвертиране между отделни валути и многостепенни работни процеси за одобрение, което го прави цялостно решение за фирми от всякакъв размер. Expensify предлага възможност да се проследи дори и пробегът на служителите - приложението се свързва с GPS-а на мобилното им устройство и събира данни за шофирането им.

Като цяло тази услуга може да помогне на компаниите да намалят времето и усилията, необходими за управление на разходите, като същевременно осигурява по-голяма видимост и контрол. [2]

1.2.2 GoodBudget

Goodbudget е софтуер за личен бюджет, който позволява на потребителите да управляват своите финанси и да проследяват разходите си. Базовият метод на работа на приложението е системата на плика (envelope method).

Тази система се основава на традиционната система с пликове, при която се разпределят средства за конкретни категории и след това се използват пари от всеки плик за плащане на разходи в тази категория. Goodbudget разгръща тази концепция в цифров вариант, позволявайки на потребителите да създават виртуални пликове за различни разходи, като наем, хранителни стоки, развлечения и други. След това потребителите могат да определят бюджет за всеки плик и докато харчат пари, могат да проследяват разходите си в реално време. Приложението показва визуално представяне на плика и каква сума е оставанала за месеца. Идеята е да бъде по-лесно планирането на финансите и спестяването на пари за определена цел.

Другата основна функционалност на GoodBudget е споделянето: Профилите на всички в едно семейство се синхронизират, за да може всички да виждат общите си категории като хранителни стоки. Вместо двамата родители да имат отделни пликове за тази категория, то семейството споделя един общ плик, с което става по-отчетливо и ефективно планирането на семейния бюджет. GoodBudget дава възможност и за създаване на пликове за дълг и за големи разходи. Потребителят може да планира изплащането на свой дълг, заедно с планирането на останалите си пликове или да дефинира спестяване на големи суми пари, например за закупуване на автомобил или организиране на скъпа екскурзия. Goodbudget е достъпен в мрежата, както и на мобилни устройства с iOS и Android операционни системи. [3]

Втора глава
Проектиране на структурата на уеб сайт за следене на личните разходи

2.1 Функционални изисквания

Сайтът представлява личен дневник за управление на разходи. Графичният потребителски интерфейс следва да бъде прост и лесен за използване. Първо и много важно изискване е всеки потребител да има свой собствен профил, който да не бъде публичен, потребителите не могат да достъпят профил, различен от собствения. Един профил се дефинира от потребителско име, парола и имейл. Регистрацията изисква и трите споменати, а потребителят влиза в профила си с потребителско име и парола. С цел сигурност, паролите трябва да не се запазват в чист текстови формат, а криптирани.

След като потребителят успешно влезе в своя профил, той получава достъп до информацията за своите разходи. Има възможност да добави нов разход. Разходите се класифицират по два основни признака. Първият признак е продължителността на разхода. Според него те се делят на еднократни или абонаментни. Пример за еднократно плащане е покупки от супермаркет, а пример за абонаментен разход е абонамент в музикалната платформа Спотифай. Вторият признак е категорията на разхода. Всеки разход трябва да има категория, която потребителят определя при въвеждането му. Не трябва да се допуска разход без категория.

Спрямо гореспоменатите два признака се изважда персонализирана статистика. В нея потребителят намира информация за своите разходи в в последния месец и в последната година, разделени на категории. Дава му

се възможност да направи собствена филтрация на разходи според избрана категория и период от време.

Последната функционалност е изцяло свързана с абонаментните разходи и тя представлява имейл известие 3 дни преди крайната дата на потребителския абонамент.

2.2 Избор на технологии

2.2.1 База от данни

В продължение на почти 40 години системите за управление на релационни бази данни (RDBMS - Relational database management systems) са предпочитаната опция за съхранение на информация в бази данни, главно за лични данни, финансова информация и производствени записи, наред с други случаи. Това прави RDBMS подходящи за сайт за управление на личните разходи. Релационният модел за управление на база данни използва табличен формат за съхраняване на данни, което го прави различен от стандартния модел за управление на данни - поставяне на всички данни на едно място.

Днес всички големи релационни бази данни използват SQL като език за заявки. Въпреки че има стандартен SQL, повечето платформи за бази данни могат да включват специфични допълнителни функции.

MySQL

MySQL е open-source система за менажиране на релационни бази данни, създадена от Oracle, и базирана на SQL като стандартен език за заявки. В настоящия момент тя е най-популярната и често използвана RDBMS - приложения като Facebook, Netflix, Twitter, Uber, Airbnb, LinkedIn и други използват MySQL за организацията на своите данни. MySQL е базиран на SQL, но не имплементира пълния SQL стандарт. Типовете данни

са стандартните, не се поддържа наследяване на таблици и има лимит на тригерите върху команди. Може да се използва с Java JDBC, C/C++, Python, PHP, Node.js и други. MySQL има вградени функции за сигурност. [4]

PostgreSQL

PostgreSQL е най-модерната RDBMS, която, подобно на MySQL, е open-source. Тази система е подходяща и използвана за големи системи, където скоростите на четене и запис са от решаващо значение и изискват изпълнение на сложни заявки. PostgreSQL имплементира стандартния SQL и може да бъде допълван. Типовете от данни са стандартните, както и масиви, hstore (тип за запазване на двойки key-value), самостоятелно дефинирани типове данни. PostgreSQL позволява наследяване на таблиците и предоставя възможност за тригъри върху командите, които нямат глобален ефект. Поддържа набор от програмни езици, някои от които са C/C++, Java, R, Perl, Python, JavaScript и има вградени функции за сигурност. [5]

H2 Database Engine

H2 database engine също е open-source система за менажиране на бази от данни, базирана на SQL стандарта. Тя е направена специфично, за да се използва с Java чрез JDBC API. Двете й основни предимства пред останалите системи са бързина и лекота. Не се налага да се инсталира допълнителна конзола за контрол над таблиците, тъй като има вградена конзола в браузъра. Системата използва мултиверсионен контрол на паралелността (MCC или MVCC) и транзакционна памет. [6][7] Мултиверсионния контрол над паралелността е метод за оптимизация на базите данни, който създава копие на базата от данни с цел позволяване на едновременното четене и писане. Транзакционната памет се използва при паралелни операции (четене и запис), като позволява група от инструкции

за зареждане и съхраняване да се изпълняват като атомарни функции, тоест не могат да бъдат прекъснати и няма загуба на информация. H2 database engine поддържа използването на PostgreSQL ODBC драйвер, като по този начин може да бъде използвана като PostgreSQL сървър за по-сложни приложения.

Избрах H2 database engine като система за менажиране на базите от данни, тъй като е най-подходяща за Java приложение, може да бъде използвана и като PostgreSQL сървър за по-сложни заявки и е най-бърза и удобна.

2.2.2 Бекенд

Spring Boot

Spring Boot е популярна open-source рамка (framework) за разработване, базирана на програмния език Java, който е език от високо ниво. Spring Boot рамката е базирана на Spring рамката.

Основен недостатък на приложенията, разработени със Spring, е, че интегрирането, тестването и конфигурирането на приложението отнемат прекалено много време, тъй като са сложни, също така няма настройки по подразбиране при инициализацията на приложението, което намалява значително продуктивността на екипа. Затова се взима решение да се напише рамка, която да реши тези проблеми и именно това е основната идея на Spring Boot. Поради факта, че Spring boot е изграден върху Spring, тя е много добре интегрирана с екосистемата на Spring, която предлага широк спектър от зависимости (dependencies), позволяващи разработката на различни основни функционалности. [8]

Всяко Spring Boot приложение е самостоятелно, бързо за конфигурация и разработване. То съдържа в себе си HTTP сървър,

най-често това е Apache Tomcat (или накратно Tomcat), и автоконфигурация, която позволява бързото структуриране на приложението и може да бъде персонализирана според нуждите на проекта. На фигура 2.1 е показана основната структура на рамката:

Фиг. 2.1

Обобщено предимствата на Spring Boot са:

- Автоконфигурация на ключови компоненти на приложението;
- Може да се стартира без да бъде разгърнато (deploy-нато) на сървър
- Избира самостоятелно кои настройки по подразбиране да бъдат приложени, спрямо избраните от разработчика зависимости при инициализиране;
- Използва популярен и добре документиран език - Java;
- Осигурява избягването на повтарящ се код (boilerplate code), аннотации и XML конфигурации;
- Поддържа Spring зависимости (dependencies) като Spring Security, Spring Web Services, Spring JDBC, Spring ORM, Spring Data и други;
- Съвместимо е с инструменти като Gradle и Maven за по-удобна разработка и тестване на web приложения;
- Поддържа много плъгини за различни видове бази данни, които са лесни за използване, но и позволяват собствена конфигурация. [9]

Основни други технологии, които използвам в комбинация с тази рамка са:

Spring MVC

Spring MVC е рамка за изграждане на уеб приложения в Java. Тя е част от по-голямата Spring рамка и често се използва заедно с други Spring компоненти като Spring Security, Spring Data и Spring Boot.

Spring MVC предоставя архитектура модел-изглед-контролер (MVC - model-view-controller) за уеб приложения. Този модел разделя приложението на три основни компонента: модел, който представлява данните и бизнес логиката, изгледът, който представя данните на потребителя, и контролерът, който обработва въвеждането от потребителя и съответно актуализира модела и изгледа.

В Spring MVC приложение HTTP заявките първо се обработват от диспечерски сървлет. Диспечерският сървлет е отговорен за маршрутизирането на заявки към съответния контролер въз основа на заявения URL адрес. След това контролерът извлича данни от модела и определя кой изглед трябва да се използва за показване на данните. Изгледът обикновено се реализира с помощта на машина за шаблони като Thymeleaf, JSP или Velocity.

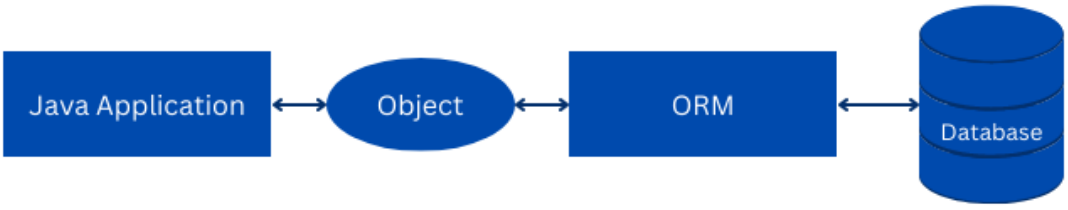
Spring MVC също предоставя редица функции, които улесняват справянето с обичайните задачи на уеб приложенията. Например, включва поддръжка за обработка на подавания на формуляри, валидиране на въведените от потребителя данни и управление на потребителски сесии.

Spring Data JPA & Hibernate

Spring Data JPA е компонент на Spring Framework, който предоставя изпълнение на спецификацията JPA (Java Persistence API), който сам по себе си представлява набор от правила, изразяващи се в класове и интерфейси, които да бъдат приложени при работа с данни. Spring Data JPA позволява на разработчиците да работят с релационни бази данни,

използвайки обектно-ориентиран API на високо ниво и предоставя удобни абстракции за обработка на общи задачи.

Реализация на Spring Data JPA се изразява в инструмента Hibernate. [10] Това е инструмент, който се използва за ORM - object-relational mapping, което само по себе си представлява преобразуването на даден обект от приложението към запис от базата данни. Hibernate е бърз и лек, не е нужно да пишем самостоятелно SQL заявки, позволява създаването на таблици и прави присъединяването на различни таблици по-лесно. Моделът на работа на ORM инструмент е показан на фигура 2.2.



Фиг. 2.2

Основните предимства на Spring Data JPA се изразяват в:

- Хранилища (repositories) - моделът на хранилищата е един от най-често използваните. Не е сложен за собствена имплементация, но няма смисъл, а и би изгубило време. Дефинирането на хранилище с помощта на Spring Data JPA е бързо и много лесно - създаваме интерфейс или абстрактен клас, който да разшири някой от трите интерфейса на Data JPA и с това комуникацията към базата данни е готова;
- Spring Data JPA самостоятелно генерира и скрива имплементацията на основните операции, свързани с данни и използването им, като се уточнява типа обект, с който работи хранилището. Най-често към всяка таблица (и съответния й модел) се прави хранилище. Някои от готовите функции са save(), delete(), findAll(), deleteAll() и други;

- Генериране на персонални заявки към базата данни - Spring Data JPA генерира имплементацията на заявки според имената на функциите в хранилището. За да няма грешки при генерирането, трябва да не са прекалено сложни заявките и да следват конвенция за име на функцията - започват с “findBy”, използват camelCase форматиране, като трябва параметрите (ако има такива) да съответстват с имената на атрибутите на обекта-модел (които от своя страна съответстват на колони от таблицата). [11]

Spring Security

Spring Security е рамка, която осигурява удостоверяване, оторизация и други функции за сигурност за приложения, базирани на Spring. Тя е много персонализирана и може да бъде интегрирана с широк набор от механизми за удостоверяване, като LDAP, OAuth и OpenID, както и персонализирани механизми за удостоверяване.

Spring Security се основава на филтърна верига, която прихваща входящи заявки и извършва проверки за сигурност. Филтриращата верига е силно конфигурируема и позволява на разработчиците да включват персонализирани филтри и механизми за удостоверяване. Spring Security също така предоставя набор от анотации, които могат да се използват за защита на контролери, методи или URL адреси, както и поддръжка за контрол на достъпа, базиран на роли и разрешения.

Apache Tomcat

Apache Tomcat е open-source уеб сървър. Сам по себе си Apache Tomcat обединява Jakarta Servlet, Jakarta Server Pages, Jakarta Expression Language, Jakarta WebSocket, Jakarta Annotations и Jakarta Authentication specifications, които са част от Jakarta EE платформата, развит вариант на Java EE [12]. Представлява контейнер за Java web приложения, осигуряващ

среда, в която да се изпълняват. Apache Tomcat има висока надеждност (High Availability) и може да продължава да обслужва заявки докато се извършва актуализиране на версията му. Той поддържа още и работа в клъстер (cluster). Това прави възможно разпределяне на товара (load balancing) на различни инстанции на сървъра с цел да се повиши броят на обслужваните потребители. Поради тези причини Apache Tomcat е избран за технология, която ще изпълнява ролята на контейнер, изпълняващ Java приложението.

Maven

Maven е популярен open-source инструмент за автоматизиране на изграждането на проекти. Разработен е от компанията Apache Group. Основната му идея е за изграждане, публикуване и внедряване на няколко проекта наведнъж за по-добро управление на проекти. Maven е написан на Java и се използва за изграждане на проекти, написани на Java, C#, Scala, Ruby и други. Помага при управлението на компилации на проекти, зависимости (dependencies) и документация.

Maven използва декларативен подход за управление на проекти, което означава, че разработчиците определят какво искат да направи проектът, а не как да го направи. Това се постига чрез писане на файл за конфигурация, наречен pom.xml - Project Object Model (проектен обектен модел), който дефинира зависимостите на проекта, процеса на изграждане и друга информация за проекта.

Едно от основните предимства на използването на Maven е, че опростява процеса на изграждане. Maven елиминира нуждата от ръчно управляване на зависимости, писане на скриптове и разпространение и управление на пакети, защото автоматично прави всички тези неща. Това прави разработката на проект по-лесна и по-бърза.

Друго предимство на Maven е, че има огромно хранилище от библиотеки на допълнителен софтуер (third-party software), които да бъдат използвани директно в проекта. Хранилището на Maven съдържа хиляди библиотеки, които могат лесно да бъдат интегрирани във вашия проект, като просто ги добавите към съответния му pom.xml файл.

Maven също включва функции като тестване и генериране на документация, което го прави цялостен инструмент за изграждане и управление на проекти и отново значително скъсява времето, нужно за разработване на даден проект.

Има алтернативни инструменти за изграждане на Maven, като Gradle и Ant. Gradle е по-модерен инструмент за изграждане, който все още набира популярност, докато Ant е по-стар инструмент за изграждане, който все още се използва.

Mapstruct

MapStruct е базиран на Java генератор на код, който опростява прилагането на картографиране (преминаване от един тип обект към друг) между типовете Java Bean. Това е базирана на анотации рамка, която елиминира необходимостта разработчиците да пишат ръчно шаблонен код за преобразуване, което им позволява да се съсредоточат върху бизнес логиката. В проект за Spring Boot MapStruct може да се използва за преобразуване на обекти за прехвърляне на данни (DTO) в обекти на базата данни и обратно. MapStruct генерира имплементацията на методи от Java класове за картографиране (които се декларират като такива с анотация) по време на компилиране, което прави кода по-бърз и по-малко податлив на грешки.

MapStruct позволява посочване на дадена стратегия за преминаване между типовете и използване на собствени конвенции за именуване на

генерираните методи и променливи. Някои от базовите видове преобразувания, които MapStruct поддържа са:

- Съпоставяне между полета с едно и също име;
- Съпоставяне между полета с различни имена;
- Съпоставяне между полета с различни типове;
- Картиране между полета с вложени обекти ;
- Съпоставяне между полета с вложени списъци.

MapStruct също поддържа персонализирани картографирания, което позволява дефиниране на собствена логика на картографиране за конкретни сценарии. За да се използват генерираните функции, трябва да бъде инжектирана инстанция на Mapper интерфейсния клас, който е създаден.

2.2.3 Фронтенд

Thymeleaf

Thymeleaf е модерен шаблонен генератор (template engine) за Java приложения. Той позволява динамични данни в html файлове - шаблони на уеб страници в Java приложения. Най-често се използва със Spring рамката, заради много добрата му интеграция с нея [13]. Той е много специфично подходящ за моето решение, затова избрах да използвам него като технология. Друга негова много полезна черта е, че позволява бърза разработка. Библиотеката е изключително разширяема и нейната естествена способност за шаблони гарантира, че можем да създаваме прототипи на шаблони без бекенд. Именно това прави разработката много бърза в сравнение с други популярни машини за шаблони като JSP. [14]

Thymeleaf е връзката между фронтенд и бекенд, позволяваща изобразяването на динамични данни и промени по шаблоните в зависимост

от стойности от бекенда, но сам по себе си не е средство за разработка на фронтенд. Затова самите шаблолни за страниците в приложението са HTML файлове, към които има и съответни CSS стилове.

HTML

Съкращението HTML идва от Hypertext Markup language. Този език е официален уеб страдарт, но не е програмен език, защото не позволява създаване и използване на динамична функционалност. World Wide Web Consortium (W3C) поддържа и развива HTML спецификациите, заедно с предоставянето на редовни актуализации. [15] Алтернативи на HTML са XML и SVG, но се използват много рядко при разработката на уеб приложения. Всяка страница е изградена от по-малки части - тагове и техните атрибути, които позволяват оформянето на страница с нейните секции, параграфи, линкове и други.

HTML елементът не е проектиран да има тагове, които да помогнат за форматирането на страницата, целта му е да се маркира структурата на страницата. Въпреки че HTML позволява писането на стилове в страницата, то това не е добра практика, защото измества фокуса от структурата на страницата към външния й вид. Затова използваме CSS.

CSS

CSS е разработен от W3C (World Wide Web Consortium) през 1996 г. Тагове като <font> са въведени в HTML версия 3.2 и причиняват доста проблеми на уеб разработчиците. Поради факта, че уеб страниците се състоят от множество различни шрифтове, цветни фонове и многобройни стилове, каквито и да е промени в кода се превръщат в дълъг, болезнен и скъп процес. Така CSS е създаден от W3C, за да реши този проблем. CSS не е технически задължително необходим, но се е наложил в разработването на уеб приложенията. [16]

Bootstrap

Bootstrap е open-source популярна библиотека на CSS, която предоставя предварително проектирани компоненти за многократна употреба, като навигационни менюта, формуляри, бутони и типография, които могат лесно да бъдат персонализирани и интегрирани в уеб проекти. Bootstrap също така включва CSS и JavaScript компоненти, което позволява на разработчиците да създават интерактивни потребителски интерфейси лесно.

JQuery

JQuery е популярна JavaScript библиотека, която е предназначена за опростяване процеса на създаване на интерактивни уеб страници. Тя предоставя лесен за използване API за манипулиране на обектния модел на документа (DOM), обработка на събития, правене на HTTP заявки и много други. Друга положителна характеристика е, че позволява писането на по-малко код, но постигането на същите резултати като с JavaScript. Също така осигурява съвместимост между различни браузъри, което означава, че кодът на уеб сайта ще работи последователно в различни уеб браузъри. В допълнение, jQuery предоставя редица добавки (plugins), които могат да се използват за разширяване на неговата функционалност. Тези плъгини могат да се използват за добавяне на анимации, създаване на плъзгачи, добавяне на валидиране на формуляр и др.

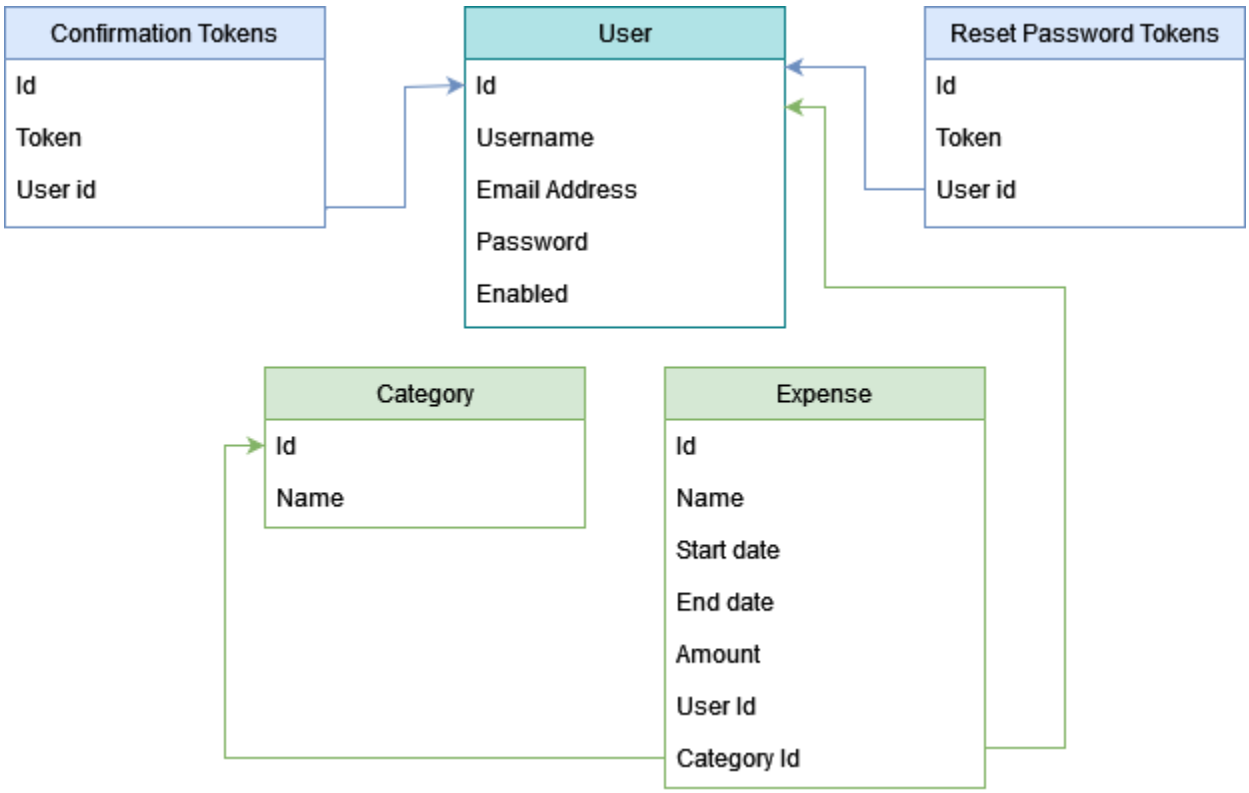
Chart.js

Chart.js е open-source библиотека на JavaScript за визуализиране на данни. Тя е втората най-популярна библиотека за диаграми на JavaScript в GitHub по брой звезди след D3.js, считана за значително по-лесна за използване, но по-малко адаптивна. Chart.js се изобразява с помощта на

HTML5 <canvas> таг и е широко разпространена като една от най-добрите библиотеки за визуализация на данни. С нея могат да се правят: стълбовидна, линеарна, сълбовиднолинератна, кръгова диаграма и други.

2.3 Проектиране на базата данни

Базата данни съдържа всичката информация, с която си служи уебсайтът. Тя съдържа информация за Потребител, Ключ за регистрация, Ключ за смяна на паролата, Разход и Категория, като това са отделните таблици, като те имат външни ключове една към друга според функционалните изисквания на приложението. На фигура 2.3 е представена графично структурата на базата данни.



Фиг. 2.3

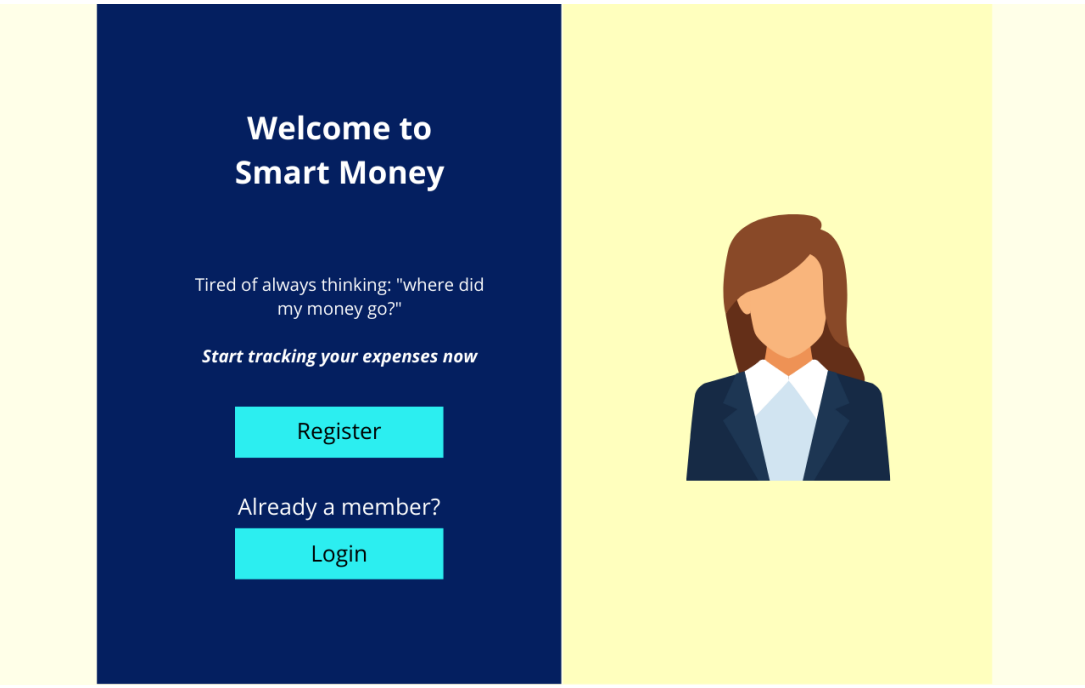
2.4 Подготовка на примерен потребителски интерфейс

2.4.1 Canva

Canva е уеб-базиран инструмент за графичен дизайн. Той позволява на потребителите лесно да създават професионално изглеждащи дизайни за широк спектър от цели, включително публикации в социални медии, плакати, листовки, покани, визитни картички и други. Платформата е популярна сред физически лица и фирми от всякакъв размер, от фрийлансъри и малки стартиращи компании до големи корпорации и организации с нестопанска цел.

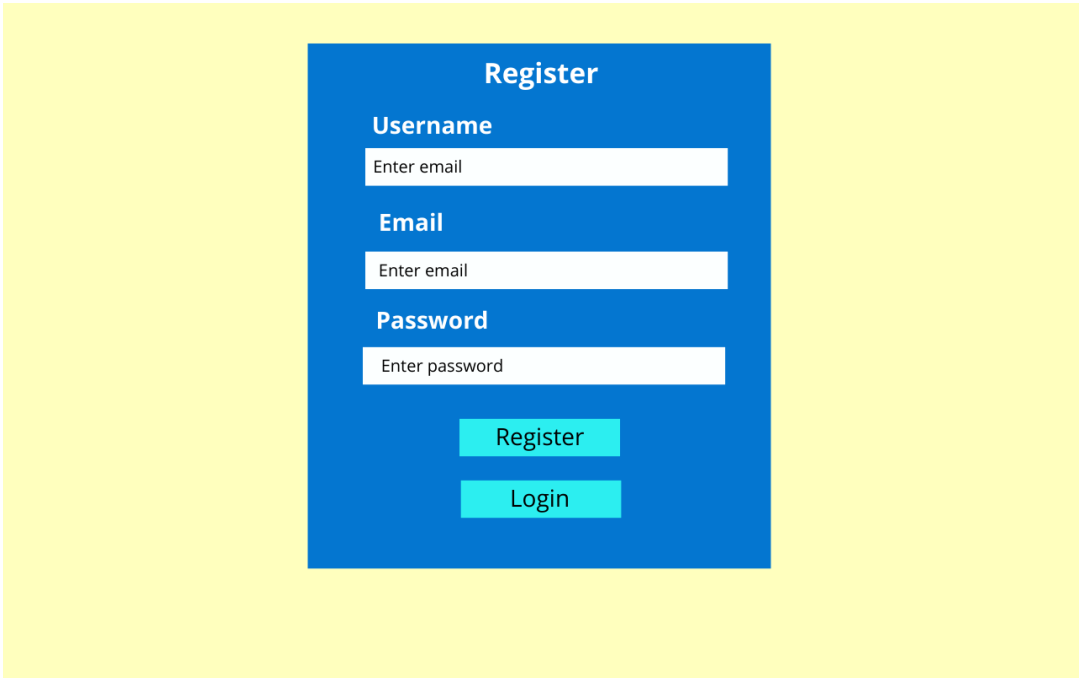
Canva предлага удобен за потребителя интерфейс и набор от шаблони, дизайнерски елементи и инструменти, които помагат на потребителите да създават дизайни бързо и лесно. Има избор от над 1 милион снимки и графики. Една от ключовите характеристики на Canva е неговата функционалност за плъзгане и пускане (drag-and-drop), която позволява на потребителите бързо и лесно да местят елементи около платното и да ги преоразмеряват, за да паснат на визията за техния дизайн. Платформата също така включва набор от филтри, шрифтове и цветови палитри, за да помогне на потребителите да създават визуално привлекателни дизайни, пасващи на техните изисквания.

В допълнение към дизайнерските си функции, Canva също така включва редица инструменти за съвместна работа, които позволяват на потребителите да споделят своите проекти с други, да получават обратна връзка и да работят заедно по проекти в реално време. Платформата също така предлага набор от интеграции с инструменти като Dropbox и Google Drive, за да улесни достъпа и споделянето на файлове. На фигури 2.4 - 2.11 са показани примерните изгледи на страници от потребителския интерфейс на приложението за следене на лични финанси.



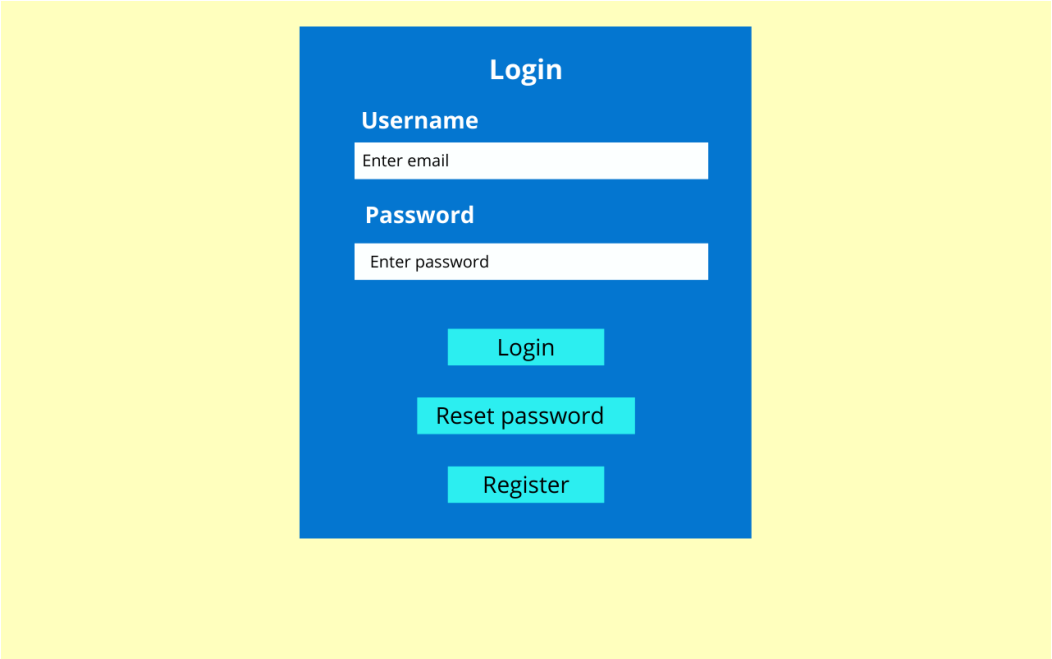
Фиг. 2.4

Начална страница (landing/welcome page)



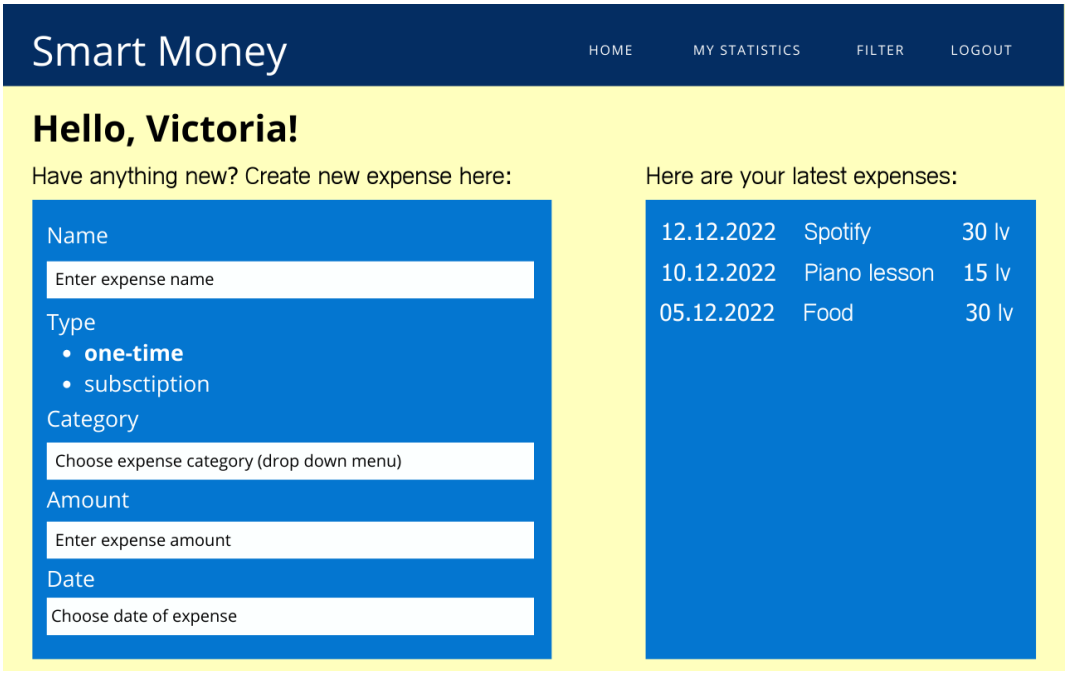
Фиг. 2.5

Страница за регистриране на нов потребител (register page)



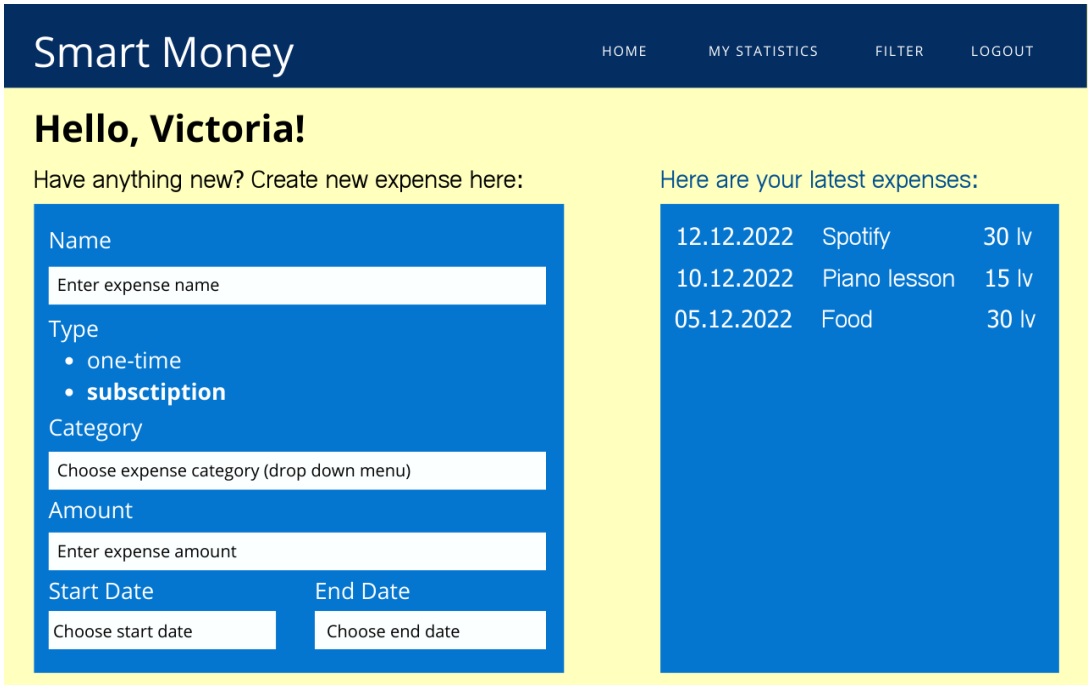
Фиг. 2.6

Страница за влизане в профила на потребител



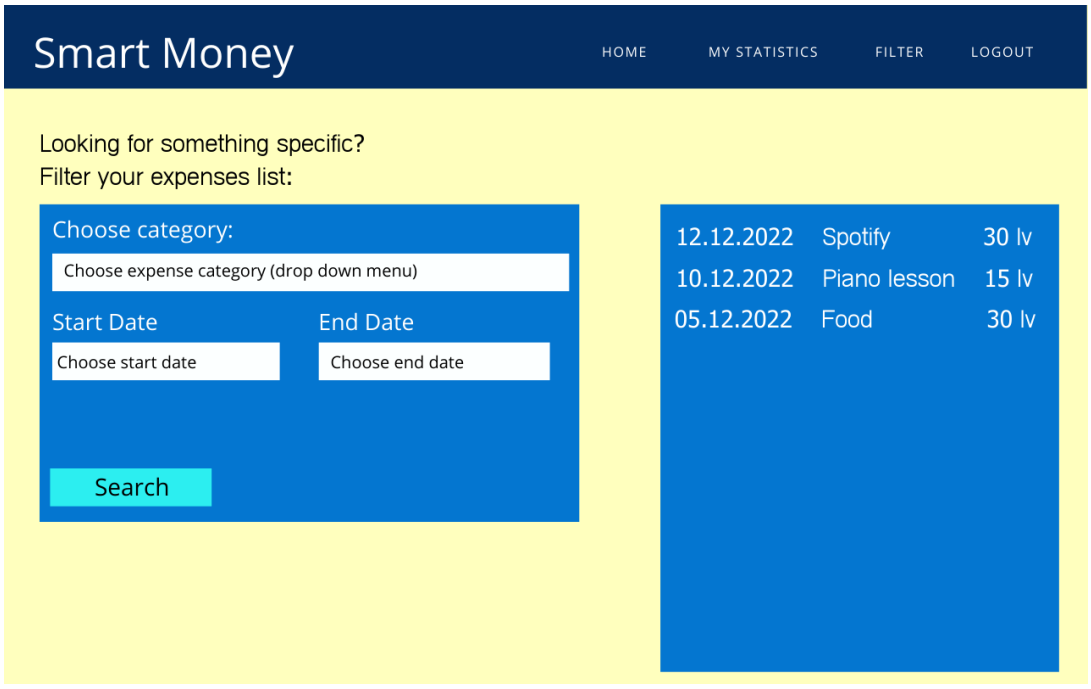
Фиг. 2.7

Страница на профила с добавяне на еднократно плащане



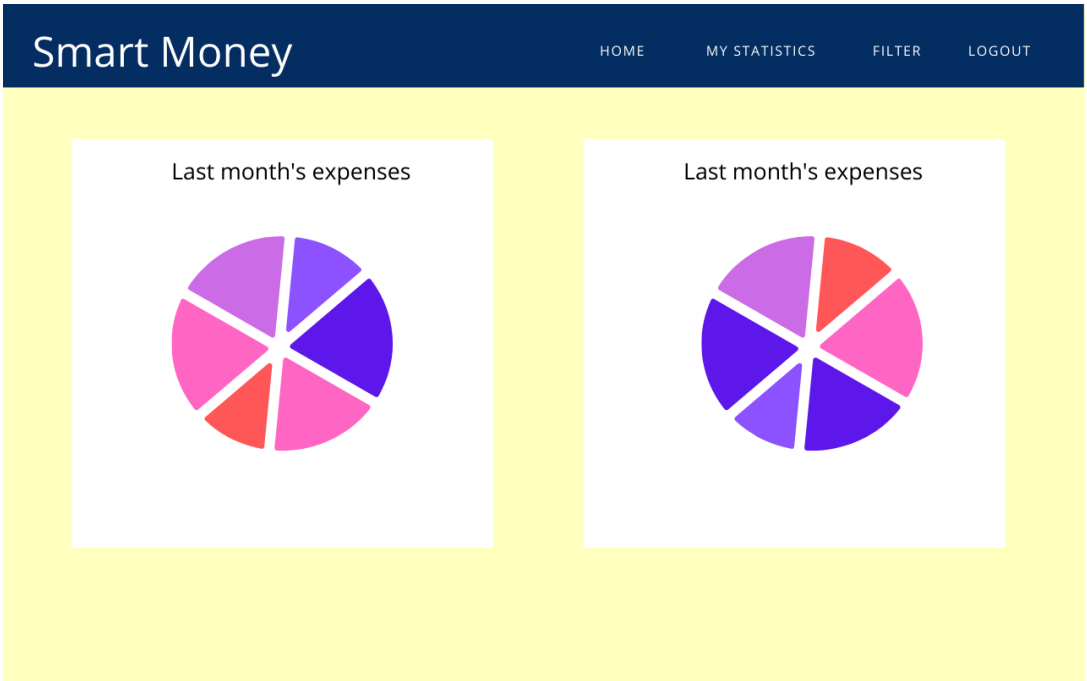
Фиг. 2.8

Страница на профила с добавяне на абонамент



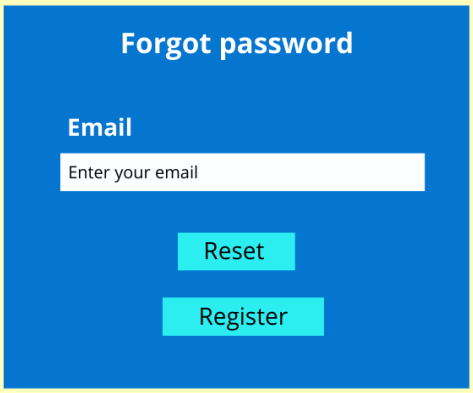
Фиг. 2.9

Страница за персонализирана статистика



Фиг. 2.10

Страница с месечна и годишна статистика



Фиг 2.11

Страница за въвеждане на имейл при забравена парола

2.4.2 UnDraw

UnDraw е open-source колекция, използвана за добавяне на висококачествени персонализирани илюстрации към проекти. С набор от налични стилове и теми unDraw улеснява намирането на перфектното изображение. Всички изображения са безплатни за използване и могат да бъдат персонализирани, за да отговарят на цветовата схема и марката на дадения проект. Библиотеката unDraw непрекъснато се разширява, като редовно се добавят нови илюстрации.

Всички изображения, активи и вектори, публикувани в unDraw, могат да се използват безплатно за некомерсиални и търговски цели. По-точно, unDraw предоставя световен лиценз за авторски права за изтегляне, копиране, модифициране, разпространение, изпълнение и използване на активите, предоставени от unDraw безплатно, включително за търговски цели, без разрешение от или посочване на създателя или unDraw.

На фиг. 2.12 е показана една от илюстрациите, взети от unDraw, като тя е преработена, за да пасва на стиловете в приложението. Използва се на началния екран на приложението - welcome page.

Фиг. 2.12

2.5 Избор на развойни среди

2.5.1 IntelliJ IDEA Community Edition

IntelliJ IDEA е интегрирана среда за разработка (IDE), написана на Java за разработване на компютърен софтуер, написан на Java, Kotlin, Groovy и други езици, базирани на JVM. Средата е продукт на JetBrains (с предишно наименование IntelliJ). [18] Има два варианта - Community и Ultimate edition, като Community версията е безплатната от двете.

IntelliJ IDEA е най-често използваната среда за разработка на Java приложения. Някои от ключовите характеристики на IntelliJ IDEA са:

- Лесно стартиране, възможност за тестване, дебъгване и декомпилиране на проектите, клавишни комбинации за бърза разработка, стандартни и персонални цветови теми за интерфейса;
- Асистенция при писане - чрез анализиране на контекста IntelliJ позволява завършване на кода (имена на променливи, класове и т.н.), навигация в кода, която представлява директно прескачане към клас или декларация на метод/променлива в кода, бърз рефакторинг на кода, отстраняване на грешки, линтинг (семантични грешки като празен ред накрая на файла) и опции за коригиране на несъответствия;
- Въпреки че IntelliJ IDEA е създаден за Java приложения, той поддържа разнообразие от други езици като Groovy, Kotlin, Scala, JavaScript, TypeScript, and SQL, като за тях има и асистенция при писане;
- Интеграция с инструменти за изграждане на приложението - IntelliJ има пълна и много добра интеграция с инструменти като Gradle, Maven, Gant и Ant. Това предоставя вградени функции за пакетиране;
- IntelliJ има вграден терминал, който поддържа всички команди на съответната операционна система;

- Интеграция с инструменти за контрол на версиите (version control systems) - IntelliJ IDEA поддържа най-популярните системи за контрол на версиите, като Git, Subversion, Mercurial и Perforce. Позволява клонирането на проект от VCS (version control system) направо от началния екран, изследването на разликите между два къмита (две версии), управляването на branch-ове (разклонения в проекта), създаване и публикуване на къмити (промени), разрешаване на конфликти, преглеждане на хронологията и др. Към тази интеграция е включен инструмент за история на локалното хранилище (Local History). Това позволява лесно отменяне на локалните проблеми и възстановяване на файлове при изтриване по грешка.

Това е основната среда, в която уеб сайтът бе разработен, тестван и дебъгван с помощта на Maven. С цел улеснение при разработката на фронтенда на приложението бе използван и Visual Studio Code.

2.5.2 Visual Studio Code

Visual Studio Code е друга известна среда за разработка. Тя е много лека и може да се използва на Windows, Linux и MacOS. Има вградена поддръжка на JavaScript, TypeScript и Node.js. [19]

Основни положителни характеристики на Visual Studio Code са:

- IntelliSense - проверка на синтаксиса и интелигентни довършвания въз основа на типове променливи, дефиниции на функции и импортирани (внесени, допълнително добавени) модули;
- Лесно дебъгване на кода от средата за разработване с интерактивна конзола и точки на прекъсване (breakpoints);
- Интеграция с Microsoft Azure за лесен deployment (инсталиране и настройване в облачна среда) на проектите директно от средата;

- Интеграция с инструменти за контрол на версиите - преглеждане на разликите във версиите, записване на промени, разрешаване на конфликти, поетапно запазване на версии (stage commit) и други функционалности директно от средата за разработване;
- Екосистема от разширения - множество разширения (extentions), позволяващи добавяне на езици (например Python, Java, C/C++, PHP и други), цветови теми, програми за отстраняване на грешки и други допълнителни услуги. Разширенията са в отделни нишки, за да не забавят работата на средата за разработка.

При разработката на фронтенда на приложението използвах Visual Studio Code, заедно с разширението Live Server. Това разширение позволява бързото разработване на фронтенда, тъй като всеки HTML документ (страница) може да се отвори в реално време в браузър по избор на разработчика. Всяка промяна по структурата на страницата и стиловете към нея се засича и се визуализира веднага в браузъра. Live server позволява персонализиране на номера на port-а и браузъра по подразбиране, поддържа svg, https и proxy. [20]

Трета глава
Програмна реализация на уеб сайта за управление на личните разходи

3.1 Начин на работа и структура и имплементация на приложението

Spring Core е основата на Spring Framework, предоставяйки основните градивни елементи за разработване на корпоративни приложения с рамката. В основата си Spring предоставя лек контейнер, който управлява жизнения цикъл на Java обекти, известни също като beans. Този контейнер се нарича „Spring IoC container“, където „IoC“ означава „Инверсия на контрола“ (Inversion of Control).

В традиционното програмиране програмата контролира потока на изпълнение, създавайки и управлявайки обекти според нуждите. Spring IOC (Inversion of Control) е модел на проектиране, при който контролът върху създаването и управлението на обекти е обърнат или преместен от кода на приложението към външна рамка. В контекста на рамката Spring това означава, че рамката поема контрола върху създаването на обекти, управлението на техния жизнен цикъл и техните зависимости и инжектирането им в кода, където са необходими. Така рамката управлява обектите и техните връзки, докато кодът на приложението просто ги използва. Това позволява по-голяма гъвкавост, възможност за тестване и поддръжка на приложенията.

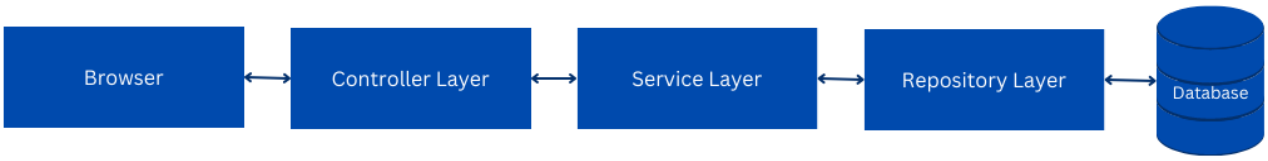
Основната концепция зад Spring IOC е използването на инжектиране на зависимости (DI - dependency injections), което е техника за постигане на свързване между класове чрез премахване на отговорността за създаване на обекти и окабеляване (wiring) от кода. С инжектирането на зависимости обектите се създават и свързват заедно от външната библиотека, вместо да

се създават от самите класове. В Spring IOC обектите се дефинират като beans, които са екземпляри на клас, които се управляват от Spring контейнера. Контейнерът Spring е отговорен за създаването и инициализирането на тези beans, свързването им заедно и предоставянето им на други обекти, които ги изискват. Ползата от използването на Spring IOC е, че прави кода по-модулен, поддържаем и тестван, като премахва зависимостта от конкретни реализации и позволява лесно заместване на компоненти. Той също така дава възможност за разделяне на проблемите, тъй като логиката на приложението е отделена от подробностите за изпълнението, които се управляват от рамката Spring.

Основният контейнер на Spring е имплементиран чрез интерфейса ApplicationContext, който предоставя възможност за зареждане на дефиниции на компоненти и връзки между компоненти. Дефинициите на Bean обикновено се съхраняват в XML файлове, въпреки че Spring също поддържа анотации и базирана на Java конфигурация.

Когато се стартира приложение Spring Boot, контейнерът Spring IoC сканира класовата пътека, за да намери bean-ове и зависимости, създава екземпляри на bean-овете и ги свързва заедно. Контейнерът управлява жизнения цикъл на bean-овете, като гарантира, че те се създават и унищожават, както е необходимо, и че зависимостите се инжектират, както се изисква. Контекстът на приложението, който е дефиниран в конфигурационните файлове, се използва за определяне кои компоненти трябва да бъдат създадени и как трябва да бъдат свързани заедно. Резултатът е напълно конфигурирано приложение, което е готово за стартиране.

На фигура 3.1 е показана най-общо структурата на проекта като слоеве.



Фиг. 3.1

Първият от тях е потребителският интерфейс, който се вижда в браузъра. Той представлява статичните ресурси на приложението - HTML страници, които са шаблони за рендериране (templates) на уеб страници, CSS стилове и JavaScript документи, с които се манипулират страниците при нужда.

Следва слоят на контролерите. Това са Java класове, които отговарят на Spring компоненти. С тях се имплементира MVC метода на работа на приложението. Те се анотират със специалната анотация @Controller (или @RestController в зависимост от същността на проекта) и това сигнализира на Spring, че са компоненти. Те получават заявка от сървлет диспечера и изпълняват съответния метод за нея на. Също отговарят за подаването на модели с информация от по-долните слоеве на приложението към изгледите (шаблоните). Контролерите включват в себе си и валидация на входните данни.

Следващият слой е слоят на услугите. В него се съдържа бизнес логиката на приложението. Всякаква обработка на данни трябва да се случва на това ниво на абстракция. Класовете с логика се аннотират като @Service класове, за да Spring ги възприеме като компоненти и Spring IoC да се погрижи за тях. На този слой се използват и POJO/DTO обекти (класове за обекти, които са само за пренос на данни и не съдържат в себе си никаква логика), тъй като не е препоръчително да се работи с Entity обекти. Информацията, нужна на контролерите, идва от това ниво и трябва

да е готова за директно предаване на template engine-а, който е Thymeleaf. Другият вид данни, с които работи този слой, са тези от базата данни. За да може да ги използва, услугите комуникират със слоя на хранилищата.

Слоят на хранилищата представлява връзката към базата данни. На това ниво се дефинират операциите, с които може да си служи слоят на услугите. Класовете се аннотират с @Repository, за да Spring се погрижи за тяхното управление.

3.2 База от данни

Приложението използва релационна база данни. Това е най-често използваният начин за съхранение на данни в настоящия момент. Базата данни се състои от пет таблици с цел нормализиране на данните, като между тях има външния ключове, които реферират първичните ключове на съответната външна таблица. Връзките между таблиците са еднопосочни (unidirectional), а базата данни се записва локално във файл.

Основните анотации са:

- @Entity - маркира клас като модел на база данни, за да може Spring да го сканира като компонент и да се погрижи за съответния bean;
- @Table - използва се за задаване името на таблицата;
- @Id - задължителна анотация за всеки Entity клас, полето се маркира като първичен ключ на съответната таблица, което го прави задължително и уникално;
- @GeneratedValue(strategy = GenerationType.AUTO) - стойността на първичния ключ ще бъде автоматично генерирана от Hibernate, като той поддържа генериране на числови първични ключове или от тип UUID;
- @Column - с нея се задават различни характеристики и ограничения на колоната, като те определят второто ниво на валидация на данните (първото ниво се намира на слоя на услугите).

Първата таблица отговаря за записването на потребителските данни - потребителско име и парола, както и имейл адрес, които се използват при регистрирането на нов потребител в сайта и автентикацията на потребителя при влизане в профила си, а имейлът е за да може да се реализира друго от изброените в 2.1 функционални изисквания - изпращане на имейл уведомление на потребителя, когато остават 3 дни до изтичане на някой от неговите абонаменти. На фиг. 3.2 е показан Java класът, който е модел на тази таблица (Entity). Първичният ключ на тази таблица е от тип Long. Втората колона на таблицата отговаря за потребителското име и е задължителна и уникална - не може двама потребители да имат едно и също потребителско име, както и не може да бъде създаден потребител без потребителско име. Третата колона на таблицата съдържа потребителския имейл, уникален и задължителен атрибут - не може да бъде направен запис в таблицата, където имейлът е празен или дублира някой от вече записаните. Следващата колона е за паролата, като тя е само задължителна. Последната колона се използва за следене дали профилът е активиран (с имейл за потвърждение).

```
package org.elsys.diplom.entity;


@Entity
@Table(name="users")
public class User {
  @Id
  @GeneratedValue(strategy= GenerationType.AUTO)
  private Long id;

  @Column(name = "username", nullable = false, unique = true)
  private String username;
```

```
  @Column(name = "email", nullable = false, unique = true)
  private String email;

  @Column(name = "password", nullable = false)
  private String password;

  @Column(name = "enabled", nullable = false)
  private boolean isEnabled;

  public User() {}
  // Getters and setters
}
```

Фиг. 3.2

Втората таблица отговаря за категориите. Тя се използва за нормализирането на данните, тъй като без нея към всеки разход трябва да има текстови атрибут за име на категорията и ще настъпи повтаряне. Класът за нея е показан на фиг. 3.3.

Таблицата се състои от две колони - първичен ключ от тип Long, чрез който таблицата за разходи реферира категорията на съответния разход, и текстови атрибут с името на категорията, която трябва да бъде записана. Първичният ключ е уникален и задължителен, но същите условия се отнасят и до името на категорията за избягване на дублиране.

```
package org.elsys.diplom.entity;

@Entity
@Table(name="categories")
public class Category {
  @Id
```

```
    @GeneratedValue(strategy= GenerationType.AUTO)
    private Long id;

    @Column(name = "name", nullable = false, unique = true)
    private String name;

    public Category() {}

    public Category(Long id, String name) {
        this.id = id;
        this.name = name;
    }
    // Getters and setters
}
```

фиг. 3.3

Третата таблица е тази на разходите и е показана на фигура 3.4. Тя съдържа информацията, която е най-съществена за работата на уеб сайта, но не може да съществува самостоятелно. Общият брой на колоните е седем, като първата е първичният ключ. Следващият атрибут е името на самия разход, като той е текстов и задължителен, но не е уникален. Следващата двойка колони са начална дата и крайна дата. Това са полета с date формат, като в тях се запазват началната и крайната дата на абонаментен разход, а когато е еднократно плащане двете колони приемат една и съща стойност. Двете колони не могат да бъдат празни, тъй като се използват за изграждането на статистиките, които са част от задължителните функционални изисквания, описани в т. 2.1. Следващият атрибут е от тип Float и се отнася до стойността на разхода. Той е задължителен, но може да се повтаря. Последните две колони на тази таблица са външните й ключове. Първият от тях реферира първичен ключ

на някой запис от таблицата с потребителите, а вторият от тях реферира първичен ключ на някоя от категориите. Тези атрибути са задължителни, защото иначе разходът ще остане без потребител и/или категория, което би го направило невалиден, неизползваем.

```
package org.elsys.diplom.entity;

@Entity
@Table(name="expenses")
public class Expense {
  @Id
  @GeneratedValue
  private Long id;

  @Column(name = "name", nullable = false)
  private String name;

  @Column(name = "start_date", nullable = false)
  @Temporal(TemporalType.DATE)
  private LocalDate startDate;

  @Column(name = "end_date", nullable = false)
  @Temporal(TemporalType.DATE)
  private LocalDate endDate;

  @Column(name = "amount", nullable = false)
  private Double amount;

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn(name = "user_id", referencedColumnName = "id")
  private User user;
```

```
  @OneToOne(fetch = FetchType.LAZY)
  @JoinColumn(name = "category_id", referencedColumnName = "id")
  private Category category;

  public Expense() {}
  // Getters and setters
}
```

фиг. 3.4

Анотацията @Temporal се използва за указване на типа на поле за времева стойност. Анотацията @Temporal може да се използва с java.util.Date, java.util.Calendar и java.time.LocalDate, java.time.LocalDateTime, java.time.LocalTime, java.time.OffsetDateTime и java.time.OffsetTime. @Temporal приема параметър от enum (предварително дефинирани опции) типа TemporalType, който указва типа на времевата стойност и има три стойности: DATE, TIME и TIMESTAMP, които съответстват на SQL стандартни времеви стойности. @Temporal(TemporalType.DATE), означава, че анотираното поле представлява само датата от цялостната стойност дата-час. Часовете, минутите и секундите от стойността не се съхраняват или вземат предвид. TemporalType.DATE се използва за преобразува към типа SQL DATE.

За съставяне на връзките между таблиците в базата данни се използват анотации. @OneToOne анотацията се използва за дефиниране на връзка едно към едно между два обекта в релационна база данни, където единият обект е "притежаващата" страна на връзката, а другият е "съпоставената" (mapped). Задава се начинът на зареждане на връзките (fetching).Fetch = FetchType.LAZY - това показва, че съпоставяният обект трябва да се зарежда лениво, което означава, че ще бъде взет от базата данни само когато действително е необходим. Това може да подобри

производителността чрез намаляване на броя на заявките към базата данни, които се изпълняват. @JoinColumn(name = "category_id", referencedColumnName = "id") анотацията се използва за указване на колоната, която ще се използва за свързване на двете таблици. В този случай това означава, че колоната „category_id“ в таблицата на притежаващия обект трябва да се използва за обединяване с колоната „id“ в таблицата на съпоставяния обект (фиг. 3.4).

По същият начин е дефинирана връзката много към един. Много разходи могат да бъдат на един и същи човек. Зареждането на потребителя отново е от тип FetchType.LAZY, а колоната, която се използва за свързване е user_id, като реферира “id” на съпоставящата таблица users.

Четвъртата таблица съдържа информацията, нужна за потвърждаването на профил при регистрация. То се извършва чрез изпращане на имейл уведомление на потребителя, което съдържа линк със случайно генериран текстов ключ. В тази таблица се записват: първичен ключ на записа в таблицата, текстовият ключ и външен ключ, който реферира потребителя, за когото се отнася този запис. Текстовият ключ е от тип String, останалите полета в таблицата са от тип Long. Всеки линк е персонален, затова таблицата има отношение едно към едно (@OneToOne) към таблицата с потребителите. Извличането на потребителя тук е с fetch = FetchType.EAGER, което означава, че свързаният потребителски обект ще бъде извлечен нетърпеливо, т.е. свързаният обект ще бъде зареден незабавно заедно с притежаващия обект (Confirmation Token обекта).

Петата таблица се използва в случай, че потребителят е забравил своята парола и иска да я промени. Това се случва чрез линк, изпратен на потребителския имейл. В тази таблица се записват: първичен ключ на записа в таблицата, текстовият ключ (token), генериран на случаен принцип и външен ключ, който реферира потребителя, за когото се отнася този запис. Отново всеки token е персонален, затова се изисква връзка едно към

едно с таблицата за потребители, а зареждането на потребителя става едновременно с извличането на записа от тип Reset Password Token.

3.3 Бекенд

3.3.1 Хранилища

Анотацията @Repository се използва, за да посочи, че определен клас предоставя операции за достъп до данни. Анотацията @Repository е специализация на анотацията @Component и се използва за предоставяне на предимствата на инжектирането на зависимости на Spring и инверсията на контрола (Spring IoC).

JPA (Java Persistence API) предоставя три типа хранилища:

- CrudRepository - предоставя основни CRUD (създаване, четене, актуализиране, изтриване) операции на обекти; разширява интерфейса на Repository;
- PagingAndSortingRepository - този тип разширява CrudRepository и предоставя допълнителни методи за страниране и сортиране на резултатите от търсенето;
- JpaRepository - предоставя допълнителни, специфични за JPA, методи в допълнение към горните два вида - CrudRepository и PagingAndSortingRepository. Това хранилище предоставя методи като изчистване на контекста, групово изтриване на записи и заключване на записи.

На фигура 3.5 е показана имплементацията на хранилището, което отговаря за операциите към таблицата с потребителите. При деклариране на интерфейса на дадено хранилище, се подават два параметъра - първият е видът на entity обекта, който хранилището управлява, а вторият е типът на първичния ключ на обекта. Тези параметри се използват, за да правилно бъдат генерирани методите на хранилището. Дефинирани са два метода -

findByUsername и findByEmail. Тяхната имплементация се реализира от Spring Data JPA, като за да това се случи е използвана camelCase конвенцията и условието името на метода да започва с “findBy”. И двете функции връщат обект от тип User. Ако не може да бъде намерен резултат, този обект ще бъде null. Друг начин за имплементиране би бил използването на ключовата дума Optional. Ако методите връщаха тип Optional<User>, то в случай, че няма намерен резултат, няма да бъде върнат никакъв обект. Това е важно за проверките на слоя на услугите.

```
package org.elsys.diplom.repository;


@Repository
public interface UserRepository extends JpaRepository<User, Long> {
   User findByUsername(String username);
   User findByEmail(String email);
}
```

Фиг. 3.5

На фигурите 3.6 и 3.7 са представени хранилищата, които управляват съответно данните за разходите и токените за потвърждение на профила. Приложението използва още две хранилища - за категориите и за токените за промяна на парола. Те работят на същия принцип.

```
package org.elsys.diplom.repository;

@Repository
public interface ExpenseRepository extends JpaRepository<Expense, Long> {
  List<Expense> findByUserId(Long id);
  List<Expense> findByEndDate(LocalDate endDate);
   List<Expense> findByUserIdAndStartDateBetween(Long id, LocalDate startDate, LocalDate endDate);
      List<Expense> findByUserIdAndCategoryIdAndStartDateBetween(Long userId, Long categoryId, LocalDate startPeriod, LocalDate endPeriod);
}
```

Фиг. 3.6

```
package org.elsys.diplom.repository;

@Repository
public interface ConfirmationTokenRepository extends CrudRepository<ConfirmationToken, String> {
  ConfirmationToken findByToken(String token);
}
```

Фиг. 3.7

3.3.2 Логически слой

Логическата част от приложението съдържа в себе си обекти за пренос на данни (DTOs), класове за валидатори на входящи данни, класове за преобразуване между типовете обекти (mappers) и самите класове за услуги.

Обекти за пренос на данни (DTOs)

Обектите за пренос на данни се избират и създават според нуждите на приложението. Имплементирани са четири обекта. UserDto, първият от тях, се използва за скриване на чувствителната информация като имейл и парола на потребителя от контролерите. Вторият е UserRegisterDto. Той се използва при процеса на регистриране на нов потребител, съдържа всички нужни атрибути и съответната валидация на първо ниво за тях. ExpenseDto е третият такъв клас и отговаря за запазването на нов разход с нужните му валидации. Последният Dto клас е FilterExpensesDto. Той събира в себе си критериите, които потребителят иска да приложи за филтриране на своите разходи.

Валидация на данни

Използваните аннотация за валидация са:

- NotNull - ограничение, което указва, че анотираният елемент не трябва да е нулев (null). Това ограничение може да се приложи към всеки тип поле или параметър и е основна проверка за валидиране, за да се гарантира, че задължително поле има някаква стойност;
- @NotEmpty - ограничение, което указва, че анотираният String (може да се слага само на String) не трябва да е нулев или празен. Това означава, че анотираният атрибут трябва да има дължина, по-голяма от нула, но не е задължително да има знаци, различни от шпация (интервал, space);
- @NotBlank - ограничение, което указва, че анотираният String (може да се слага само на String) не трябва да е нулев, празен или да се състои само от “празни” знаци (шпации). Това означава, че анотираният низ трябва да има поне един знак, който не е интервал;
- @Length се използва за указване на ограниченията за дължина на поле или параметър на клас. Обикновено се използва за валидиране

на дължината на String-ове, масиви или колекции, като има аргументи min и max за минимална и максимална дължина на параметъра;

- @Pattern се използва за указване на шаблон на регулярен израз (regular expression или по-кратко - regex), на който трябва да съответства даден параметър. Самият израз се подава като аргумент с наименованието regexp;
- @DecimalMin е валидиращо ограничение в Spring, което се използва, за гарантира, че числова стойност е по-голяма или равна на определена минимална стойност, която се подава като параметър с наименованието value.

Всяка от изброените анотации има параметър по подразбиране message. Това е съобщението, което ще съдържа една грешка, ако валидацията не е премината успешно. То се показва на потребителя. В приложението са създавани и персонализирани анотации за валидиране на данни. Те се състоят от два класа, като единият я дефинира, а другият се грижи за нейната имплементация. Класът за имплементация разширява ConstraintValidator и дефинира имплементацията в isValid() метод. Собствените валидации за класове и атрибути са:

- @EndDateAfterStartDate - използва се в ExpenseDto, за да гарантира, че новият разход има правилно въведени дати. Случаят с еднократни плащания, където двете дати съвпадат също се включва в тази анотация. Нейната имплементация е на фиг. 3.8 и фиг 3.9;
- @Password - персонализирани проверки за plain text паролата (паролата преди да бъде криптирана), използва се върху тип String;
- @Period - изпълнява функцията на @EndDateAfterStartDate, но за FilterExpensesDto класа, тъй като всяка валидация е дефинирана за определен клас или тип атрибут;

- @UniqueUsername - проверява за наличието на даденото потребителско име в базата данни, защото то е дефинирано като уникален идентификатор;
- @UniqueEmail - проверява за наличието на дадения имейл в базата данни, тъй като имейлът също е уникален идентификатор.

```
package org.elsys.diplom.service.validation;

@Constraint(validatedBy = EndDateAfterStartDateValidator.class)
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface EndDateAfterStartDate {
  String message() default "End date should be after start date!";
  Class<?>[] groups() default {};
  Class<? extends Payload>[] payload() default {};
}
```

Фиг. 3.8

```
package org.elsys.diplom.service.validation;
public class EndDateAfterStartDateValidator implements
ConstraintValidator<EndDateAfterStartDate, ExpenseDTO> {
  @Override
  public boolean isValid(ExpenseDTO value, ConstraintValidatorContext context) {
     LocalDate startDate = value.getStartDate();
     LocalDate endDate = value.getEndDate();
     if(startDate == null || endDate == null){
        return false;
     }
     return startDate.isBefore(endDate) || startDate.isEqual(endDate);
  }
}
```

Фиг. 3.9

Използваните анотации в контекста на създаването на нова такава са:

- @Constraint - показва, че анотираният клас е анотация за ограничение, която може да се използва за анотиране на клас, поле или параметър на метод. Нейният параметър validatedBy указва вторият клас на създадената анотация, в който е предефиниран и имплементиран isValid() метода;
- @Target - указва типа на елемента, към който може да се приложи анотацията за ограничение. Параметърът ElementType може да бъде TYPE, FIELD, METHOD и други. В приложението са използвани само първите две опции, като TYPE са отнася за клас в случая, но може да се отнася и до интерфейс или enum, а FIELD - за атрибут на клас;
- @Retention - указва колко дълго трябва да се запази анотираният елемент. @Retention(RetentionPolicy.RUNTIME) показва, че анотацията трябва да се запази по време на изпълнение.

```
package org.elsys.diplom.service.validation;

@Constraint(validatedBy = PasswordValidator.class)
@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Password {
  String message() default "Password must contain digit, lowercase letter and uppercase
letter!";
  Class<?>[] groups() default {};
  Class<? extends Payload>[] payload() default {};
}
```

Фиг. 3.10

```
package org.elsys.diplom.service.validation;

public class PasswordValidator implements ConstraintValidator<Password, String> {
  @Override
  public boolean isValid(String value, ConstraintValidatorContext context) {
    boolean hasDigit = false;
    boolean hasLower = false;
    boolean hasUpper = false;
    for (char c : value.toCharArray()) {
      if (Character.isDigit(c)) {
        hasDigit = true;
      } else if (Character.isLowerCase(c)) {
        hasLower = true;
      } else if (Character.isUpperCase(c)) {
        hasUpper = true;
      }
    }
    return hasDigit && hasLower && hasUpper;
  }
}
```

Фиг. 3.11

Мапинг

За преобразуване на обекти се използва MapStruct. За да се използва MapStruct в проект на Spring Boot, трябва да бъде добавен чрез зависимостта MapStruct към компилационния файл на проекта (pom.xml). След това се създава интерфейсен клас на Mapper, който дефинира съпоставянето между двата типа обекти. Интерфейсът на Mapper трябва да бъде анотиран с @Mapper, за да инструктира MapStruct да генерира имплементацията на преобразуване. Анотацията @Mapper се поставя над съответния абстрактен клас. Параметърът componentModel в @Mapper се

използва за посочване на начина за генериране на класа за преобразуване на типове обекти. В този случай componentModel = "spring" указва, че класът трябва да бъде генериран като Spring компонент, което позволява да бъде автоматично свързан към други компоненти на Spring. Параметърът uses указва допълнителните класове, които да се използват, когато генерира имплементацията на кода за преминаване на типовете. На фиг. 3.12 е показан класът за конвертиране на обекти, представляващи разходи. ExpenseMapper използва класовете User и Category, защото това са класовете, към които има референции. Класът ExpenseMapper дефинира два основни метода при конвертиране, toDto и toEntity, които са отговорни за конвертирането между Expense и ExpenseDTO обекти. Анотациите @Mapping вътре в методите се използват, за да се укаже как трябва да се извърши преобразуването, за да се избегнат грешки при генерирането на имплементацията.

```
package org.elsys.diplom.service.mapper;

@Mapper(componentModel = "spring", uses = {User.class, Category.class})
public abstract class ExpenseMapper{
  @Mapping(source = "user.id", target = "userId")
  @Mapping(source = "category.id", target = "categoryId")
  abstract public ExpenseDTO toDto(Expense expense);

  @Mapping(source = "userId", target = "user.id")
  @Mapping(source = "categoryId", target = "category.id")
  abstract public Expense toEntity(ExpenseDTO expenseDTO);
}
```

Фиг. 3.12

Класове за услуги

Класовете за услуги (Service classes) са основната част от имплементацията на функционалностите в приложението.

Те се отбелязват със @Service аннотация, което указва, че даденият клас е сервизен компонент в Spring приложението. Това е специализация на анотацията @Component, която показва, че анотираният клас е компонент, управляван от Spring. По-специално @Service се използва за анотиране на клас, който изпълнява услуга, като обработка на бизнес логика, извършване на достъп до данни или извикване на външни API. След това Spring контейнерът може да управлява жизнения цикъл на услугата и да я инжектира в други компоненти, ако е необходимо.

@Autowired е аннотация в Spring, която се използва за автоматично инжектиране на зависимост в Spring bean. Когато се създаде bean, Spring търси всякакви анотации @Autowired и инжектира подходящата зависимост, ако бъде намерена такава. Това позволява по-отделен и модулен дизайн, тъй като намалява необходимостта от ръчно управление на обекти и насърчава хлабавото свързване (loose coupling) между класовете, представляващо по-голяма самостоятелност на отделните класове. На фиг. 3.13 е показана най-обща дефиниция на класът, който отговаря за работата с разходи, която е определяща за работата на проекта.

```
package org.elsys.diplom.service;

@Service
public class ExpenseService {
  @Autowired
  ExpenseRepository expenseRepository;
   // More Autowired objects
```

```
public void addExpense(ExpenseDTO expenseDto){...}

public List<Expense> getUsersExpenses(Long userId){...}

private HashMap<String, Double> calculateExpenses(List<Expense> expenses){...}

public HashMap<String, Double> getLastMonthExpenses(Long userId){…}

public HashMap<String, Double> getLastYearExpenses(Long userId){…}

public List<Expense> customFilterExpenses(FilterExpenseDTO filter)…}

public List<Expense> toBeReminded(){…}

public Double calculateTotal(LocalDate startDate, LocalDate endDate, Long UserId){...}
// Calculate total methods for a month and an year

}
```

Фиг. 3.13

Функционалностите, имплементирани в този клас са:

- addExpense() - добавяне на нов разход към даден потребител;
- getUsersExpenses() - четене на всички разходи на даден потребител;
- Зареждане от базата данни и обработване на информация за съставяне на статистики, които съдържат разделение на разходите по категории за последния месец и последната година, както и обща сума на разходите за последния месец и последната година; Имплементацията на тези функционалности се покрива от функциите calculateExpenses() и calculateTotal(), като за двата периода има допълнителни функции;
- customFilterExpenses() - по данните на обект от тип FilterExpenseDto се

четат от базата данни съвпадащите с условията записи от таблицата;

- toBeReminded() - връща списък с изтичащите в следващите три дни абонаменти, за да бъдат изпратени имейл уведомления.

На фиг. 3.14 е показана класът UserService, отговорен за потребителите.

```
package org.elsys.diplom.service;

@Service
public class UserService {
  @Value("${spring.mail.username}")
  private String senderEmail;
  @Autowired
  private BCryptPasswordEncoder bCryptPasswordEncoder;
  // More Autowired objects

  public void addNewUser(@Valid UserRegisterDTO userRegisterDTO){
     User user = userMapper.toEntity(userRegisterDTO);
     user.setPassword(bCryptPasswordEncoder.encode(userRegisterDTO.getPassword()));
     userRepository.save(user);

     ConfirmationToken token = new ConfirmationToken(user);
     confirmationTokenService.saveConfirmationToken(token);
    // Prepare mail message
     emailService.sendEmail(mailMessage);
  }

   // Other class methods

  public UserDTO retrieveLoggedInUser(){
     CustomUserDetails loggedInUser = (CustomUserDetails)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
     UserDTO userDTO = new UserDTO();
```

```
        userDTO.setId(loggedInUser.getId());
        userDTO.setUsername(loggedInUser.getUsername());
        return userDTO;
    }
    // Other class methods
}
```

- @Value("${spring.mail.username}") се използва, за да в приложението не се поставят директно данни. С @Value се извършва инжектиране на променливи от application.properties файл или env.properties/environment variables файл (променливи на средата), като в случая това е spring.mail.username от application.properties файла на приложението;
- addNewUser() - в тази функция се добавя нов потребител към базата данни, като преди това паролата му се криптира с помощта на bCryptPasswordEncoder, а на предоставения от потребителя имейл се изпраща линк за потвърждение на профила, за да той стане активен;
- retrieveLoggedInUser() - използва SecurityContextHolder клас, който е предоставен от Spring Security, за да се получи достъп до текущия контекст на сигурността, който включва подробности за текущия потребител и неговия статус на удостоверяване. Методът getContext() връща контекста на сигурността, свързан с текущата нишка. Обектът Authentication представлява състоянието на удостоверяване на потребителя, а методът getPrincipal() връща обект, свързан с удостоверяването. В този случай принципалът е от тип CustomUserDetails, който е персонализирана реализация на интерфейса UserDetails, използван от Spring Security.

Класът за имейл услуги се казва EmailService и неговата структура е представена на фиг. 3.15.

```
package org.elsys.diplom.service;

@Service
public class EmailService {
   // Autowired objects and constructor
   @Async
   public void sendEmail(SimpleMailMessage email) {
      javaMailSender.send(email);
   }

   @Scheduled(cron = "0 0 0 * * *")
   @Transactional
   public void sendReminders(){
      for(Expense expense : expenseService.toBeReminded()){
         // Prepare mail message
         sendEmail(email);
      }
   }
}
```

Фиг. 3.15

- @Async - анотация, която може да се прилага към методи и показва, че методът трябва да се изпълнява в отделна нишка. Той позволява на метод да върне бъдещ обект, който може да се използва за асинхронно извличане на резултата от изпълнението на метода. В случая функцията е асинхронна, за да не забавя зареждането на template от контролера, когато се изпраща имейл;
- @Scheduled - анотация, която се използва за планиране на изпълнението на метод в определено време или на конкретен

интервал. Той предоставя няколко опции за начин на планиране, включително cron израз, какъвто е използван в настоящото приложение;

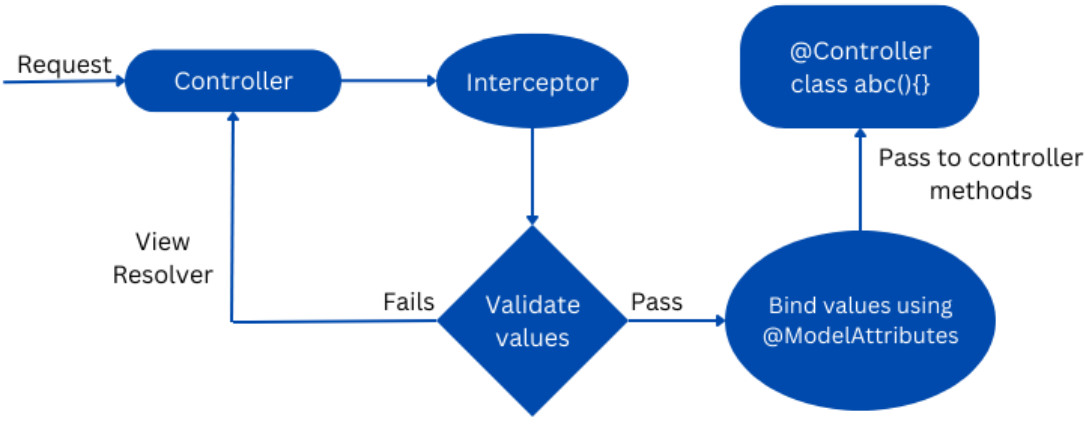
- Изразът cron (cron expression) е String, състоящ се от шест полета, които указват кога трябва да се изпълни планирана задача. Полетата указват минута, час, ден от месеца, месец, ден от седмицата и година (по избор). Всяко поле може да бъде конкретна стойност или диапазон от стойности и може да включва специални знаци за представяне на заместващи знаци или стъпки. Например изразът "0 0 * * * *" ще изпълни метода в началото на всеки час, всеки ден. Изразът "0 0 0 * * *" е използван за изпращане на уведомления всеки ден в 00:00 часа;
- @Transactional е аннотация, която се използва за маркиране на метод като транзакционен. Това гарантира, че методът се изпълнява в рамките на транзакция и че транзакцията се извършва, ако методът завърши успешно, или се връща обратно, ако е хвърлена грешка. Анотацията може да се приложи както към ниво клас, така и към ниво метод.

3.3.3 Слой на контролерите

Нивото на контролера е отговорно за обработката на HTTP заявки от клиенти и връщането на подходящи отговори. Слоят на контролера се реализира чрез Spring MVC, който предоставя гъвкава и мощна рамка за обработка на HTTP заявки. Класовете за контролерите се анотират с анотацията @Controller, която казва на Spring да третира класа като контролер. Тя е спецификация на @Component аннотацията.

Когато клиент изпрати HTTP заявка към сървъра, Spring MVC преобразува заявката към метод на контролер въз основа на URL пътя и HTTP метода. Тогава методът на контролера обикновено използва други компоненти на Spring, като услуги или хранилища, за извършване на бизнес

логика или достъп до данни. Накрая, контролерът връща отговор на клиента, който обикновено е име на изглед, JSON данни или някакъв друг формат. Схемата на имплементиране на валидацията на това ниво е показана на фиг. 3.16.



Фиг. 3.16

Приложението използва два контролера. Единият отговаря за операциите, свързани с профили. Това е Authentication Controller-а, който е показан на фиг. 3.17. Вторият контролер е отговорен за функционалностите, които потребителят получава, след като е удостоверен. Това е Home Controller класът, чиято имплементация е показана на фиг. 3.18.

Ключови аннотации и обекти за тези класове са:

- @Valid - слага се преди обекта, за да Spring валидира според анотациите в съответния клас подадения обект. По подразбиране, когато възникне грешка при валидиране, се хвърля изключение Spring да връща код за състояние 400 Bad Request, но в случая е дефинирано поведение при грешка;
- @GetMapping е специализация на @RequestMapping аннотацията на Spring MVC и се използва за съпоставяне на HTTP GET заявка към конкретен метод на класа;

- @PostMapping също е специализация на @RequestMapping аннотацията на Spring MVC и се използва за съпоставяне на HTTP POST заявка към конкретен метод в клас на контролер. Методът се използва за обработване на формуляри с информация;
- Моделът (model) е структура от данни, която съдържа информация, която трябва да бъде показана от изгледа чрез template engine. Моделът се попълва с данни от контролера, който използва слоят на услугите, за да ги вземе, и след това се предава на изгледа, за да може да бъде показан в браузъра. В Spring MVC моделът обикновено се представя от класовете Model или ModelAndView. Когато изгледът се визуализира, той има достъп до данните на модела чрез контейнери или изрази, в зависимост от използваната технология за изглед, която в случая е Thymeleaf;
- @ModelAttribute е Spring MVC аннотация, използвана за свързване на параметър на метод или връщана стойност на метод към наименуван атрибут на модела. В имплементацията на приложението @ModelAttribute се използва за обвързване на данни при изпращане на формуляр, което ги преобразува към обект;
- @RequestParam е Spring MVC аннотация, която се използва за извличане на параметри на заявка или данни от формуляр, изпратени като част от HTTP заявка и свързване на параметър на заявка към параметър на метода, който я обработва в контролера. Анотацията @RequestParam може да се използва за указване на името на параметъра (value). Може да се дефинира и стойност по подразбиране, но това ще направи параметъра незадължителен, тъй като винаги ще се създава със своя defaultValue;
- Класът BindingResult представлява резултат от обвързване на данни, извършено между HTTP заявка и обект. Когато се изпрати формуляр, Spring MVC обвързва данните на формуляра с даден обект и класът

BindingResult се използва за улавяне на всички грешки, които възникват по време на този процес. Обектът BindingResult работи заедно с анотацията @Valid Ако обектът BindingResult съдържа някакви грешки, те могат да бъдат върнати на потребителя под формата на съобщения за грешка и/или използвани за контрол на потока на логиката на приложението. В настоящата изплементация се използват и двата подхода - връщат се грешки, но и се избягва записване на невалидни данни в базата данни, защото връщаните изгледи зависят от наличието на грешки при преобразуването на данните;

- RedirectAttributes е специален тип атрибут, който се използва за пренасяне на информация между HTTP заявки при пренасочване от един изглед към друг. Когато трябва да предадете данни към пренасочен URL адрес, можете да използвате RedirectAttributes, за да съхраните тези данни като флаш атрибути. Flash атрибутите са временни атрибути, които продължават само до следващата заявка и се премахват автоматично след това. Имплементирането му в приложението е с цел визуализиране на съобщения за грешки.

```
package org.elsys.diplom.controller;

@Controller
public class AuthController {
   // Autowired objects
   // Welcome, Login. Register Get Mappings

   @PostMapping("/register")
   public String doRegister(@ModelAttribute("userRegisterDto") @Valid UserRegisterDTO
userRegisterDto, BindingResult result, Model model){
      if(result.hasErrors()){
```

```
            model.addAttribute("userRegisterDto", userRegisterDto);
            return "register";
        }
        userService.addNewUser(userRegisterDto);
        model.addAttribute("userMail", userRegisterDto.getEmail());
        return "verify";
    }

    @RequestMapping(value = "/confirm-account", method = {RequestMethod.GET,
RequestMethod.POST})
    public String confirmUserAccount(@RequestParam("token") String confirmationToken,
RedirectAttributes redirectAttributes){
        if (userService.confirmAccount(confirmationToken)) {
            redirectAttributes.addFlashAttribute("successMessage", "Account verified
successfully");
        } else {
            redirectAttributes.addFlashAttribute("errorMessage", "Verification failed!");
        }
        return "redirect:/login";
    }
     // Other class methods
}
```

Фиг. 3.17

Методите @PostMapping(“login”) и @PostMapping(“logout”) се менажират от Spring Security и не е нужно да бъдат имплементирани в контролер. Тези филтри се добавят автоматично към филтърната верига от Spring Security (filterChain), когато е конфигурирана в SecurityConfiguration клас. Могат да бъдат персонализирани, техните настройки са показани в раздел “Имплементиране на сигурността” на фиг. 3.21

```
package org.elsys.diplom.controller;

@Controller
public class HomeController {
   // Autowired objects
   // Other class methods

   @PostMapping("/home")
   public String postHomePage(@ModelAttribute("newExpense") @Valid ExpenseDTO
newExpense, BindingResult result, Model model){
      if(result.hasErrors()){
         model.addAttribute("newExpense", newExpense);
         model.addAttribute("usersExpenses",
expenseService.getUsersExpenses(userService.retrieveLoggedInUser().getId()));
         model.addAttribute("categories", categoryService.getAllCategories());
         model.addAttribute("user", userService.retrieveLoggedInUser());
         return "home";
      }
      expenseService.addExpense(newExpense);
      return "redirect:/home";
   }

   @PostMapping("/filterStatistics")
   public String customFiltering(@ModelAttribute("filter") @Valid FilterExpensesDTO filter,
BindingResult result, Model model, RedirectAttributes redirectAttributes) {
      if(result.hasErrors()){
         model.addAttribute("filter", filter);
         model.addAttribute("user", userService.retrieveLoggedInUser());
         model.addAttribute("categories", categoryService.getAllCategories());
         return "customStatistic";
      }
      redirectAttributes.addFlashAttribute("customExp",
expenseService.customFilterExpenses(filter));
```

```
        return "redirect:/filterStatistics";
    }
}
```

Фиг. 3.18

3.3.4 Сигурност

Варианти за доставчик на услуги за удостоверяване са DAO, LDAP, OAuth, OpenID и други. Доставчикът за удостоверяване на DAO е персонализирано решение за удостоверяване на потребители с потребителска таблица в база данни, базирано на достъп до Dto обекти, като също така използва кодиране на пароли, докато LDAP, OAuth и OpenID са стандартни решения за удостоверяване на потребители, използващи външни доставчици на идентичност или услуги за директории.

На фиг. 3.19 и фиг. 3.20 са показани имплементациите на UserDetails и UserDetailsService. Това са интефейси, които Spring Security използва за автентикация и оторизация на потребители. Интерфейсът UserDetails представлява основната информация за потребителя, която се връща от DAO authentication provider-a. Той съдържа информация като потребителско име на потребителя, парола, права/роли и дали акаунтът е активиран, изтекъл или заключен. Този интерфейс се използва от Authentication Provider-a за конструиране на обект за удостоверяване, който представлява успешно удостоверяване. Интерфейсът UserDetailsService се използва от Spring Security за зареждане на специфични за потребителя данни за удостоверяване. Той предоставя един метод, loadUserByUsername(String username), който приема потребителско име като вход и връща имплементация на интерфейса UserDetails. Този метод обикновено се прилага за зареждане на потребителски данни от база данни.

```
package org.elsys.diplom.security;

public class CustomUserDetails implements UserDetails {
  private final User user;

  public CustomUserDetails(User user) {
  this.user = user;
  }
   // Override UserDetails methods
 }
```

Фиг. 3.19

```
package org.elsys.diplom.security;

public class CustomUserDetailsService implements UserDetailsService {
  @Autowired
  private UserRepository userRepository;

  @Override
  public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
      User user = userRepository.findByUsername(username);
      if (user == null) {
         throw new UsernameNotFoundException("User not found");
      }
      if (!user.isEnabled()){
         throw new UsernameNotFoundException("User not enabled");
      }
  return new CustomUserDetails(user);
  }
}
```

Фиг. 3.20

Самият конфигурационен файл за сигурността на приложението е показан на фиг. 3.21.

```
package org.elsys.diplom.security;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
  @Bean
  public UserDetailsService userDetailsService() {
     return new CustomUserDetailsService();
  }

  @Bean
  public BCryptPasswordEncoder passwordEncoder() {
     return new BCryptPasswordEncoder();
  }

  @Bean
  public DaoAuthenticationProvider authenticationProvider() {
     DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
     authenticationProvider.setUserDetailsService(userDetailsService());
     authenticationProvider.setPasswordEncoder(passwordEncoder());
     return authenticationProvider;
  }

  @Bean
  public SecurityFilterChain configure(HttpSecurity http) throws Exception {
     return http
          .csrf().disable()
          .authenticationProvider(authenticationProvider())
          .authorizeHttpRequests(auth -> {
```

```
            try {
                auth.requestMatchers(
                        // All permitted URL paths - login, register, confirm-password, etc)
                        new AntPathRequestMatcher("/welcome")).permitAll()
                    .anyRequest().authenticated().and()
                    .headers().frameOptions().disable();
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        })
        .formLogin(form -> {
            form.loginPage("/login");
            form.defaultSuccessUrl("/home", true);
            form.failureForwardUrl("/login");
            form.failureUrl("/login?error=true");
        })
        .logout(logout -> {
            logout.logoutUrl("/logout");
            logout.logoutSuccessUrl("/welcome");
            logout.clearAuthentication(true);
            logout.invalidateHttpSession(true);
            logout.deleteCookies("JSESSIONID");
        })
        .build();
  }
}
```

Фиг. 3.21

С помощта на този конфигурационен файл за сигурността се определят компонентите, които Spring да използва. Ключовите моменти в него са:

- @Configuration - показва, че даден клас е конфигурационен клас, предоставяйки дефиниции на компоненти и информация за

конфигурация в контекста на приложението Spring. Клас, анотиран с @Configuration, може да се използва заедно с други анотации, като @Bean, за дефиниране и конфигуриране на bean-ове в контекста на приложението;

- @EnableWebSecurity - аннотация, използвана за активиране на конфигурацията за уеб сигурност на Spring Security. Тя задейства зареждането на конфигурацията на Spring Security в контейнера при изграждане на проекта и настройва инфраструктурата за работа с уеб-базирана сигурност. Анотацията добавя конфигурацията WebSecurityConfiguration и springSecurityFilterChain bean към контекста на приложението;
- userDetailsService() – създава инстанция на CustomUserDetailsService, която предоставя потребителски подробности за Spring Security;
- passwordEncoder() – създава инстанция на BCryptPasswordEncoder, който се използва за кодиране на паролите;
- authenticationProvider() – създава нов DaoAuthenticationProvider, който е реализация на интерфейса AuthenticationProvider;
- configure(HttpSecurity http) - методът връща SecurityFilterChain, който е отговорен за филтриране на заявки и прилагане на политики за сигурност; в тази функция дефинират login() и logout();
- AntPathRequestMatcher е клас в Spring Security, който предоставя начин за проверяване на HTTP заявки, използвайки шаблони на пътища в стил Ant. Моделите на пътеки в стил Ant са прост начин за съпоставяне на пътеки в уеб приложения, подобно на заместващите символи, използвани в пътищата на файловата система;
- Деактивирането чрез headers().frameOptions().disable() е необходимо, за да се позволи достъп до конзолата за база данни H2.

3.3.5 Application.properties & env.properties

Application.properties и env.properties файловете се използват за екстернализиране на конфигурационни данни. Това включва данни като подробности за връзката с база данни, настройки на имейл сървъра и други специфични за приложението свойства. Съдържанието на Application.properties е показано на фиг. 3.22, а на фиг. 3.23 е представена структурата на env.properties файла. Той не е добавян в системата за контрол на версиите, тъй като съдържа чувствителна информация.

```
spring.config.import=env.properties

spring.datasource.url=jdbc:h2:file:./db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=${DB_USER}
spring.datasource.password=${DB_PASSWORD}

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=${EMAIL_USER}
spring.mail.password=${EMAIL_PASSWORD}
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Фиг. 3.22

- spring.datasource.url=jdbc:h2:file:./db - задава URL адреса за H2 базата данни, в този случай това е локален файл, наречен db в основната директория на приложението;
- spring.datasource.driverClassName=org.h2.Driver - задава класа на JDBC драйвера за базата данни H2;
- spring.datasource.username=${DB_USER} - задава потребителското име на базата данни, като взима стойността от на променливата DB_USER от файла env.properties. Действителната стойност е предоставена по време на изпълнение;
- spring.datasource.password=${DB_PASSWORD} - задава паролата на базата данни, като взима стойността от на променливата DB_PASSWORD. Действителната стойност се предоставя по време на изпълнение;
- spring.jpa.database-platform=org.hibernate.dialect.H2Dialect - задава диалекта на Hibernate за H2 базата данни;
- spring.jpa.generate-ddl=true - позволява автоматично генериране на схема на база данни;
- spring.jpa.hibernate.ddl-auto=update - конфигурира Hibernate автоматично да актуализира схемата на базата данни, когато е необходимо;
- spring.h2.console.enabled=true - активира конзолата за база данни H2, която предоставя уеб интерфейс за взаимодействие с базата данни в браузъра;
- spring.mail.host=smtp.gmail.com - задава името на хоста за имейл сървъра. В този случай това е SMTP сървърът на Gmail;
- spring.mail.port=587 - задава порта за имейл сървъра;
- spring.mail.username=${EMAIL_USER} - задава потребителското име на имейл акаунта на стойността на променливата EMAIL_USER, която

се взима от env.properties файла. Действителната стойност бива предоставена по време на изпълнение;

- spring.mail.password=${EMAIL_PASSWORD}: Това задава паролата за имейл акаунт на стойността на променливата EMAIL_PASSWORD, която се взима от env.properties файла. Действителната стойност ще бъде предоставена по време на изпълнение;
- spring.mail.properties.mail.smtp.auth=true - настройка за разрешаване на SMTP удостоверяване за имейл сървъра;
- spring.mail.properties.mail.smtp.starttls.enable=true - настройка за позволяване на използването на STARTTLS за имейл сървъра.

```
DB_USER=example
DB_PASSWORD=examplepass
EMAIL_USER=example@gmail.com
EMAIL_PASSWORD=yourgmailpass
```

Фиг. 3.23

3.4 Фронтенд

Потребителският интерфейс на приложението е реализиран с HTML, CSS и JavaScript в комбинация с Bootstrap, JQuery и Thymeleaf. Всяка HTML страница е изглед, който се визуализира с помощта на контролерния слой, описан в 3.3.

3.4.1 HTML

- Viewport - използва се за дефиниране на свойства на страницата. Такова е width, което може да бъде фиксирана големина в пиксели или широчината на устройството, както е в случая. Initial scale определя първоначалното ниво на мащабиране на страницата. Това

свойство може да бъде зададено на число между 0 и 10, като 1 е по подразбиране и означава без мащабиране;

- <link> тагове - за използване на външни ресурси за шрифтове и готови елементи на Bootstrap, както и персонализираните CSS файлове;
- За оформянето на страниците е използван bootstrap grid - класовете “container” и “row” са базови за страниците, а подредбата използва изрази като . Bootstrap grid има 12 колони. С този израз страницата използва по подразбиране 8 колони с отместване 2 колони, което фиксира осемте колони в средата (2-8-2), но когато прозорецът е с широчина по-голяма от 720 px (максималната ширина за контейнера при col-md), страницата използва 6 колони, отместени с 3, което отново поставя данните в центъра (3-6-3).

3.4.2 Thymeleaf

- xmlns:th - декларира Thymeleaf namespace, така че да може да се използва в HTML документа;
- th:if - клауза за проверка на дадено условие. Когато един таг има th:if клауза, то той ще се рендерира, само ако условието е изпълнено;
- th:unless - обратното на th:if. Прави проверка на дадено условие и тагът се рендерира, когато условието не е изпълнено;
- th:text - задава текстовото съдържание на елемент, като може да бъде и израз;
- th:href се използва за задаване на URL за елемент и може да приеме израз за своя стойност;
- th:object - указва обекта, който ще бъде обвързан с формата;
- th:field - специфицира към кое поле на обекта, дефиниран в th:object, да бъде обвързана стойността;

- th:action - използва се за указване на URL адреса, към който да се изпрати формата. Може да се използва за динамично генериране на URL адреси въз основа на данните на модела, тоест да приеме израз за своя стойност;
- th:errors - визуализира грешки при проверка за поле на форма. Работи с атрибута th:field и показва съобщение за грешка, ако полето има грешка при валидиране;
- th:value - използва се за обвързване на стойността на поле стойност на модел. Може да се използва в комбинация с th:field за задаване на стойността по подразбиране на input полето;
- #fields - помощен обект, който предоставя достъп до информация за полетата на формуляр, включително информация за всички грешки при валидиране, свързани с полетата;
- #numbers - помощен обект, който предоставя методи за форматиране и анализиране на числови стойности.

На фиг. 3.24 е представена кода на страницата за влизане в профила.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- BOOTSTRAP -->
  // link to Bootstrap
  <!-- GOOGLE FONTS  -->
  // links to Google fonts
  <!-- CUSTOM CSS -->
  <link rel="stylesheet" href="css/style.css">
```

```
  <link rel="stylesheet" href="css/buttons.css">
  <link rel="stylesheet" href="css/login.css">
  <title>Login to Smart Money</title>
</head>

<body>
  <main>
    <div class="container">
      <div class="row">
        <div class="col-md-6 offset-md-3 col-8 offset-2">
          <div class="form-holder">
            <h1>Login</h1>
            <div th:if="${param.error}">
              <div class="error">Invalid Username or Password</div>
            </div>
            // other th:if checks for error/success messages
            <form id="login-form" th:action="@{/login}" method="post">
              <div class="holder">
                <label for="username">Enter your username:</label><br>
                <input type="text"
                    id="username"
                    name="username"
                    placeholder="Your username" >
              </div>
              <div class="holder">
                <label for="password">Enter your password:</label><br>
                <input type="password"
                    id="password"
                    name="password"
                    placeholder="Your password" >
              </div>
              <button type="submit" class="button">Login</button>
            </form>
```

```
                <p>Forgot your password?</p>
                <a th:href="@{/forgot-password}" class="cyan-outline-btn"
id="outline-reset">Reset Password</a>
                <p>New to Smart Money?</p>
                <a th:href="@{/register}" class="cyan-outline-btn"
id="outline-register">Register</a>
            </div>
        </div>
    </div>
  </main>
</body>
```

Фиг. 3.24

На фиг. 3.25 е представена част потребителската начална страница (home page), която включва използването на #fields и хедъра на страницата, който използва JavaScript.

```
<!DOCTYPE html>
<!-- head tags ->
<body>
  <header>
    <div class="container">
      <nav class="navbar">
        <a href=""><img class="logo" src="/images/logo.svg" alt="logo"></a>
        <ul class="nav-menu">
          <li class="nav-item">
            <a th:href="@{/home}" class="nav-link">Home</a>
          </li>
          <li class="nav-item">
            <a th:href="@{/statistics}" class="nav-link">My Statistics</a>
          </li>
```

```
            <li class="nav-item">
                <a th:href="@{/filterStatistics}" class="nav-link">Filter Expenses</a>
            </li>
            <li class="nav-item">
                <a th:href="@{/logout}" class="nav-link">Logout</a>
            </li>
        </ul>
        <div class="hamburger">
            <span class="bar"></span>
            <span class="bar"></span>
            <span class="bar"></span>
        </div>
    </nav>
  </div>
</header>
<body>
<!-- code ->
  <div th:if="${#fields.hasErrors('newExpense')}"
    th:errors = "*{newExpense}"
    class="error">
 ] </div>
<!-- code ->
</body>
<script src="/js/menu.js"></script>
```

Фиг 3.25

На фиг. 3.26 е показана част от имплементацията на страницата за статистика по подразбиране (за месец и година по категории). Тя отговаря за генерирането на pie chart-овете, които се създават с помощта на Chart.js библиотеката, и използва #numbers обекта.

```
<div th:if="${monthExp.size() > 0}">
  <canvas id="lastMonthChart" ></canvas>
  <script th:inline="javascript">
  var month = [[${monthExp}]];
  createChart("lastMonthChart", month, "Last Month Expenses");
  </script>
</div>
<div class="background" th:if="${monthExpTotal}">
   <p>Your expenses for last month are <span
th:text="${#numbers.formatDecimal(monthExpTotal, 0, 'COMMA', 2,
'POINT')}"></span></p>
</div>
<script src="/js/chart.js"></script>
```

Фиг. 3.26

3.4.3 JavaScript

На фиг. 3.27 е показан кода на menu.js, който се използва за промяна на навигационния бар на страницата към хамбургер меню.

```
const hamburger = document.querySelector(".hamburger");
const navMenu = document.querySelector(".nav-menu");

hamburger.addEventListener("click", () => {
  hamburger.classList.toggle("active");
  navMenu.classList.toggle("active");
})
```

Фиг. 3.27

На фиг. 3.28 е представен form.js, който се използва за промяна на потребителския интерфейс при въвеждане на нов разход спрямо неговия вид.

```
var startDate = document.getElementById("start-date")
var endDate = document.getElementById("end-date")

function updateForm(){
  var type = document.getElementById("expense-type").value;
  if(type == "one-time"){
    document.getElementById("start-date-label").innerHTML = "Date";
    document.getElementById("end-date-label").style.display = "none";
    endDate.style.visibility = "hidden";
    if(document.getElementById("error-end-date") != null){
      document.getElementById("error-end-date").style.display = "none";
    }
    startDate.addEventListener("change", function(){
      endDate.value = startDate.value;
    });
  }
  else{
    document.getElementById("end-date-label").style.display = "block";
    endDate.style.visibility = "visible";
    startDate.removeEventListener("change", function(){
      endDate.value = startDate.value;
    });
  }
}
```

Фиг. 3.28

Четвърта глава
Ръководство на потребителя

4.1 Инсталиране на необходимия софтуер

4.1.1 Java

За инсталиране трябва да се посети сайтът на Oracle - https://www.oracle.com/java/technologies/downloads/#java17 . На този линк трябва да се вземе правилният архив за операционната система на потребителя. Инсталацията се състои от пускане на инсталационните файлове и следване на инструкциите по време на инсталация.

4.1.2 Intellij Idea Community

Инсталирането на Intellij Idea става от сайта на JetBrains - https://www.jetbrains.com/idea/download/#section=linux , като потребителят трябва да избере Community или Ultimate edition. Ultimate е платената версия на средата за разработка, но има 30 дни неплатено използване. Потребителят трябва да се съобрази с операционната си система. Даденият линк е за Линукс-базирани операционни системи, средата може да бъде използвана и върху MacOS и Windows. Инсталацията се състои от пускане на инсталационните файлове и следване на инструкциите по време на инсталация. Проектът може да бъде използван и в друга среда, поддържаща Java като език за програмиране.

4.2 Процедура по пускане на приложението

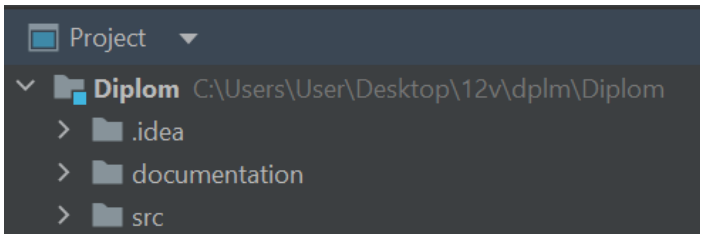
4.2.1 GitHub

Потребителят трябва да влезе в GitHub. Ако няма регистрация там, трябва да си направи такава. След това трябва да посети следния линк: https://github.com/AleksandraAleksandrova/Diplom. След това трябва да

отвори терминал и да навигира до директорията, в която иска да постави проекта, например Desktop/SpringBoot. След това трябва да изпълни командата:

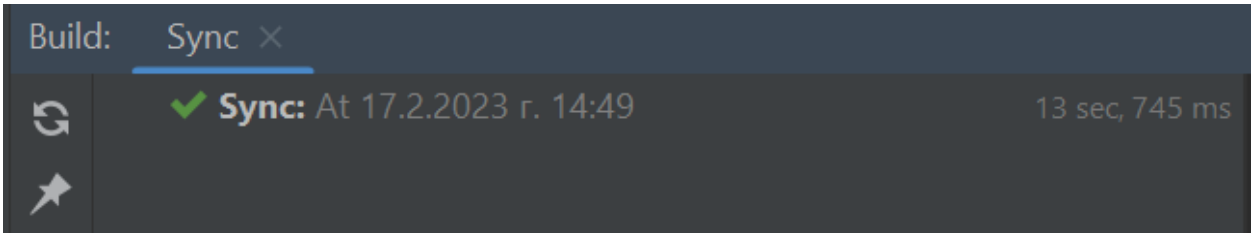
```
git clone https://github.com/AleksandraAleksandrova/Diplom.git
```

Потребителят трябва да зареди проекта като Maven проект, ако средата за разработка не го прави по подразбиране. Проектът е зареден, когато има синя точка на папката му, както е показано на фиг. 4.1.



Фиг. 4.1

После трябва да се изчака Maven да подготви и зареди всички зависимости към проекта (dependencies). Когато всички зависимости са синхронизирани и готови, в Build прозореца ще бъде изписано “Sync” като на фиг. 4.2.



Фиг. 4.2

4.2.2. Environmental properties

Потребителят трябва да напише свой собствен env.properties файл в директорията на application.properties файла (папка resources), който дефинира чувствителната информация за проекта - потребителски имена и пароли за базата данни и имейл услугата.

4.2.3 Application properties

Този файл няма нужда от директна промяна, но ако потребителят иска да променя конфигурацията на имейл услугите и/или на базите данни, трябва да промени application.properties файла, според описанието му в точка 3.3.

4.2.4. Пускане на проекта

Проектът трябва да бъде отворен и зареден като Maven проект в Intellij Idea. Един проект е зареден, когато основната му папка е означена със синя точка. Ако има промени в pom.xml, той трябва да бъде прекомпилиран. Прекомпилирането става с натискане на иконката в горен десен ъгъл. Пускането на проекта може да се случи по някой от следните начини:

- През основния файл на приложението DiplomApplication.java - пуска се функцията DiplomApplication(), която съдържа main() функцията;
- С терминал - трябва да се използват следните команди (за Ubuntu), показани на фиг. 4.3.

```
projectDirectory: mvn clean install
projectDirectory: mvn clean build
projectDirectory: mvn spring-boot:run
```

Фиг. 4.3.

- Създаване на Run/Debug конфигурация - задават се име, версия на Java, аргументи и други настройки на конфигурацията, след което се натиска Run;

Заключение

Успешно е разработена уеб система за управление на личните финанси. Потребителят може да се регистрира или да влезе в профила си, като не може да създаде профил без валиден уникален имейл. След това успешно може да създава нови разходи и получава нагледна статистика за тях, като може да филтрира разходите си за период от време и категория.ю

Бъдещото развитие на проекта включва следните функционалности и подобрения:

- Възможност за редактиране на разходи;
- Възможност за триене на разходи;
- Възможност за определяне на месечно ограничение на категориите, което да представлява максимална сума от разходи за дадена категория. Преминаването на тази граница да води до имейл уведомление, а с помощта на това ограничение да се направи прогрес бар, който да показва каква част от “разрешената” сума е изхарчена до момента;
- Възможност за въвеждане на приходи и с тяхна помощ да се смята месечния баланс на потребителя;
- Deployment на приложението в EC2 услугата на Amazon.

Използвани източници

[1] Консуматорското общество - Консуматорско общество

[2] Expensify - https://www.cnbc.com/select/expensify-app-review/

[3] GoodBudget - https://goodbudget.com/what-you-get/

[4] MySQL - https://www.oracle.com/mysql/what-is-mysql/

[5] PostgreSQL- https://www.postgresql.org/about/

[6] H2 Database Engine - https://www.h2database.com/html/main.html

[7] Multiversion concurrency - What is MVCC? How does it work?

[8] What is Spring Boot? - https://stackify.com/what-is-spring-boot/

[9] Spring Boot - Spring Boot Tutorial

[10] What is Hibernate? - https://www.javatpoint.com/hibernate-tutorial

[11] Spring Data JPA - What is Spring Data JPA and why to choose it

[12] Apache Tomcat - https://tomcat.apache.org/

[13] Thymeleaf - https://www.thymeleaf.org/

[14] Thymeleaf in Spring - https://www.baeldung.com/thymeleaf-in-spring-mvc

[15] HTML - https://www.hostinger.com/tutorials/what-is-html

[16] CSS - https://www.hostinger.com/tutorials/what-is-css

[18] IntelliJ IDEA - https://www.jetbrains.com/idea/

[19] Visual Studio Code - https://code.visualstudio.com/

[20] Live Server - Live Server - Visual Studio Marketplace

Съдържание