Санкт-Петербургский Государственный Университет

Математико-механический факультет

Математическое обеспечение и администрирование информационных систем

Степырев Даниил Федорович

Создание веб-приложения с использованием технологий Java Spring

Отчет по учебной практике

Научный руководитель:

к.т.н., доцент Брыксин Т.А.

Санкт-Петербург

2020

Оглавление

Введение

Spring MVC является одним из самых популярных фреймворков для разработки веб-приложений[1]. Неудивительно, что для создания почти любого веб-приложения на Java требуются знания Spring. Поэтому на многих стажировках на вакансию Java-разработчика необходимы знания этого фреймворка[2].

Чтобы изучить Spring, было принято решение создать собственный проект. Мотивация заключалась в нескольких причинах. Во-первых, после завершения семестровой работы будет готовый проект, который можно включить в резюме при отправлении заявки на стажировку. Во-вторых, в течение всей работы предстояло тесное знакомство с технологиями Spring, знание которых будет большим плюсом при устройстве на стажировку.

Форматом приложения был выбран блог с сообщениями, планировалось создать что-то похожее на Twitter[3]. Мотивация к выбору такого формата заключалась в том, что блоги становятся популярными в настоящее время.

Постановка задачи

Целью данной работы является создание веб-приложения с возможностью добавления новых постов, регистрацией, авторизацией и профилем пользователя. Проект коллективный: в нем участвуют два человека.

Для достижения поставленной цели от команды требуется:

- разработать архитектуру приложения;
- разработать дизайн приложения;
- добавить базу данных;
- разработать страницу регистрации и авторизации;
- разработать страницу, позволяющую добавлять и редактировать посты;
- разработать страницу, отвечающую за профиль пользователя;

[1] https://www.jrebel.com/blog/java-web-framework-usage-stats

[2] https://www.quora.com/What-frameworks-and-skills-must-a-junior-Java-developer-be-familiar-with-to-land-a-job-in-a-finance-company

[3] https://twitter.com/

Для достижения поставленной цели от автора курсовой работы требуется:

- разработать архитектуру приложения;
- разработать дизайн нескольких страниц приложения;
- реализовать авторизацию и регистрацию пользователей;
- реализовать изменение добавленных постов;
- добавить базу данных.

От второго участника требуется:

- разработать дизайн нескольких страниц приложения;
- реализовать добавление новых постов;
- реализовать профиль пользователя и администратора;
- реализовать отправщик писем для подтверждения регистрации.

1. Архитектура приложения

Перед началом реализации приложения была продумана его структура и основные страницы. Было решено добавить регистрацию новых пользователей, авторизацию существующих и изменение данных в аккаунте.

Веб-приложение можно разделить на несколько отдельных сущностей:

- окно регистрации новых пользователей;
- окно авторизации;
- окно сообщений;
- окно профиля пользователя.

Первые два отвечают за регистрацию и авторизацию, следующее – за добавление сообщений, последнее – за изменение данных аккаунта.

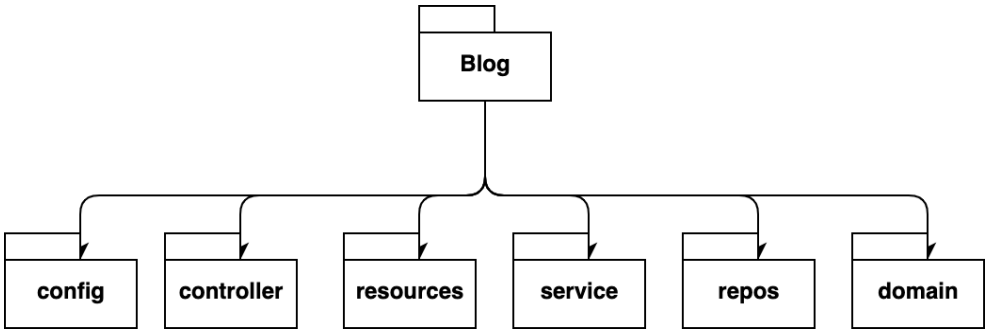


Рис. 1: Структура приложения

Компоненты приложения было решено разделить на несколько пакетов (рис. 1).

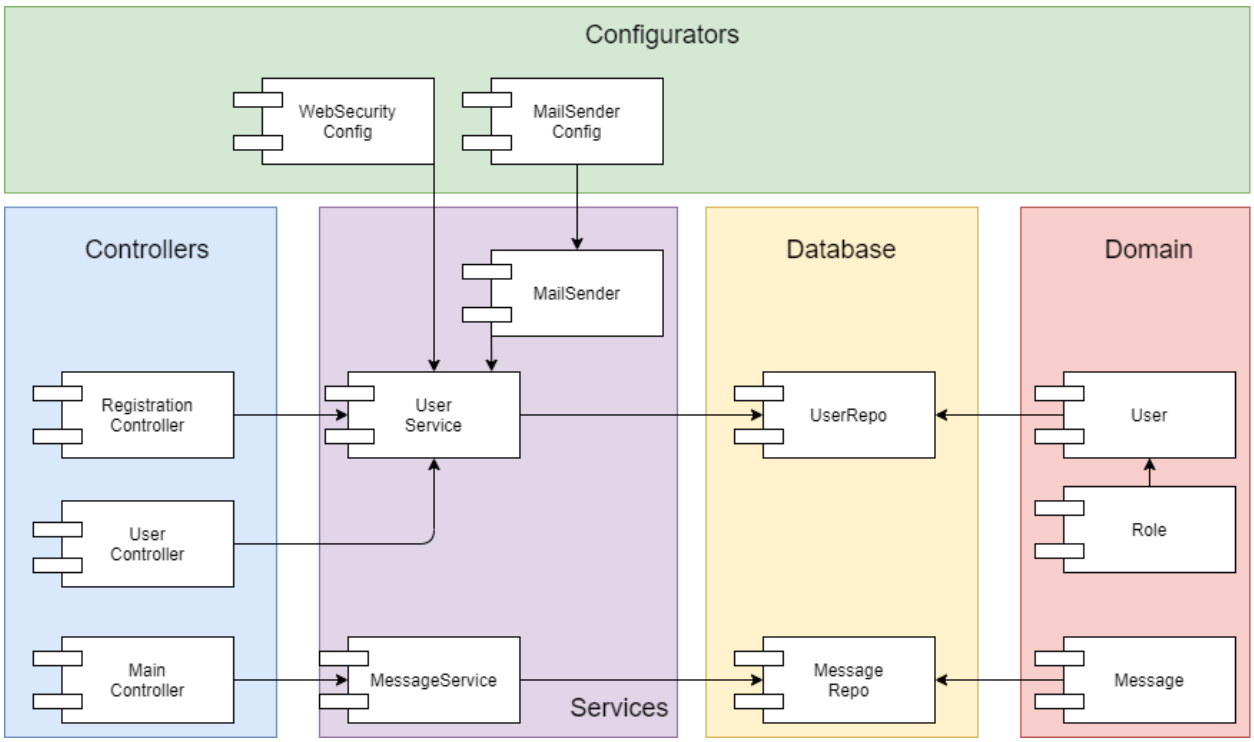


Рис. 2: Архитектура приложения

Архитектура приложения может быть описана с помощью следующей диаграммы (рис. 2).

Пакет *config* отвечает за настройку конфигураций сервисов.

В папке *controller* находятся классы, реализующие логику приложения. MainController отвечает за основное окно с постами: добавление, отображение и фильтрацию сообщений. RegistrationController реализует добавление нового пользователя в приложение и активацию уже существующего. UserController отвечает за окно администратора (получение списка всех пользователей, изменение пользователей) и реализует логику, доступную простому пользователю: отобразить профиль, подписаться/отписаться, получить список всех подписок/подписчиков.

В пакете *domain* располагаются классы, реализующие сущности приложения. Message – класс, отвечающий за сообщение; в Role хранятся доступные в приложении роли пользователей (на данный момент доступны следующие роли: простой пользователь, администратор); User – класс, реализующие сущность пользователя.

В папке *repos* находятся интерфейсы взаимодействия с базами данных (MessageRepo – база, в которой хранятся посты; UserRepo – база, в которой хранятся добавленные пользователи).

В пакете *service* располагаются классы, реализующие логику сущностей. В MailSender реализованы методы отправки сообщения на почту для подтверждения регистрации, в UserService находятся методы, позволяющие взаимодействовать с базой данных пользователей, MessageService отвечает за отображение страницы с постами.

В пакете *resources* находятся html-коды страниц приложения.

Таким образом, все классы, отвечающие за логику приложения, располагаются в пакете *controller*, а графическая составляющая проекта находится в папке *resources*.

2. Дизайн приложения

Когда были подготовлены первые наброски, стало понятно, что в проекте надо будет заниматься разработкой графической составляющей приложения. Однако хотелось уделить большее внимание разработке функциональностей приложения, так как это являлось главной целью проекта. Эту дилемму удалось разрешить благодаря использованию шаблонизатора (программное обеспечение, позволяющее использовать html-шаблоны для создания html-страниц).

В качестве шаблонизатора был использован FreeMarker [1]. Именно этот шаблонизатор был выбран по ряду причин. Во-первых, FreeMarker – это свободное программное обеспечение, что позволяет использовать его в проекте. Более того, FreeMarker является одним из самых популярных шаблонизаторов при написании приложений на Spring[4]. Помимо этого, FreeMarker был проще в использовании, чем его конкуренты (к примеру, Thymeleaf[5]), но, несмотря на это, сильно не уступал в функциональности. Например, в этом шаблонизаторе реализована поддержка макросов[6], что позволило переиспользовать код реализации пагинации.

Чтобы страницы приложения не выглядели как прямоугольные формы на белом фоне, используется CSS[7] (язык стилей, с помощью которого придается стиль отображения html-страниц). В проекте было решено использовать Bootstrap [2]. Была использована именно эта технология, потому что Bootstrap является свободным программным обеспечением, а также достаточно легок в освоении и очень популярен при разработке дизайна приложений[8].

Благодаря этому фреймворку отпала необходимость в изучении CSS, что позволило уделить большее внимание реализации функциональности приложения.

[4] https://www.baeldung.com/spring-template-engines
[5] https://www.thymeleaf.org/
[6] https://freemarker.apache.org/docs/ref_directive_macro.html
[7] https://www.bigcommerce.com/ecommerce-answers/what-css-and-why-it-important/
[8] https://www.htmlgoodies.com/html5/markup/10-common-uses-of-bootstrap.html

3. База данных

В приложении было необходимо использовать несколько таблиц: для хранения данных о пользователях и для добавленных сообщений. Так как основная идея проекта заключалась в изучении технологий Spring, для работ с базами данных было решено использовать библиотеку Spring Data JPA[9], которая позволяет легко взаимодействовать с базами данных. В качестве базы данных была выбрана PostgreSQL[10]. Эта база данных была выбрана по нескольким причинам: во-первых, она является свободным программным обеспечением, и, во-вторых, является одной из самых популярных баз данных[11].

Для упрощения взаимодействия с базой данных была использована ORM-библиотека (технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования) Hibernate[12]. В проекте была использована эта библиотека, потому что она находится в свободном доступе и является одной из самых популярных библиотек при работе с Spring[13].

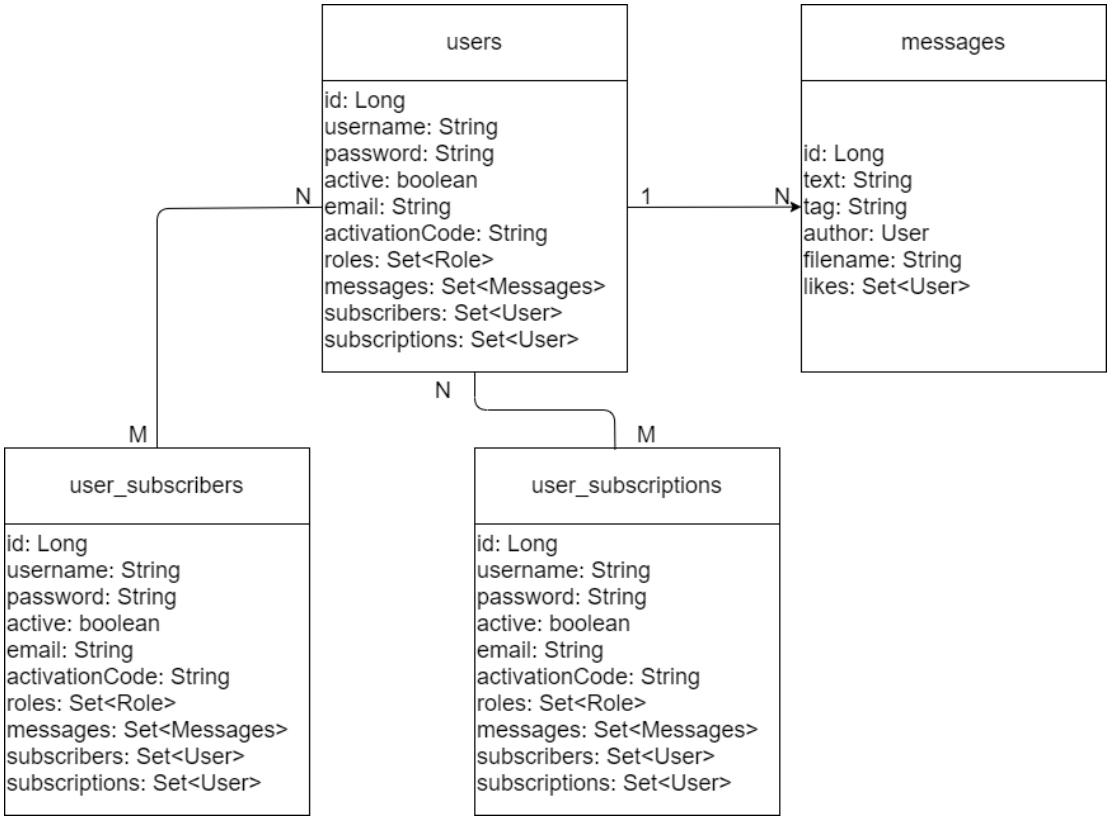


Рис. 3: Структура базы данных

[9] https://spring.io/projects/spring-data-jpa
[10] https://www.postgresql.org/
[11] https://365datascience.com/mysql-postgresql/
[12] https://hibernate.org/
[13] https://www.mindfiresolutions.com/mindfire/Java_Hibernate_JDBC.pdf

При использовании Hibernate можно создавать сущности, которыми будет потом наполняться база данных. Для работы с базой данных были созданы две сущности: сообщение и пользователь (рис. 3). Пользователь хранит в себе регистрационные данные об аккаунте, роль, список сообщений, подписок и подписчиков. Сообщение хранит текст, тег, автора и список отметок “нравится”.

Используя Hibernate, можно устанавливать связи между сущностями, что позволяет упростить работу с базой данных (сущность хранит не ссылку на другой элемент в таблице, а сам объект: например, пользователь хранит не множество id сообщений, а множество объектов сообщений). При этом можно настраивать тип связи[14].

В проекте было решено установить следующие связи:

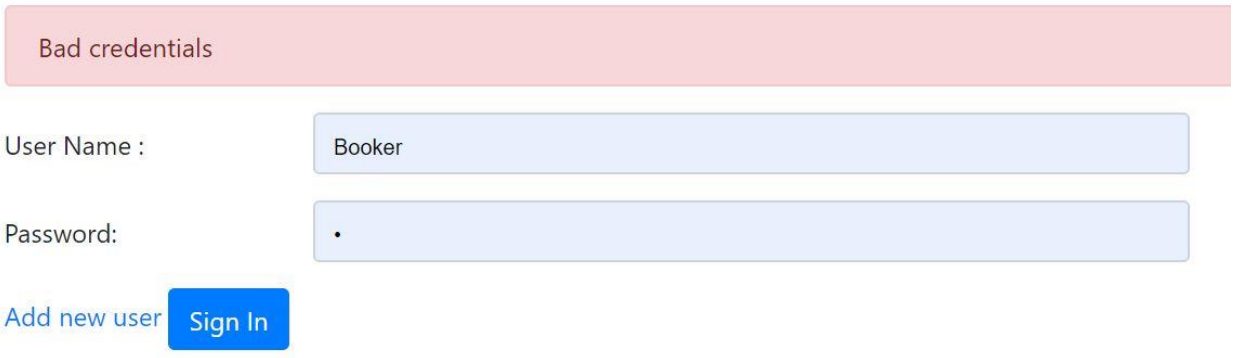
- OneToMany между пользователем и сообщением. Такая связь позволяет хранить множество сообщений, опубликованных текущим пользователем;
- ManyToMany между пользователем и подписчиками, пользователем и подписками (множество других пользователей). Такая связь удобна, потому что пользователь может быть подписан не нескольких других пользователей, ровно, как и обратное.

4. Регистрация и авторизация

Изначально регистрация и авторизация пользователей были реализованы вручную. Однако впоследствии было найдено готовое решение этой задачи с использованием технологий Spring Security[15].

4.1. Авторизация пользователя

Окно авторизации в проекте состоит из нескольких элементов: поля для ввода имени пользователя и пароля, кнопки подтверждения авторизации и кнопки добавления нового пользователя.



[14] https://www.baeldung.com/spring-data-rest-relationships
[15] https://spring.io/projects/spring-security

Рис. 4: Оповещение об ошибке при авторизации

При неудачной попытке входа пользователю сообщается об ошибке (рис. 4).

4.2. Регистрация со стороны пользователя

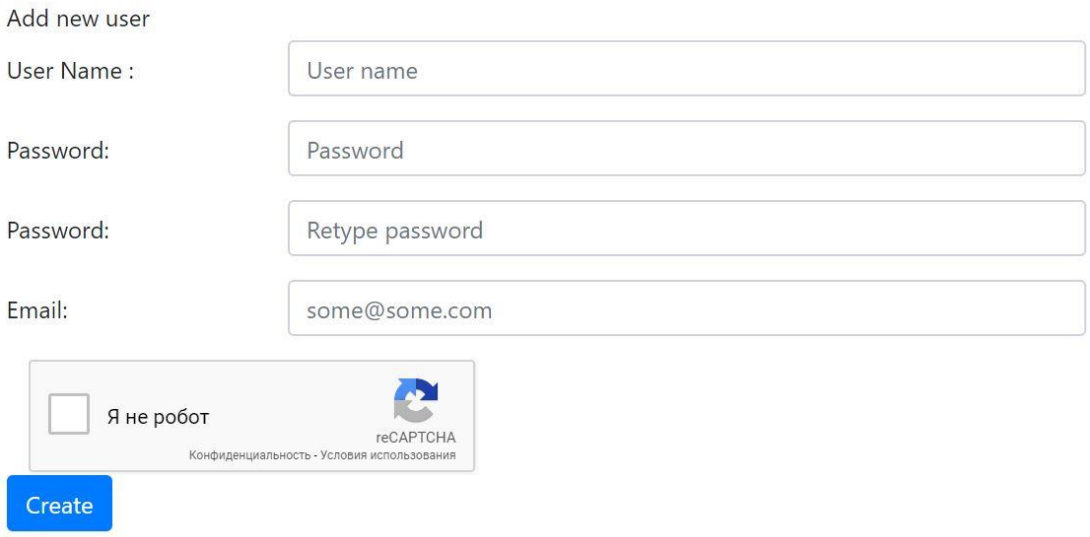


Рис. 5: Окно регистрации

Окно регистрации нового пользователя содержит несколько элементов: поля для ввода имени пользователя, паролей, почты и captcha (рис. 5).

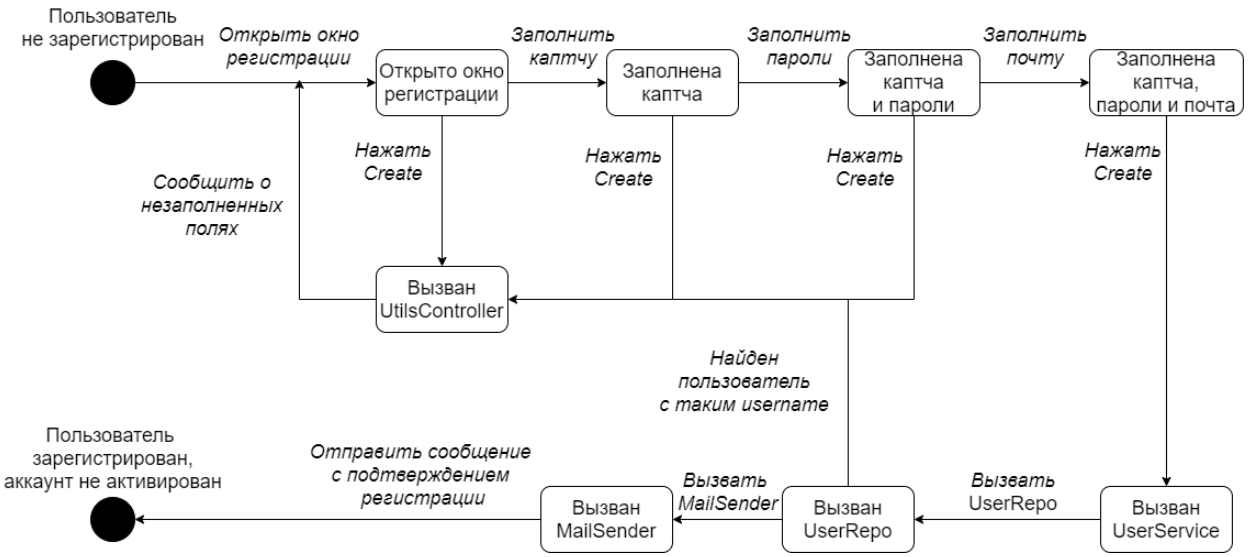


Рис. 6: Регистрация нового пользователя

Регистрация нового пользователя может быть описана с помощью следующей диаграммы (Рис. 6).

Сначала новый пользователь должен заполнить данные в окне регистрации: ввести имя пользователя, два раза пароль, электронную почту и заполнить capctha. После того, как пользователь нажмет кнопку подтверждения регистрации, начнется проверка заполненных данных.

Первым делом происходит проверка на стороне клиента: сравниваются на совпадение пароли, проверяется заполнение capctha, почты пользователя. Если хотя бы один из этих пунктов не был выполнен, вызывается UtilsController, сообщающий пользователю о допущенных ошибках. В противном случае продолжается проверка данных уже на стороне сервера.

Если проверки на стороне клиента были успешно пройдены, то вызывается UserService, который проверяет, не существует ли уже в базе данных пользователь с таким именем. Если такой был найден, то процесс регистрации прерывается, пользователь оповещается об ошибке. Иначе для нового пользователя генерируется активационный ключ, который отправляется на указанный адрес с помощью MailSender.

Второй этап регистрации заключается в активации созданного аккаунта. Новому пользователю необходимо перейти по ссылке, указанной в письме. После перехода вызывается запрос в базу данных пользователей, проверяющий существует ли пользователь с таким именем, и совпадают ли активационные ключи. В случае успешной проверки аккаунт становится активированным, и можно авторизоваться в приложении, иначе пользователю сообщается о возникшей проблеме.

4.3. Реализация регистрации

С помощью технологий Spring Security можно использовать уже готовый контроллер для системы авторизации. Для этого в классе MvcConfig необходимо было установить шаблон, который будет использоваться при отображении страницы, а также ссылку, по которой будет отображаться регистрация.

Затем необходимо было разграничить страницы, доступные для авторизованного и неавторизованного пользователя. WebSecurityConfig конфигурирует приложение при запуске следующим образом: для вступительной страницы, страницы регистрации и активации доступ предоставляется всем пользователям, не требуя быть авторизованным. Для всех остальных страниц будет требоваться авторизация.

Однако со стандартным контроллером возникла проблема: после регистрации он возвращал менеджер, обслуживающий учетные записи.

Такую реализацию было не очень удобно использовать: приходилось вручную добывать информацию о каждом пользователе и отдельно добавлять это в базу данных. Так возникла идея переписать контроллер и добавить сервис по работе с пользователем в приложение.

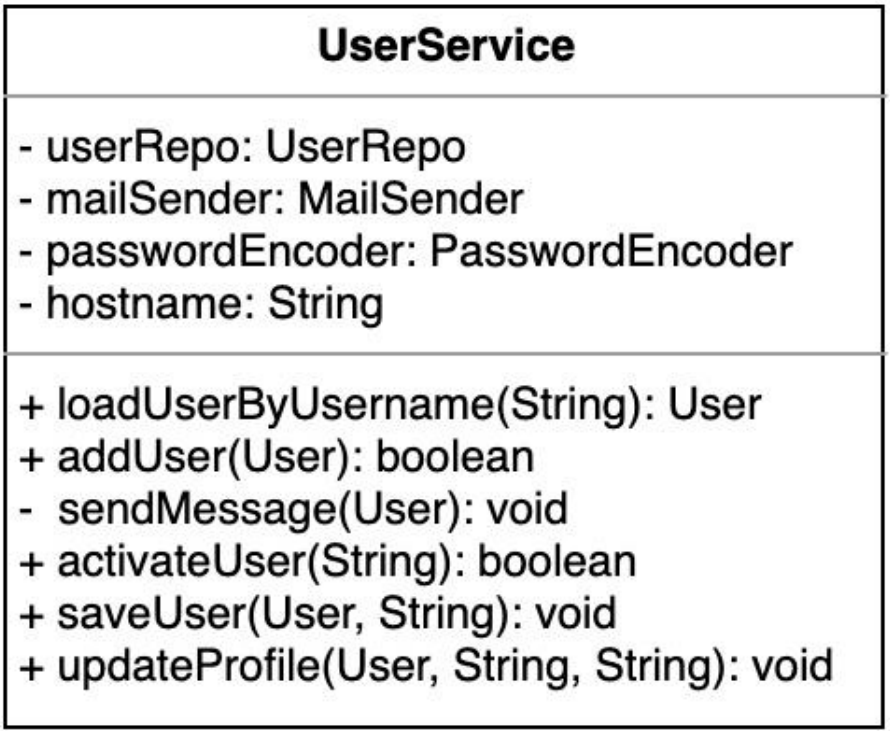


Рис. 7: Основные поля и методы класса UserService

Самое первое, что необходимо было сделать – настроить WebSecurityConfig на работу с базой данных пользователей. Для этого был создан класс UserService (рис. 7), в котором были реализованы методы взаимодействия с базой данных пользователя: поиск по имени, добавление, отправление сообщений, активация и обновление профиля.

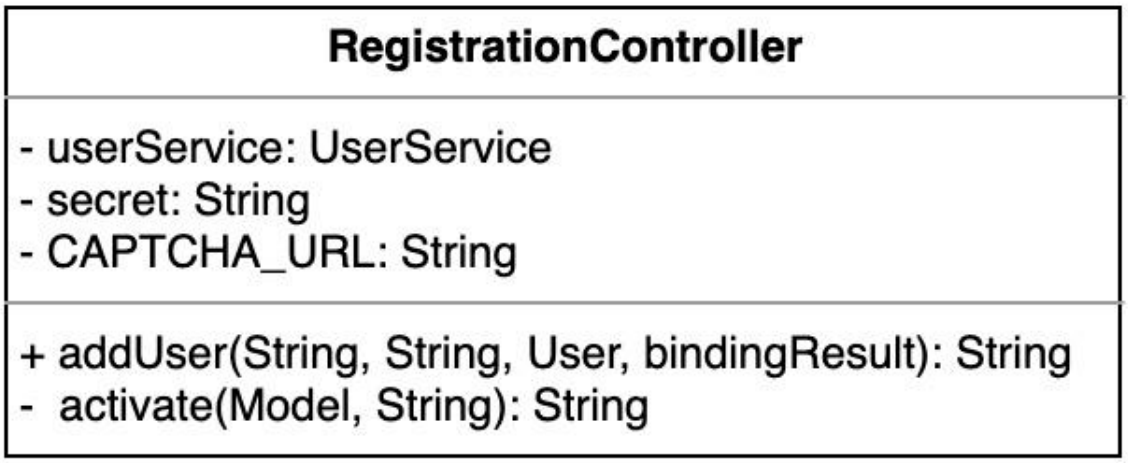


Рис. 8: Основные поля и методы класса RegistrationController

После настройки WebSecurityConfig был создан класс RegistrationController (рис. 8), в котором находилась реализация регистрации.

При добавлении нового пользователя принималось несколько параметров: второй введенный пароль, ответ на заполнение captcha, пользователь, которого необходимо добавить, и bindingResult – список ошибок при проверке входящих данных. Сначала происходит проверка на стороне клиента: была ли заполнена captcha, совпали ли пароли (первоначальный пароль узнается у самого пользователя с помощью метода getPassword()). Если эти проверки завершились неудачей, или же существовали ошибки входящих данных, то вызывается UtilsController, который сообщает пользователю о проблеме.

Затем начинается проверка на стороне сервера: происходит попытка добавить пользователя в базу данных с помощью UserService: изначально в базе данных пользователей проходит поиск по имени: если такой пользователь был найден – возвращается false, что сообщает о неудаче при регистрации; в противном случае пользователь сохраняется в базе данных и возвращается true. При сохранении пользователя генерируется активационный ключ, который отправляется на указанный адрес. Пока в базе данных у пользователя на месте ключа не будет пустой строки, авторизация не может быть выполнена.

В случае неудачной регистрации пользователь перемещается на страницу логина, где сообщается о допущенных ошибках, иначе регистрация продолжается.

При активации аккаунта пользователя требуется несколько параметров: модель, куда помещается информация о результате активации и строка, содержащая активационный код, с которого был выполнен переход. С помощью UserService происходит попытка активации: устраивается поиск в базе данных пользователей по активационным ключам, если пользователь был найден, то его активационный ключ зануляется и возвращается true, в противном случае – false. Если UserService вернул true, то в модель добавляется сообщение об успешном прохождении регистрации, которое отображается пользователю, в противном случае – сообщение об ошибке. Пользователь перенаправляется на страницу авторизации**.**

4.4. Кэширование паролей

Изначально данные новых пользователей не изменялись никаким образом при сохранении в базе. Такая реализация влекла существенную

проблему: слабая безопасность. Если злоумышленники получили бы доступ к базе пользователей, то у них автоматически оказались бы данные всех зарегистрированных людей.

Чтобы исправить этот недочет, было предпринято решение кэшировать пароли перед их сохранением. Для реализации был использован класс библиотека Spring под названием Password Encoder[16]. Теперь при регистрации в базу пользователей добавляется закэшированный пароль. Проверка на совпадение паролей при авторизации была реализована следующим образом: введенный пользователем пароль кэшируется и сравнивается с паролем из базы.

[16] https://docs.spring.io/spring-security/site/docs/4.2.4.RELEASE/apidocs/org/springframework/security/crypto/password/PasswordEncoder.html

5. Основное окно сообщений

Рис. 9: Добавление нового сообщения

Основное окно сообщений отвечает за добавление новых постов и отображение уже существующих. У каждого поста указывается (Рис. 9):

- текст сообщения;
- картинка (если она есть);
- тег;
- имя пользователя, который добавил это сообщение;
- отметки “нравится” и их количество;
- кнопка редактирования сообщения (если пост был добавлен текущим пользователем).

5.1. Редактирование сообщений

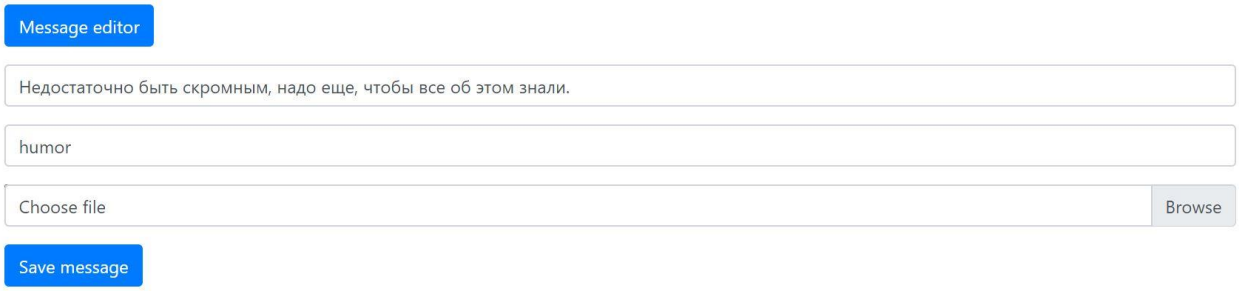


Рис. 10: Редактор сообщений

Изначально в приложении нельзя было изменять добавленные ранее сообщения, что было крайне не гуманно по отношению к пользователю. Поэтому было решено исправить это и добавить редактор текстовых сообщений (Рис. 10). Теперь пользователь может нажать кнопку “Edit” и изменить текст сообщения, тег или же загрузить другую картинку.

Редактирование добавленного сообщения происходит по следующему принципу: изначально происходит проверка на совпадение автора сообщения и пользователя, пытающегося отредактировать это сообщение. Если проверка была пройдена, то текст сообщения и тег могут быть успешно заменены, иначе сообщение не изменится.

5.2. Пагинация

Во время тестирования работы приложения возникла проблема: все сообщения отображались на одной странице. Во-первых, было крайне неудобно листать такой большой список. Другая негативная сторона состояла в том, что при добавлении большого количества постов загрузка главной страницы замедлялась.

Решение данной проблемы – разделение всех постов на страницы с ограниченным числом сообщений. Однако из этого решения вытекала другая проблема: сколько сообщений надо отображать на каждой странице? Так возникла необходимость в динамическом изменении отображаемого числа сообщений.

Казалось, задача была решена: посты разделялись на страницы, количество сообщений на странице варьировалось. Однако была не просчитана одна ситуация: если загруженных сообщений будет много, а количество постов на странице будет ограничено маленьким числом, сколько страниц будет отображаться? К примеру, для 100 сообщений разделение по 2 сообщения на страницу приведет к 50 страницам. А так как отображаются все страницы, это приведет к тому, что список страниц уйдет за границы экрана. Таким образом, возникла необходимость в динамической подгрузке списка страниц.

Было решено ограничить список страниц числом 7. Если всего страниц получалось меньше 7 – отображается весь список, иначе необходимо отобразить некоторые страницы, к примеру, для 3 страницы: две страницы до текущей, текущую, две страницы после и заключительную.

Рис. 11: Перелистывальщик страниц

Реализованный таким образом “pager” (перелистывальщик страниц) стал удобен в использовании и не занимал много пространства на экране (рис. 11).

6. Дополнительные окна

Помимо окон регистрации, авторизации и сообщений в проекте реализованы несколько дополнительных окон: окно сообщений пользователя, окно профиля пользователя и окно администратора.

6.1. Окно сообщений пользователя

Когда был реализован редактор сообщений, возникла небольшая проблема: как пользователь найдет свои сообщения? Безусловно, можно было искать по страницам сайта или же воспользоваться фильтром по тэгу, но это было не очень удобно, так как сообщений могло быть очень много. Поэтому было решено добавить дополнительное окно, где будут отображаться сообщения, опубликованные с текущего пользователя.

В этом окне можно редактировать сообщения, как и на основной странице с постами, также реализована поддержка пагинации.

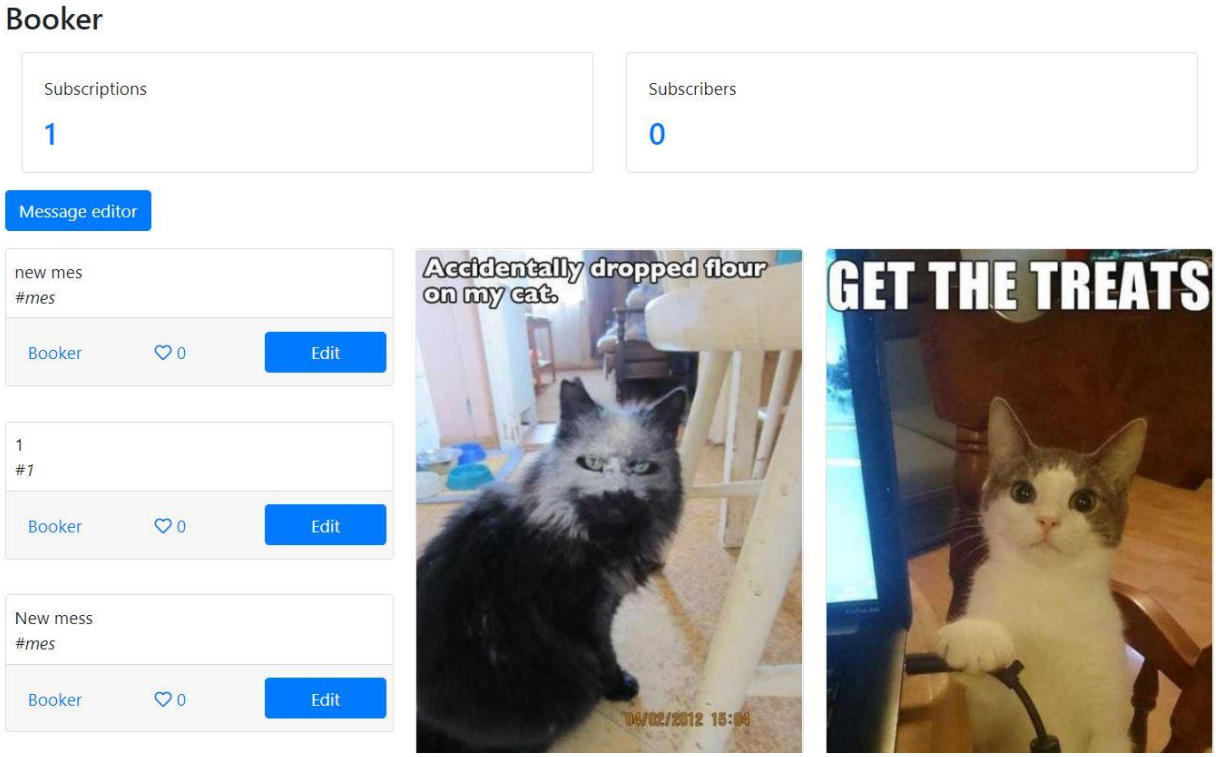


Рис. 12: Окно сообщений пользователя

Помимо этого в окне сообщений пользователя отображаются подписки и подписчики (рис. 12). Если нажать на любое элемент из этих двух, то в списке отобразятся все пользователи, на которых была осуществлена подписка (или же которые подписались на аккаунт). При нажатии на любого из них, происходит переход на страницу другого пользователя.

Для реализации механизма подписок и подписчиков пришлось создать несколько дополнительных таблиц: для подписок и подписчиков. Логика подписок реализована в классе UserService следующим образом: в качестве параметров передаются два элемента: текущий пользователь и тот, на которого подписались. Текущему пользователю в подписки необходимо добавить второго пользователя, а другому – текущего в подписчиков. Реализовать такое добавление с помощью Spring Data Jpa нетрудно: так как у каждого пользователя есть список подписок и подписчиков, то необходимо просто добавить в нужный нового пользователя.

Заключение

В ходе данной работы были достигнуты следующие результаты:

- разработано веб-приложение в формате блога;
- добавлена база данных;
- использованы технологии Spring Framework при разработке;

Таким образом, в результате данной работы было создано веб-приложение на Java с использованием технологий Spring Framework, позволяющее добавлять и просматривать посты. Код проекта доступен по ссылке[17].

[17] https://github.com/daniil-steperev/Blog

Список литературы

[1] What is Apache FreeMarker? [Электронный ресурс]. — Режим доступа: https://freemarker.apache.org/.

[2] Build fast, responsive sites with Bootstrap [Электронный ресурс]. — Режим доступа: https://getbootstrap.com/.

[3] The new way to stop bots [Электронный ресурс]. — Режим доступа: https://www.google.com/recaptcha/intro/v3.html.

[4] Remember-Me Authentication [Электронный ресурс]. — Режим доступа: https://docs.spring.io/spring-security/site/docs/3.0.x/reference/remember-me.html.