Министерство образования Республики Беларусь

Учреждение образования  
 БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
 ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Рефакторинг и оптимизация программного кода

Отчет

по практической работе № 4

на тему:

**МЕТОДЫ РЕФАКТОРИНГА ПРОГРАММНОГО КОДА**

|  |  |  |  |
| --- | --- | --- | --- |
| Проверил | |  | А.В. Шелест |
|  | (подпись) | |  |
| зачтено |  | |  |
|  | (дата защиты) | |  |
|  |  | |  |
| Выполнил |  | | О.Д. Семерник  гр. 114302 |
|  | (подпись) | |  |

Минск, 2024

**Содержание**

[1 Ссылка на github 3](#_Toc184052873)

[2 Авторизация и аутентификация пользователей ПС 4](#_Toc184052874)

# **1 ССЫЛКА НА GITHUB**

<https://github.com/oliaSD/TextProof.git>

# **2 АВТОРИЗАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ ПС**

При разработке системы аутентификации и авторизации в приложениях одной из ключевых задач является создание надежного и гибкого механизма управления доступом, который эффективно защищает чувствительные данные и функции. Специфика предметной области требует, чтобы система могла быстро и точно идентифицировать пользователей, проверяя их полномочия и обеспечивая доступ только к разрешенным ресурсам. Это создает дополнительные требования к архитектуре системы, поскольку она должна быть способна легко обрабатывать и проверять учетные данные, сопоставляя их с уже имеющимися записями.

Проектирование системы управления доступом требует не только тщательной проверки идентификационных данных, но и учета множества технических и функциональных требований. Система должна поддерживать высокую скорость обработки запросов, обеспечивать масштабируемость и безопасность данных, а также поддерживать различные методы аутентификации и уровни доступа для предоставления детальных отчетов и статистики. Помимо основного ядра системы авторизации, механизм должен интегрироваться с другими системами безопасности, такими как *LDAP, OAuth* или *SAML*, для поддержки внешней аутентификации и управления доступом.

Архитектура системы аутентификации также должна учитывать возможность последующего расширения функциональности и включения новых методов авторизации, таких как многофакторная аутентификация или адаптивная авторизация. Разработанная структура должна позволять пользователям легко взаимодействовать с системой, управлять своими учетными записями и разрешениями, отслеживать активность и получать уведомления о попытках входа.

Для реализации авторизации и аутентификации были использованы *Spring Security*, входящий в состав *Spring Framework* и *JWT*.

*Spring Security* – это мощный и настраиваемый фреймворк для аутентификации и авторизации в приложениях на базе *Java*.

*Spring Security* предоставляет всестороннюю поддержку аутентификации и авторизации как на уровне *HTTP*, так и на уровне метода. Можно настроить, какие страницы требуют аутентификации, и какие роли требуются для доступа к различным ресурсам. *Spring Security* может интегрироваться с различными сервисами аутентификации и авторизации, такими как *LDAP, OAuth2, OpenID Connect*, а также с более традиционными методами, как *HTTP Basic* и *Form-based* аутентификация.

*Spring Security* обладает гибкостью и расширяемость, возможно расширение его функциональности через собственные фильтры, провайдеры аутентификации и другие компоненты. Так же *Spring Security* предоставляет возможности для управления сессиями, в том числе одноразовые сессии и контроль за счёт использования *Cookie* и других подходов. Можно настроить политику для обработки ошибок и перенаправления после успешной или неуспешной аутентификации.

*Spring Security* предоставляет надежную основу для защищенных приложений, позволяя легко создавать приложения с высокими требованиями безопасности без необходимости реализовывать сложную логику вручную.

На рисунке 2.1 изображена использованная конфигурация конфигурация:



Рисунок 2.1 – Использованная конфигурация

*JWT (JSON Web Token*) – это компактный, *URL*-безопасный способ предавать утверждения между сторонами как объект *JSON*. Эти утверждения можно проверить и доверять им благодаря применению цифровой подписи. *JWT* используется в контексте обмена информацией, особенно для аутентификации и передачи проверенных данных.

*JWT* состоит из трех частей, закодированных в *Base64* и разделенных точками:

– *Header* (заголовок): обычно содержит тип токена (*JWT*) и алгоритм подписи, например, *HMAC* или *RSA*.

– *Payload* (полезная нагрузка): содержит утверждения. Утверждения могут быть предопределенными (как «*iss*» (*issuer*, издатель), «*exp*» (*expiration*, истечение) и «*sub*» (*subject*, субъект)) или пользовательскими.

– *Signature* (подпись): подписывается основной частью токена (*Header* и *Payload*) с использованием секретного ключа или приватного ключа в случае асимметричного шифрования.

На рисунке 2.2 изображен пример *JWT*.

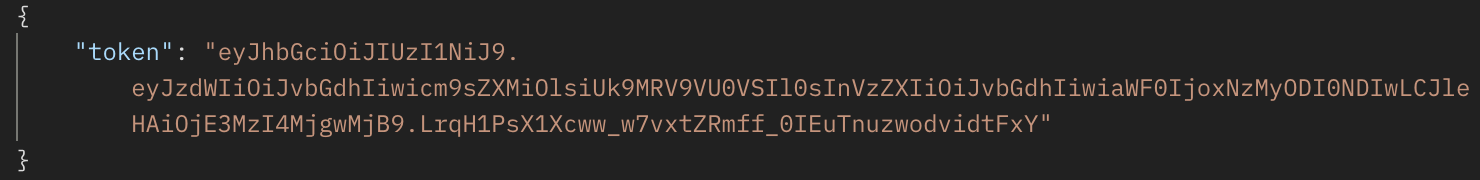


Рисунок 2.2 – Пример *JWT*

*JWT* используется для передачи информации о пользователе между клиентом и сервером. Например, после успешной аутентификации сервер генерирует *JWT* и отправляет его клиенту, который затем присылает его в заголовке *Authorization* для последующих запросов. Благодаря подписи получатель может проверить, что содержимое токена не было изменено.

За счёт формата *JWT* очень компактны, что делает их идеальными для передачи через *URL, HTTP*-заголовки или в теле *HTTP POST* запросов. *JWT* является самодостаточной сущностью, в которой можно хранить все необходимые утверждения для того, чтобы не обращаться к базе данных повторно.

Для обеспечения безопасности *JWT* подписаны, чтобы гарантировать, что данные, хранящиеся внутри токена, не были изменены. Секретный или приватный ключ используется для подписи токена, а в случае использования публичного ключа – для его проверки.

*JWT* может содержать утверждение «*exp»* (*expiration*), указывающее на временной промежуток, в течение которого токен считается действительным.

*JWT* активно используется в современных веб-приложениях, особенно в архитектурах микросервисов и *RESTful API*, где требуются легковесные и быстрые решения для аутентификации и передачи данных.

Пример кода создания *JWT* изображен на рисунке 2.3.

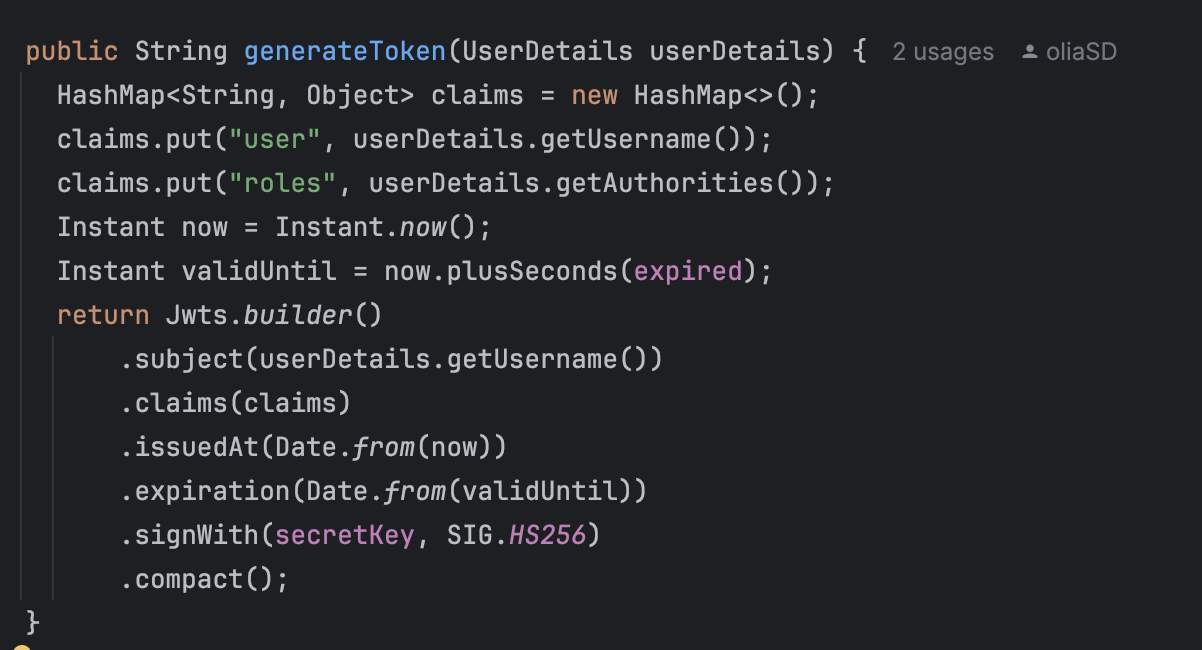


Рисунок 2.3 – Создание *JWT*

Одна из главных целей использования хеширования паролей – это защита паролей пользователей от утечек и несанкционированного доступа. Вместо хранения паролей в открытом виде, они хранятся в зашифрованном (хэшированном) виде.

*BCryptPasswordEncoder* – это компонент из *Spring Security*, который используется для хеширования паролей. Он реализует алгоритм *BCrypt*, который считается одним из наиболее безопасных и надежных для шифрования паролей. *BCrypt* автоматически генерирует уникальную «соль» (*random salt*) для каждого пароля, которую добавляет к нему перед хешированием. Это предотвращает атаки с использованием заранее вычисленных таблиц хешей (*rainbow tables*).

*BCrypt* позволяет настраивать так называемый «работоспособный фактор» (или «*cost factor*»), который определяет, сколько времени требуется для хеширования пароля. С увеличением значения этого фактора хеширование занимает больше времени и ресурсов, что затрудняет выполнение атак грубой силы (*brute-force attacks*). Поскольку «работоспособный фактор» можно изменять, *BCrypt* может адаптироваться к увеличению мощности аппаратного обеспечения. Это означает, что даже если компьютеры становятся быстрее, можно увеличить фактор сложности, чтобы поддерживать безопасность на высоком уровне.

*BCryptPasswordEncoder* предоставляет разработчикам простой и безопасный способ хеширования паролей, снижая риски, связанные с хранением и обработкой чувствительных данных. Это важная часть общей стратегии обеспечения безопасности приложений.

На рисунке 2.4 показана конфигурация *BCryptPasswordEncoder*.

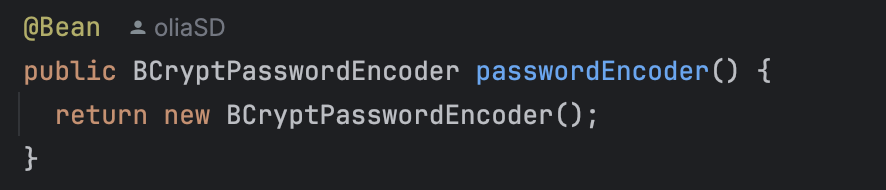


Рисунок 2.4 – Конфигурация *BCryptPasswordEncoder*

Таким образом, система аутентификации и авторизации должна обеспечивать надежный и гибкий механизм управления доступом, удовлетворяя требованиям безопасности, масштабируемости и интеграции. Для этого используется сочетание современных технологий, таких как *Spring Security* и *JWT*. *Spring Security* предоставляет широкие возможности настройки аутентификации и авторизации, интеграцию с различными системами (*LDAP*, *OAuth2*, *SAML*) и поддержку современных методов, включая многофакторную аутентификацию. *JWT* позволяет компактно и безопасно передавать данные аутентификации между клиентом и сервером, минимизируя зависимость от базы данных.

*Spring Security* обеспечивает гибкость и расширяемость, что позволяет адаптировать систему под конкретные задачи. Это включает управление сессиями, настройку политик обработки ошибок и возможность создавать собственные провайдеры аутентификации. *JWT*, благодаря своему формату и поддержке цифровой подписи, обеспечивает быструю проверку данных и минимизирует риски их изменения. Его структура из заголовка, полезной нагрузки и подписи делает токены удобными для использования в современных архитектурах, таких как микросервисы и *RESTful API*.

Для обеспечения безопасности хранения паролей применяется *BCryptPasswordEncoder* из *Spring Security*, который использует алгоритм *BCrypt*. Этот алгоритм включает автоматическую генерацию соли и настройку сложности, что защищает от атак грубой силы и делает систему устойчивой к повышению вычислительных мощностей. Таким образом, использование современных инструментов и подходов обеспечивает высокую надежность и удобство работы с системой аутентификации и авторизации, удовлетворяя требованиям безопасности и гибкости.