DOI: 10.58168/PPSPM2025_280-286
УДК 004.056.5

АРХИТЕКТУРА ЗАЩИЩЕННОГО ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ УПРАВЛЕНИЯ ПРОГРАММНЫМИ ПРОЕКТАМИ

Д.А. Малышев, С.А. Евдокимова

ФГБОУ ВО «Воронежский государственный университет инженерных технологий»

В работе рассматривается разработка веб-приложения для управления программными проектами, которое предназначено для планирования, назначения и отслеживания задач, контроля выполнения этапов, разграничения прав доступа, организацию командной работы и оперативного оповещения участников. Особое внимание уделено вопросам защиты приложения от киберугроз.

Ключевые слова: управление проектами, веб-приложение, информационная безопасность, архитектура, угрозы безопасности.

ARCHITECTURE OF A SECURE WEB APPLICATION FOR MANAGING SOFTWARE PROJECTS

D.A. Malyshev, S.A. Evdokimova

Voronezh State University of Engineering Technologies

The paper considers the development of a web application for managing software projects, which is designed for planning, assigning and tracking tasks, monitoring the completion of stages, delimiting access rights, organizing teamwork and promptly notifying participants. Special attention is paid to the issues of protecting the application from cyber threats.

Keywords: project management, web application, information security, architecture, security threats.

Одним из важнейших направлений цифровой трансформации является автоматизация управления программными проектами, включая планирование,

распределение задач, контроль исполнения и взаимодействие между членами команды. Особенно остро этот вопрос стоит в удаленных и распределенных рабочих средах, где качество цифрового инструмента напрямую влияет на продуктивность и координацию работы всех участников проекта [1-3].

Проектируемое веб-приложение ориентировано на использование в условиях корпоративной среды, что предопределяет особые требования к архитектуре – как в части надёжности и безопасности, так и в части масштабируемости и сопровождения. Основная цель проектирования архитектуры в данном случае заключается в построении стабильного, гибкого и защищённого решения, которое может быть развёрнуто в изолированной серверной среде с возможностью контролируемого расширения функциональности. Разрабатываемая программная система предназначена для небольших и средних команд разработки, работающих над одним или несколькими проектами в рамках общей корпоративной среды.

В качестве основного архитектурного подхода выбрана модель модульного монолита с изолированной логикой и строгой иерархией слоёв. Такой выбор обусловлен необходимостью сохранить централизованное управление бизнес-логикой, что особенно важно при ограниченном бюджете и в условиях эксплуатации внутри одной организации. В то же время структура приложения закладывает потенциальную возможность миграции в микросервисную архитектуру за счёт контейнеризации, изолированных модулей и унифицированных интерфейсов взаимодействия.

Архитектура системы охватывает три основные области: клиентское приложение (frontend), серверное ядро (backend), а также инфраструктурную и системную часть, включающую в себя серверные компоненты, систему хранения данных, кеширование и маршрутизацию (рисунок 1).

Клиентская часть системы реализована на базе современного JavaScript-фреймворка Vue 3 с использованием UI-библиотеки Quasar. Такой выбор обусловлен высокой степенью адаптируемости фреймворка, его производительностью, а также возможностью легко реализовывать компоненты пользовательского интерфейса, соответствующие требованиям корпоративных приложений. Система управления состоянием компонента построена на Composition API. Для взаимодействия с серверной частью используется GraphQL-клиент Apollo, обеспечивающий не только запросы и мутации, но и подписки на события в реальном времени. Важным аспектом является то, что авторизация реализована на основе JWT-токенов, передаваемых через httpOnly

cookie, что повышает защищённость от атак XSS и делает фронтенд полностью изолированным от хранения критичных данных [4].

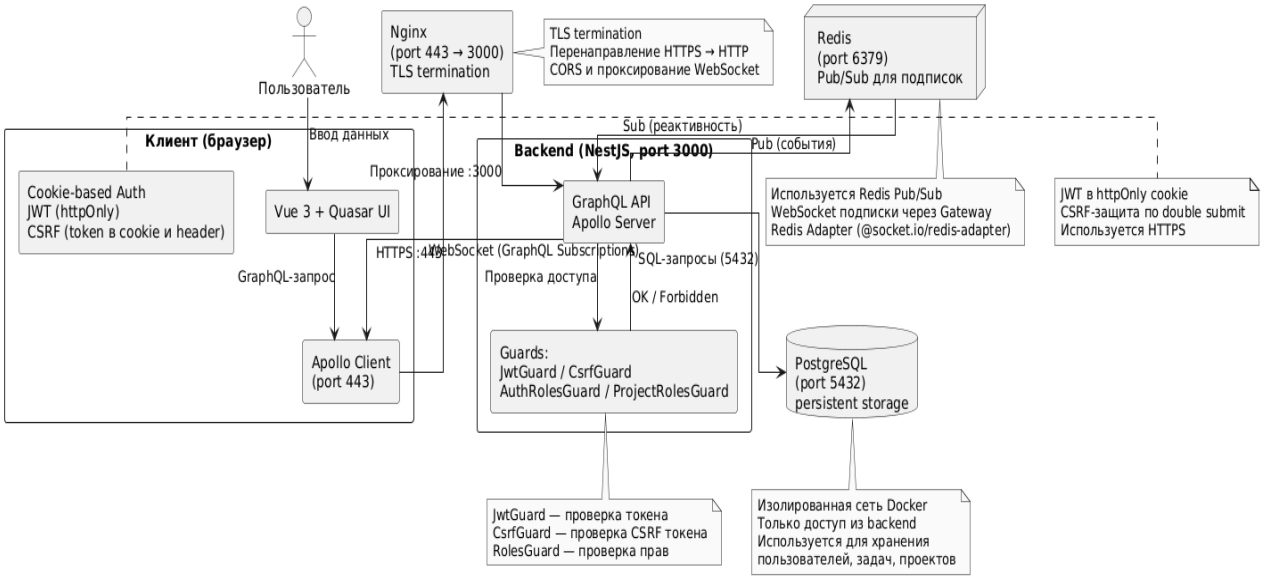

Рисунок 1 – Логическая архитектурная диаграмма

Серверная часть построена на платформе NestJS и реализует архитектурные принципы чистой архитектуры и инверсии зависимостей. NestJS, как прогрессивный фреймворк, предоставляет развитую систему модулей, которая идеально подходит для реализации масштабируемого бизнес-приложения. Основная логика разнесена по модулям: пользователи, проекты, задачи, авторизация и роли, каждый из которых изолирован и может быть независимо протестирован и доработан. Сервер предоставляет единую точку входа для всех клиентов через GraphQL API. Это решение выбрано ввиду его высокой выразительности, строгой типизации и возможности гибкой агрегации данных в одном запросе [5].

Каждый запрос, отправляемый на сервер, проходит через цепочку middleware и Guard-компонентов, которые обеспечивают многоуровневую защиту. Авторизация реализована через JwtGuard, верифицирующий токен, а доступ к отдельным операциям регулируется AuthRolesGuard для глобальных ролей и ProjectRolesGuard для контекста конкретного проекта. Такой подход позволяет гибко разграничивать доступ не только между обычными пользователями и администраторами, но и между участниками проектов с разными ролями: владельцем, администратором или участником.

Особенностью архитектуры является наличие дополнительного уровня защиты от CSRF-атак на уровне всех операций изменения данных (mutation), что реализовано через CsrfGuard, валидирующий токен из cookie и из заголовков.

Слой хранения данных построен на реляционной СУБД PostgreSQL. Структура базы данных тщательно спроектирована в соответствии с требованиями нормализации и учётом бизнес-логики приложения. Использование ORM-библиотеки TypeORM позволило добиться высокой декларативности моделей и упростить работу с вложенными сущностями и связями. Все связи между пользователями, проектами, задачами и ролями отражаются на уровне модели, что позволяет динамически строить сложные выборки с учётом ролевого контекста и прав доступа. Также реализована система миграций, обеспечивающая версионирование схемы базы данных и возможность автоматического отката изменений при необходимости [6, 7].

Подсистема реактивности реализована на основе Redis, выступающего в роли брокера сообщений для подписок в GraphQL. Это даёт возможность клиентам в режиме реального времени получать уведомления о событиях, происходящих в рамках их проектов: создании новых задач, изменениях статуса, назначении участников. Redis выбран как отказоустойчивое и производительное решение, которое легко масштабируется и хорошо интегрируется с Node.js-экосистемой. Для корректной работы подписок используется адаптер @socket.io/redis-adapter, обеспечивающий согласованность состояния между несколькими экземплярами серверов.

Инфраструктурный слой реализован с применением инструментов контейнеризации. Каждое логическое приложение (frontend, backend, Redis, PostgreSQL, Nginx) развернуто в отдельном Docker-контейнере, что позволяет изолировать среду выполнения, ускорить деплой и обеспечить воспроизводимость. Docker-контейнеры управляются с использованием docker-compose, а весь процесс CI/CD автоматизирован через GitLab, включая этапы сборки, тестирования, публикации образов и выкладки на сервер. Это исключает ручной фактор и позволяет гарантировать идентичность среды разработки и продукта.

Дополнительную защиту и маршрутизацию выполняет Nginx, который функционирует как обратный прокси, маршрутизирует HTTPS-запросы, обрабатывает WebSocket-подключения, а также реализует SSL-терминацию. Его конфигурация позволяет отсекать потенциальные атаки уже на периметре и минимизирует экспозицию backend-приложения во внешнюю сеть.

Таким образом, логическая структура приложения формирует собой стройную вертикаль взаимодействующих компонентов, где каждый уровень строго изолирован, но интегрирован в общую архитектуру. В верхнем уровне находится пользователь, взаимодействующий с клиентом; далее идёт слой API, защищённый авторизацией и ролевыми фильтрами; затем бизнес-логика и слой доступа к данным; и наконец, инфраструктурные элементы, обеспечивающие маршрутизацию, хранение и реактивную коммуникацию. Такое решение, построенное на современных технологиях и подходах, позволяет не только удовлетворить текущие функциональные и нефункциональные требования, но и формирует задел для долгосрочного развития приложения в корпоративной среде.

Безопасность веб-приложения, особенно внедряемого в корпоративную информационную среду, не может рассматриваться как вспомогательная функция или этап “после реализации”. Напротив, в условиях постоянных угроз, компрометаций данных, социальной инженерии и атак на уровне протоколов, архитектура защиты должна быть интегрирована в каждую составляющую системы — как на уровне кода, так и на уровне инфраструктуры, каналов связи и логики взаимодействия пользователей.

Проектируемая система реализует комплексную, сквозную модель безопасности, охватывающую все аспекты: идентификацию, аутентификацию, авторизацию, защиту каналов передачи данных, изоляцию компонентов, валидацию пользовательского ввода, защиту от наиболее распространённых атак и поддержку контролируемого окружения. Все решения принимались с опорой на актуальные рекомендации OWASP (Open Web Application Security Project) [4], а также опыт разработки корпоративных систем. В дополнение к требованиям стандартов ФСТЭК, реализация модулей безопасности ориентировалась на стандарты безопасной разработки ПО, в частности, OWASP ASVS и ISO/IEC 27034, регламентирующий проектирование безопасных приложений.

Ключевым элементом архитектуры безопасности является правильно спроектированная и реализованная система аутентификации. В данном приложении используется cookie-based подход с JWT (JSON Web Token), при котором токен авторизации формируется на стороне сервера и возвращается клиенту в виде httpOnly cookie. Это означает, что токен недоступен из JavaScript-кода на клиенте и, следовательно, не может быть считан через XSS-атаки.

На стороне сервера используется Guard-компонент JwtGuard, который валидирует подпись токена, проверяет срок действия и извлекает из токена полезную нагрузку (payload), содержащую идентификатор пользователя, email и его глобальные роли. Все эти данные передаются далее в контекст запроса (req.user) и используются как единая точка авторизации на всём протяжении обработки запроса.

Один из часто игнорируемых, но критичных в корпоративной среде векторов атак – это межсайтовая подделка запросов (CSRF). В условиях использования cookie для хранения авторизационного токена становится необходимым реализовать дополнительный механизм проверки «на подлинность намерения» клиента.

В данном приложении для всех GraphQL-операций типа mutation реализован Guard CsrfGuard, валидирующий наличие CSRF-токена. Принцип работы следующий: при первом посещении клиент получает CSRF-токен в виде отдельной cookie (не httpOnly), а затем должен передавать этот токен в HTTP-заголовке x-csrf-token при выполнении мутаций. Сервер проверяет, что значение заголовка совпадает со значением в cookie, и только после этого допускает выполнение мутации. Такой двойной контроль гарантирует, что вредоносный скрипт, работающий в другом домене, не сможет инициировать изменение данных от имени пользователя.

Данный подход сочетается с наличием httpOnly JWT и позволяет выстроить защищённую схему передачи данных без необходимости хранения токенов в браузере в виде переменных [8].

Таким образом, была спроектирована архитектура защищенного веб-приложения для управления программными проектами, особое внимание в работе было уделено техническому качеству архитектуры, защите пользовательских данных, разграничению прав доступа и соблюдению современных стандартов безопасности.

Список литературы

1. Задача о назначениях при управлении проектами / Ю.В. Бугаев [и др.] // Моделирование систем и процессов. – 2023. – Т. 16, № 4. – С. 15-23.

2. Выбор критерия оптимальности при принятии управленческих решений в сложных технических системах / А.В. Скрыпников [и др.] // Моделирование систем и процессов. – 2024. – Т. 17, № 1. – С. 120-128.

3. Высоцкая, И.А. Обоснование информационно-интеллектуальной поддержки принципов действия технических систем / И.А. Высоцкая // Моделирование систем и процессов. – 2024. – Т. 17, № 1. – С. 19-26.

4. OWASP Foundation. OWASP Top 10:2021. – URL: https://owasp.org/www-project-top-ten/(дата обращения: 12.10.2025).

5. ГОСТ ISO/IEC 29100-2021. Информационные технологии. Методы и средства обеспечения безопасности. Основы защиты персональных данных. – М.: Стандартинформ, 2021. – 28 с.

6. Smith, J. PostgreSQL Performance Tuning and Optimization / J. Smith, A. Jones. – London: O'Reilly, 2021. – 250 p.

7. Advanced PostgreSQL Optimization / D. Black [et al.]. – London: TechPress, 2021. – 400 p.

8. Thompson, M. JWT Authentication: Best practices and when to use it / M. Thompson. – LogRocket Blog, 2023. – 10 p.

References

1. The assignment problem in project management / Yu.V. Bugaev [et al.] // Modeling of systems and processes. – 2023. – Vol. 16, No. 4. – Pp. 15-23.

2. Selection of the optimality criterion in making managerial decisions in complex technical systems / A.V. Skrypnikov [et al.] // Modeling of systems and processes. – 2024. – Vol. 17, No. 1. – Pp. 120-128.

3. Vysotskaya, I.A. Substantiation of information and intellectual support for the principles of operation of technical systems / I.A. Vysotskaya // Modeling of systems and processes. - 2024. – Vol. 17, No. 1. – Pp. 19-26.

4. The OWASP Foundation. OWASP Top 10:2021. – URL: https://owasp.org/www-project-top-ten /(date of publication: 12.10.2025).

5. GOST ISO/IEC 29100-2021. Information technology. Methods and means of ensuring security. Fundamentals of personal data protection. – Moscow: Standartinform, 2021. – 28 p.

6. Smith, J. PostgreSQL performance tuning and optimization / J. Smith, A. Jones. – London: O'Reilly, 2021. – 250 p .

7. Advanced optimization of PostgreSQL / D. Black [et al.]. – London: TechPress, 2021. – 400 p.

8. Thompson, M. JWT authentication: Best practices and when to use it / M. Thompson. – LogRocket blog, 2023. – 10 p.