DOI: 10.18137/RNU.V9187.23.03.P.194 УДК 004.658

В.В. Пивоваров, Р.М. Хабибуллин, Р.Р. Нуркаев

# BFF – ПОДХОД К РАЗРАБОТКЕ МОБИЛЬНЫХ И ВЕБ-ПРИЛОЖЕНИЙ, ОПТИМИЗИРОВАННЫЙ ДЛЯ ПОЛЬЗОВАТЕЛЬСКОГО ОПЫТА

**Аннотация.** Статья посвящена изучению Backend for Frontend (BFF) – оптимизированного для пользовательского опыта подхода к разработке мобильных и веб-приложений. Обосновывается актуальность и значимость темы исследования. В настоящей статье рассматривается концепция BFF в контексте разработки мобильных и веб-приложений. BFF – это подход, который помогает оптимизировать пользовательский опыт путем создания специализированного серверного API, приспособленного к конкретным потребностям мобильного и веб-клиента. Объясняются преимущества BFF и его роль в современной разработке. Во-первых, BFF позволяет устранить избыточность запросов, которая часто возникает при использовании универсальных API для множества клиентских приложений. Во-вторых, BFF обеспечивает гибкость и независимость клиентской и серверной частей, позволяя каждой стороне эволюционировать независимо друг от друга. Рассматриваются различные подходы к реализации BFF и примеры его применения. Описываются основные задачи BFF, такие как агрегация данных с разных источников, адаптация данных для клиентского приложения и кэширование. Предлагаются лучшие практики и рекомендации по использованию BFF в различных сценариях разработки. В заключении подводятся итоги и делаются выводы о значимости BFF в создании мобильных и веб-приложений с улучшенным пользовательским опытом. BFF представляет собой мощный инструмент, который позволяет разработчикам эффективно управлять взаимодействием клиентской и серверной частей.

*Ключевые слова:* BFF, архитектура микросервисов, разработка фронтенда, проектирование API, интеграция с бэкендом, одностраничные приложения, сервисно-ориентированная архитектура, шлюз API, прокси-сервер, преобразование и агрегация данных, оркестрация запросов, обработка ошибок и ведение журнала.

V. Pivovarov, R. Khabibullin, R. Nurkaev

# BFF – A USER EXPERIENCE-OPTIMIZED APPROACH TO MOBILE AND WEB DEVELOPMENT

**Abstract.** The article is devoted to the study of Backend for Frontend (BFF) – a user experience-optimized approach to developing mobile and web applications. The authors substantiate the relevance and significance of the research topic. This article discusses the concept of BFF in the context of mobile and web application development. This article discusses the concept of Backend for Frontend (BFF) in the context of developing mobile and web applications. BFF is an approach that helps optimize the user experience by creating a custom back-end API tailored to the specific needs of the mobile and web client. The authors explain in detail the benefits of BFF and its role in modern development. First, BFF eliminates the request redundancy that often occurs when using generic APIs for multiple client applications. Then, BFF provides the flexibility and independence of the client and server parts, allowing each side to evolve independently of each other. Various approaches to the implementation of BFF and examples of its application are also considered. The main tasks of BFF are described, such as aggregation of data from different sources, adaptation of data for a client application, and caching. Best practices and recommendations are offered for using BFF in various development scenarios. The conclusion summarizes and draws conclusions about the importance of BFF in creating mobile and web applications with improved user experience. BFF is a powerful tool that allows developers to effectively manage the interaction between client and server parts.

*Keywords:* BFF, microservices architecture, front-end development, API design, back-end integration, single page applications, service-oriented architecture, API gateway, proxy server, data transformation and aggregation, query orchestration, error handling and logging.

**Пивоваров Виталий Владиславович**
ведущий инженер-программист, Университет Иннополис, город Иннополис. Сфера научных интересов: информационные технологии. Автор трех опубликованных научных работ.
Электронный адрес: pivovarov.vitaliy.work@gmail.com

**Хабибуллин Ринат Мударисович**
специалист технического обслуживания, США
Электронный адрес: mail@rinat.pro

**Нуркаев Руставиль Рустамович**
ведущий инженер-программист, город Таллин, Эстония. Сфера научных интересов: информационные технологии. Автор трех опубликованных научных работ.
Электронный адрес: rustaviln@gmail.com

*Введение*

Backend for Frontend (BFF) – это архитектурный шаблон, получивший значительную популярность в сфере разработки мобильных и веб-приложений. В современном цифровом ландшафте, где фронтенд-технологии быстро развиваются, стала очевидной потребность в более специализированном бэкенде, который специально удовлетворяет требования каждого фронтенд-клиента. BFF решает эту проблему, вводя промежуточный уровень между интерфейсной и серверной системами.

По своей сути BFF действует как выделенная внутренняя служба, предназначенная для обслуживания определенного внешнего интерфейса или типа клиента. Вместо того чтобы иметь один монолитный бэкенд, обслуживающий несколько типов клиентов, BFF позволяет создавать специализированные бэкенд-сервисы, оптимизированные для уникальных потребностей каждого внешнего приложения [1]. Этот подход позволяет разработчикам внешнего интерфейса лучше контролировать данные и API, которые им требуются, а также способствует лучшему разделению задач между внешним и внутренним интерфейсами.

Одним из ключевых преимуществ BFF является возможность оптимизировать взаимодействие между интерфейсом и сервером, что приводит к повышению производительности и снижению нагрузки на сеть. Предоставляя API-интерфейсы для конкретных клиентов, BFF устраняет необходимость для внешнего интерфейса делать несколько запросов к различным внутренним службам для извлечения и обработки данных. Вместо этого BFF действует как шлюз, собирая данные из различных серверных систем и преобразовывая их в формат, оптимальный для внешнего интерфейса. Это не только улучшает общее взаимодействие с пользователем, но и упрощает процесс разработки внешнего интерфейса. Кроме того, BFF способствует масштабируемости и гибкости разработки приложений. С помощью BFF серверные службы можно проектировать и масштабировать независимо друг от друга в зависимости от требований каждого внешнего клиента. Это упрощает обслуживание и обновления, поскольку изменения, внесенные в определенный интерфейс, могут быть изолированы от соответствующей службы BFF, не затрагивая другие части системы. Кроме того, BFF позволяет командам внедрить архитектуру микросервисов, в которой каждый BFF можно рассматривать как отдельный сервис, что упрощает модульную разработку и развертывание.

BFF предлагает мощное решение для оптимизации взаимодействия между интерфейсом и сервером при разработке современных приложений. Предоставляя выделенный серверный уровень, адаптированный к конкретным потребностям каждого внешнего интерфейса, BFF повышает производительность, масштабируемость и гибкость, способствуя лучшему разделению задач. Поскольку цифровой ландшафт продолжает развиваться, BFF представляет собой ценный архитектурный шаблон, который позволяет разработчикам создавать эффективные и адаптируемые приложения.

*Цели исследования*

**Изучение принципов и концепции BFF**. Целью исследования может быть более глубокое понимание того, что представляет собой BFF, его основные принципы и концепции, и как они применяются в разработке приложений.

**Оценка преимуществ и недостатков BFF.** Исследование может быть направлено на выявление преимуществ и недостатков использования BFF в сравнении с другими подходами, такими как монолитные приложения или другие архитектурные стили.

**Исследование влияния BFF на разработку фронтенда**. Целью исследования может быть оценка того, как использование BFF влияет на разработку фронтенда, включая улучшение производительности, масштабируемости и удобства разработки.

**Исследование эффективности BFF в микросервисной архитектуре**. Исследование может быть направлено на оценку того, как BFF взаимодействует с микросервисной архитектурой, его роль в интеграции и координации сервисов, а также эффективность такого подхода в реальных сценариях.

**Исследование лучших практик и рекомендаций по использованию BFF.** Цель исследования – сбор и анализ лучших практик и рекомендаций по использованию BFF для предоставления разработчикам руководства по эффективному применению этого подхода.

*Что такое BFF*

BFF – это паттерн проектирования, который позволяет разработчикам создавать отдельный слой бэкенда для каждого фронтенда или клиентского приложения. Когда фронтенд-разработчики разрабатывают клиентское приложение, они часто используют один и тот же бэкенд для всех клиентов. Это может привести к проблемам с масштабированием и гибкостью, поскольку изменения на бэкенде могут затронуть все клиентские приложения. BFF решает эту проблему, создавая специализированный слой бэкенда для каждого клиентского приложения [9]. Это позволяет разработчикам более гибко управлять каждым клиентским приложением, легче масштабировать и поддерживать его.

*Преимущества использования BFF [3; 4; 8]*

**Гибкость.** Позволяет разработчикам создавать специализированный слой бэкенда для каждого клиентского приложения, что увеличивает гибкость и масштабируемость приложения.

**Быстродействие.** Может оптимизировать API для каждого клиента, что позволяет ускорить работу приложения.

**Безопасность.** Может использовать различные методы аутентификации и авторизации для каждого клиента, что увеличивает безопасность приложения.

**Разработка и тестирование.** Упрощает разработку и тестирование, поскольку каждый клиент имеет свой собственный слой бэкенда.

*Недостатки использования BFF [3; 4; 8]*

**Сложность.** Увеличивает сложность архитектуры приложения и может потребовать больше времени на разработку.

**Дополнительные затраты.** Создание дополнительного слоя бэкенда для каждого клиента может потребовать дополнительных затрат на разработку и поддержку.

**Управление.** Управление несколькими слоями бэкенда может быть сложным и требовать дополнительных ресурсов.

**Неполнота данных.** Данные могут быть неполными, поскольку каждый клиент имеет свой собственный слой бэкенда, что может приводить к различиям в информации между клиентскими приложениями.

**Зависимость от клиента.** Может зависеть от конкретного клиента, что может привести к проблемам при переносе приложения надругой клиент или при добавлении новых клиентов.

*Основные задачи BFF [1; 5]*

**Агрегация данных с разных источников.** BFF может собирать данные из различных микросервисов или API и агрегировать их для передачи в клиентское приложение. Это позволяет уменьшить количество запросов от клиента и оптимизировать производительность.

**Адаптация данных для клиентского приложения.** BFF может выполнять преобразование данных, чтобы соответствовать требованиям клиентского приложения. Например, он может преобразовывать данные в формат JSON, оптимизировать размеры изображений или адаптировать данные под специфические требования интерфейса.

**Кэширование**. BFF может кэшировать данные, чтобы уменьшить количество запросов к бэкенд-сервисам и улучшить отзывчивость приложения. Он может использовать различные стратегии кэширования, такие как кэширование полностью готовых ответов или фрагментов данных.

**Управление авторизацией и безопасностью.** BFF может обеспечивать слой авторизации и аутентификации для клиентского приложения. Он может проверять права доступа, обрабатывать токены аутентификации и применять политики безопасности, чтобы защитить доступ к данным.

**Оптимизация производительности.** BFF может выполнять различные оптимизации, такие как сокращение объема передаваемых данных, предварительная загрузка данных, кеширование запросов или предварительный расчет сложных вычислений, чтобы улучшить производительность клиентского приложения.

**Маршрутизация и управление запросами.** BFF может выступать в роли прокси-сервера, обрабатывая запросы от клиента и маршрутизируя их к соответствующим бэкенд-сервисам. Он может также управлять ошибками, логированием и контролировать поток запросов.

*Как можно использовать BFF в разработке*

Пример использования BFF на языке Java с использованием фреймворка Spring. Допустим, есть микросервисы для работы с продуктами и корзиной покупок. Нужно создать два клиентских приложения: одно для мобильных устройств, другое – для веб-приложений. Можно использовать BFF для создания отдельного слоя бэкенда для каждого приложения.

Для начала создадим класс ProductDTO, который будет содержать информацию о продукте:

```
publicclassProductDTO {
private Long id;
private String name;
private String description;
privateBigDecimal price;
// геттерыисеттеры
}
```

Затем создадим контроллеры для каждого микросервиса и BFF-контроллеры для каждого клиентского приложения. Например, Product Controller для микросервиса продуктов:

```
@RestController
@RequestMapping("/products")
publicclassProductController {
@Autowired
privateProductServiceproductService;

@GetMapping("/{id}")
publicProductDTOgetProduct(@PathVariable Long id) {
        Product product = productService.getProductById(id);
returnnewProductDTO(product.getId(), product.getName(), product.getDescription(),
product.getPrice());
    }
}
```

Теперь можно создать BFF-контроллеры для каждого клиентского приложения. Например, для мобильного приложения:

```
@RestController
@RequestMapping("/mobile")
publicclassMobileBFFController {
@Autowired
private ProductService productService;

@GetMapping("/products/{id}")
public ProductDTO getProduct(@PathVariable Long id) {
        Product product = productService.getProductById(id);
// добавляемдополнительнуюинформациюдлямобильногоприложения
returnnew ProductDTO(product.getId(), product.getName(), product.getDescription(),
product.getPrice().multiply(BigDecimal.valueOf(0.9)));
    }
}
```

Аналогично создаем BFF-контроллеры для веб-приложения:

```
@RestController
@RequestMapping("/web")
publicclassWebBFFController {
@Autowired
privateProductServiceproductService;

@GetMapping("/products/{id}")
publicProductDTOgetProduct(@PathVariableLongid) {
Productproduct = productService.getProductById(id);
// добавляем дополнительную информацию для веб-приложения
returnnewProductDTO(product.getId(), product.getName(), product.getDescription() + " (10%
скидка на следующую покупку)", product.getPrice());
    }
}
```

Таким образом, созданы два отдельных слоя бэкенда для мобильного и веб-приложений, каждый из которых может использовать свои собственные BFF-контроллеры для работы с микросервисами.

В целом JWT – это удобный и безопасный способ передачи информации между клиентом и сервером, особенно при работе с микросервисами и разделением на клиентскую и серверную части. Он обеспечивает безопасность и надежность передаваемых данных, а также гибкость в хранении любой информации. Однако стоит учитывать некоторые из его недостатков, включая возможность подделки токена при неудачной настройке безопасности, необходимость расшифрования информации из токена при каждом запросе и некоторую сложность в управлении его сроком действия [2]. В целом, использование JWT должно осуществляться с осторожностью и с учетом его преимуществ и недостатков в конкретной ситуации.

*Практики и рекомендации по использованию BFF*
*в различных сценариях разработки [6; 7; 10]*

**Разработка клиенто-специфичных API.** Создание BFF-сервисов, которые предоставляют API, специально адаптированные для нужд клиентского приложения. Это позволит упростить работу фронтенд-команды, обеспечивая необходимые данные и функциональность.

**Разделение бизнес-логики.** BFF-сервисы должны обрабатывать только те запросы, которые связаны с клиентскими интерфейсами. Бизнес-логика должна оставаться в бэкенд-сервисах. Такое разделение помогает поддерживать чистоту и ясность кода.

**Агрегация данных.** В сценариях, где клиентское приложение требует получения данных из различных источников, BFF может выполнять рольагрегатора данных. Он может обращаться к разным бэкенд-сервисам, собирать необходимую информацию и предоставлять ее клиентскому приложению.

**Кэширование и оптимизация производительности.** Использовать BFF для кэширования данных и оптимизации производительности. BFF может кэшировать данные, чтобы уменьшить нагрузку на бэкенд-сервисы и ускорить ответы на запросы.

**Обработка ошибок и логирование.** BFF должен корректно обрабатывать ошибки и предоставлять информативные сообщения для разработчиков и конечных пользователей. Логирование ошибок поможет в идентификации и устранении проблем.

**Масштабируемость и гибкость.** BFF-сервисы должны быть масштабируемыми и гибкими, чтобы легко адаптироваться к изменяющимся требованиям клиентского приложения. Используйте горизонтальное масштабирование и возможности автоматического масштабирования облака для обеспечения отзывчивости и надежности.

**Безопасность.** Обеспечить соответствие безопасности в BFF-сервисах. Применяйте необходимые меры аутентификации и авторизации, защищайте API от атак и уязвимостей, особенно при обработке клиентской информации.

**Тестирование.** Проводить тестирование BFF-сервисов, включая модульные, интеграционные и тесты производительности, чтобы обнаружить и исправить проблемы до выпуска в продакшн.

**Мониторинг.** Отслеживать производительность и доступность BFF-сервисов с помощью мониторинга. Это поможет быстро обнаружить и решить проблемы, а также оптимизировать производительность.

**Версионирование.** Если клиентское приложение и BFF-сервисы развиваются независимо друг от друга, необходимо обеспечить механизм версионирования API. Это поможет поддерживать совместимость между различными версиями клиентского приложения и BFF-сервисов.

**Документация.** Создать хорошую документацию для BFF-сервисов, чтобы разработчики фронтенда и бэкенда могли легко понять, как использовать и взаимодействовать с этими сервисами.

**Постоянная поддержка.** Обеспечить постоянную поддержку и обновление BFF-сервисов. Мониторить изменения в клиентском приложении и вносить необходимые изменения в BFF для поддержания его актуальности и эффективности.

*Заключение*

Таким образом, BFF является мощным инструментом в разработке современных приложений. Он позволяет улучшить архитектуру и оптимизировать работу между клиентскими приложениями и бэкенд-сервисами.

Использование BFF позволяет разработчикам создавать клиенто-специфичные API, разделять бизнес-логику, агрегировать данные из разных источников, кешировать информацию и обеспечивать безопасность.

Однако важно помнить о лучших практиках и рекомендациях при использовании BFF. Разделение ответственности, масштабируемость, безопасность, тестирование и мониторинг являются ключевыми аспектами успешного применения BFF.

В конечном итоге BFF упрощает разработку и поддержку клиентских приложений, улучшает производительность и удовлетворение пользователей. С его помощью команды фронтенда и бэкенда могут более эффективно сотрудничать, достигая лучших результатов в разработке и построении надежных и масштабируемых приложений. BFF продолжает развиваться и находить свое применение в различных сценариях разработки. Правильное использование этого концепта и следование лучшим практикам поможет создавать более гибкие, эффективные и надежные приложения для удовлетворения потребностей пользователей в современном мире.

Литература

1. *Ричардсон К.* Микросервисы. Паттерны разработки и рефакторинга; пер. с англ. С.В. Черникова; под ред. Н. Гринчика. СПб.: Питер, 2022. 544 с.

2. *Форд Н., Куа П., Парсонс Р.* Эволюционная архитектура. Поддержка непрерывных изменений; пер. с англ. А. Демьяникова; под ред. П. Ковалева. СПб.: Питер, 2019. 272 с.
3. *Эванс Э.* Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем; пер. с англ. В.Л. Бородовой. М.: Вильямс, 2018. 448 с.
4. *Bruce M., Pereira P.A.* (2018) Microservices in Action. N.Y.: Manning, 2018, 392 p.
5. *Burns B.* (2018) Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. CA: O'Reilly Media, 2018, 162 p.
6. *Hoffman K.* (2017) Building Microservices with ASP.NET Core: Develop, Test, and Deploy Cross-Platform Services in the Cloud. CA: O'Reilly Media, 2017, 299 p.
7. *Krause L.* (2015) Microservices: Patterns and Applications: Designing fine-grained services by applying patterns. N.Y.: Lucas Krause, 2015, 126 p.
8. *Newman S.* (2015) Building Microservices: Designing Fine-Grained Systems. Sebastopol, CA: O'Reilly Media, 2015, 280 p.
9. *Newman S.* Pattern: Backends for Frontends, 18.11.2015. Sam Newman & Assiciates. Available at: https://samnewman.io/patterns/architectural/bff/ (accessed: 30.06.2023).
10. *Vijayakumar T.* (2018) Practical API Architecture and Development with Azure and AWS: Design and Implementation of APIs for the Cloud. N.Y.: Apress, 2018, 188 p.

## Literature

1. Richardson K. (2022) *Mikroservisy. Patterny razrabotki i refaktoringa* [Microservices. Patterns of development and refactoring]. St. Petersburg, Piter Publishing, 2022, 544 p. (in Russian).
2. Ford N., Kua P., Parsons R. (2019) *Jevoljucionnaja arhitektura. Podderzhka nepreryvnyh izmenenij* [Evolutionary architecture. Support for continuous change]. St. Petersburg, Piter Publishing, 2019, 272 p. (in Russian).
3. Evans E. (2018) *Predmetno-orientirovannoe proektirovanie (DDD). Strukturizacija slozhnyh programmnyh system* [Domain-oriented design (DDD). Structuring of complex software systems]. Moscow, Williams Publishing, 2018, 448 p. (in Russian).
4. Bruce M., Pereira P.A. (2018) Microservices in Action. N.Y.: Manning, 2018, 392 p.
5. Burns B. (2018) Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. CA: O'Reilly Media, 2018, 162 p.
6. Hoffman K. (2017) Building Microservices with ASP.NET Core: Develop, Test, and Deploy Cross-Platform Services in the Cloud. Sebastopol, California, O'Reilly Media, 2017, 299 p.
7. Krause L. (2015) Microservices: Patterns and Applications: Designing fine-grained services by applying patterns. New York, Lucas Krause, 2015, 126 p.
8. Newman S. Pattern: Backends for Frontends. 11/18/2015. Sam Newman &Assiciates. Available at: https://samnewman.io/patterns/architectural/bff/ (accessed: 06.30.2023).
9. Newman S. (2015) Building Microservices: Designing Fine-Grained Systems. Sebastopol, California, O'Reilly Media, 2015, 280 p.
10. Vijayakumar T. (2018) Practical API Architecture and Development with Azure and AWS: Design and Implementation of APIs for the Cloud. New York, Apress, 2018, 188 p.