УДК 004.4

АРХИТЕКТУРА ПРОГРАММНОГО СРЕДСТВА УПРАВЛЕНИЯ ЗАЯВКАМИ И ИНЦИДЕНТАМИ С IT-ОБОРУДОВАНИЕМ КОМПАНИИ С ПРИМЕНЕНИЕМ ПРИНЦИПА DDD

Лесак П. С.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Научный руководитель: Лихачевский Д.В. – к. т. н., доцент, декан ФКП

Аннотация. Обосновано применение принципа *DDD* для построения архитектуры программного средства управления заявками и инцидентами с *IT*-оборудованием компании. Предложен вариант построения такой архитектуры. Выделены преимущества доменно-ориентированной архитектуры, включая адаптивность, масштабируемость, легкость внедрения нового функционала.

Ключевые слова: архитектура, домен, *DDD,* оборудование, моделирование, программное средство, микросервис, масштабирование.

Введение. В современных компаниях эффективное управление заявками и инцидентами, связанными с *IT*-оборудованием, играет ключевую роль в обеспечении стабильной работы информационных систем. Отсутствие централизованного подхода к обработке заявок, несогласованность процессов и недостаточная автоматизация могут привести к задержкам в обслуживании, росту эксплуатационных затрат и снижению продуктивности сотрудников.

Одним из подходов к созданию надежного и гибкого программного средства для управления такими процессами является использование принципов *Domain-Driven Design* (*DDD*). *DDD* позволяет моделировать программное средство на основе предметной области, что способствует лучшему пониманию бизнес-процессов и упрощает дальнейшее развитие программного обеспечения.

Основная часть. В отличие от традиционных подходов, где первостепенное внимание уделяется техническим аспектам, таким как базы данных или протоколы взаимодействия, *DDD* делает акцент на моделировании бизнес-процессов и концепций, характерных для конкретной области.

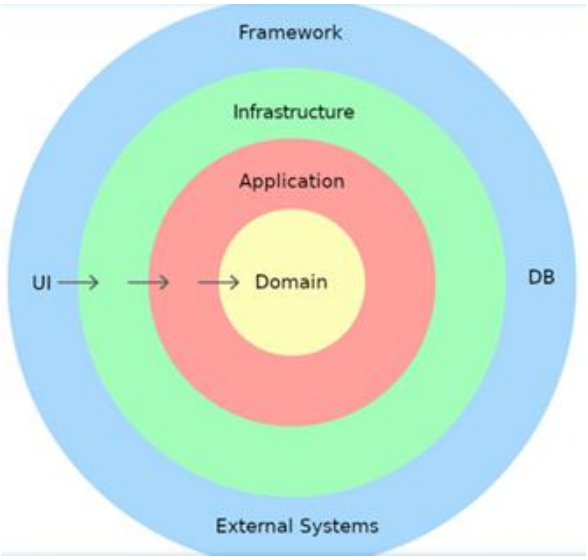

Рисунок 1 – Схема многослойной архитектуры

Основная идея заключается в построении программного средства вокруг доменной модели – набора объектов, сущностей и бизнес-правил, отражающих структуру и динамику предметной области. Для этого используется многослойная архитектура, изображенная на рисунке 1, где каждый уровень программного средства отвечает за свою зону ответственности: слой *API* обрабатывает внешние запросы, слой приложения управляет бизнес-процессами, слой домена содержит основную бизнес-логику, а слой инфраструктуры отвечает за взаимодействие с базами данных и внешними сервисами. Такой подход позволяет отделить бизнес-логику от технических деталей, делая программное средство гибкой, легко расширяемой и удобной для сопровождения. Для того чтобы уравнять разработчиков и экспертов предметной области, чтобы было гораздо проще обмениваться полезными знаниями о предметной области, подход *DDD* предлагает применять общий набор терминов, понятий и фраз, который будет использоваться в общении между членами команды, и который позже отразится в исходном коде результирующей программы.

Применение *DDD* в архитектуре проекта проявляется через несколько ключевых особенностей: разделение программного средства на предметные контексты, использование слоеной структуры сервисов, применение агрегатов для управления целостностью данных, асинхронное взаимодействие через брокер сообщений, а также изолированное хранение данных для каждой предметной области. Эти принципы позволяют построить гибкую, отказоустойчивую и легко масштабируемую программное средство.

Разделение программного средства на предметные контексты минимизирует зависимости между модулями и облегчает управление бизнес-логикой. В архитектуре проекта каждый микросервис представляет отдельный контекст, выполняя строго определенные функции. Сервис авторизации занимается исключительно управлением учетными записями пользователей, сервис заявок отвечает за обработку запросов, а сервис товаров управляет каталогом оборудования. Благодаря такому разделению каждый модуль остается независимым, что не только упрощает разработку и поддержку, но и повышает отказоустойчивость всей программного средства: сбой в одном сервисе не приводит к отказу всей платформы.

Каждый микросервис включает несколько уровней, это позволяет четко разграничить логику работы программного средства и технические детали, такие как взаимодействие с базой данных или брокером сообщений. Сервис заявок не просто сохраняет информацию о новых запросах, а применяет доменные правила, проверяя корректность данных и связывая заявки с пользователями через доменные объекты. Таким образом, архитектура проекта остается чистой и понятной, а изменения в одном слое не затрагивают другие.

Для обеспечения целостности данных в программном средстве активно используются агрегаты, они объединяют связанные сущности в единую структуру и позволяют гарантировать согласованность изменений. В сервисе комплектации товаров каждая заявка может включать несколько позиций, и программное средство должна отслеживать, чтобы все изменения в составе заказа происходили корректно. Агрегаты помогают предотвратить ошибки, связанные с несогласованными изменениями, и обеспечивают предсказуемое поведение программного средства, так как изменения в одном объекте автоматически распространяются на все связанные сущности.

Асинхронное взаимодействие между сервисами реализовано через брокер сообщений *ActiveMQ*. Вместо жестких зависимостей между сервисами используется событийная модель, позволяющая каждому компоненту работать автономно. При оформлении заявки информация сначала отправляется в сервис заявок, который публикует событие о новом заказе в *ActiveMQ*. Другие сервисы, такие как сервис комплектации товаров, подписываются на это событие и обрабатывают его независимо. Такой подход снижает

нагрузку на программное средство, исключает задержки при обработке запросов и повышает масштабируемость архитектуры.

В организации хранения данных, каждая предметная область имеет собственную базу или схему данных. Это исключает избыточные зависимости между сервисами и гарантирует целостность информации внутри каждого контекста. Сервис авторизации хранит данные о пользователях отдельно от сервиса товаров, что позволяет изменять модель пользователей без влияния на структуру каталога оборудования. Такой подход не только упрощает развертывание обновлений, но и снижает риск конфликтов при внесении изменений в программное средство.

Заключение. Архитектура проекта, построенная на основе принципов *DDD*, обеспечивает четкое разделение ответственности, модульность и гибкость программного средства. Использование доменных моделей, агрегатов и событийной архитектуры позволяет создать надежную и масштабируемую платформу для управления заявками и инцидентами с *IT*-оборудованием. Такой подход не только облегчает разработку и поддержку программного средства, но и делает её более устойчивой к изменяющимся требованиям бизнеса, позволяя легко адаптировать логику обработки заявок, добавлять новые модули и расширять функциональность без риска нарушить работу существующих компонентов.

Список литературы

1. Предметно-ориентированное проектирование (DDD) / Эванс, Э. // Структуризация сложных программных систем. – Изд-во Вильямс, 2015. – 448 с.

2. Learning Domain-Driven Design: Aligning Software Architecture and Business Strategy / V. Khononov // O'Reilly Media Inc, 1005 Gravenstein Highway North, 2021. – P. 3-17.

UDC 004.04

BUILDING THE ARCHITECTURE OF A SOFTWARE APPLICATION AND INCIDENT MANAGEMENT TOOL FOR THE COMPANY'S IT EQUIPMENT USING THE DDD PRINCIPLE

Lesak P.S.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Likhachevsky D.V. – Cand. of Sci., Associate Professor, Dean of the FCAD

Annotation. The application of the DDD principle to build the architecture of a software application and incident management tool with the company's IT equipment is substantiated. A variant of building such an architecture is proposed. The advantages of domain-oriented architecture are highlighted, including adaptability, scalability, and ease of implementation of new functionality.

Keywords: architecture, domain, DDD, hardware, modeling, software, microservice, scaling.