|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА** – **Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий (ИТ)  
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

|  |  |
| --- | --- |
| **ОТЧЕТ О ПРАКТИЧЕСКИМ РАБОТАМ** | |
| **по дисциплине** | |
| «Проектирование клиент-серверных систем»  **на тему**  «Информационная система Электронный журнал школьника» | |
|  | |
| Выполнил студент группы ИКБО-20-21  Принял | Сидоров С.Д.  Мельников Д.А. |

**Оглавление**

[1 Практическая работа №1 3](#_Toc179142133)

[2 Практическая работа №2 9](#_Toc179142134)

[3 Практическая работа №3 14](#_Toc179142135)

[4 Практическая работа №4 15](#_Toc179142136)

[5 Практическая работа №5 17](#_Toc179142137)

[6 Практическая работа №6 21](#_Toc179142138)

[7 Практическая работа №7 24](#_Toc179142139)

[8 Практическая работа №8 26](#_Toc179142140)

[9 Список литературы 31](#_Toc179142141)

# Практическая работа №1

**Цель работы:**

1 Знакомство с графической нотацией формализации и описания бизнес-процессов IDEF0. Знакомство c понятием функциональной модели AS-IS («как есть»).

2 Описание и построение функциональной модели AS-IS выбранной предметной области с применением нотации IDEF0.

**Постановка задачи:**

Для заданной предметной области разработать модель AS-IS. Вы можете выбрать один из вариантов процессов, описанных в приложении, или предложить свой вариант.

**Ход работы:**

Была спроектирована контекстная диаграмма А0 в нотации IDEF0

В качестве входа по управлению были выбраны:

* Правила хранения
* График работы
* Законодательство

В качестве входящих потоков были выбраны:

* Вещи посетителей
* Запрос клиента

В качестве механизмов используются:

* Сотрудники гардероба
* Гардеробные ряды
* Ячейки хранения

В качестве выходов после выполнения ИС получены:

* Хранимые вещи
* Выданные вещи
* Талоны

Сама контекстная диаграмма процесса ИС «Автоматизированный гардероб» представлена на рисунке 1.

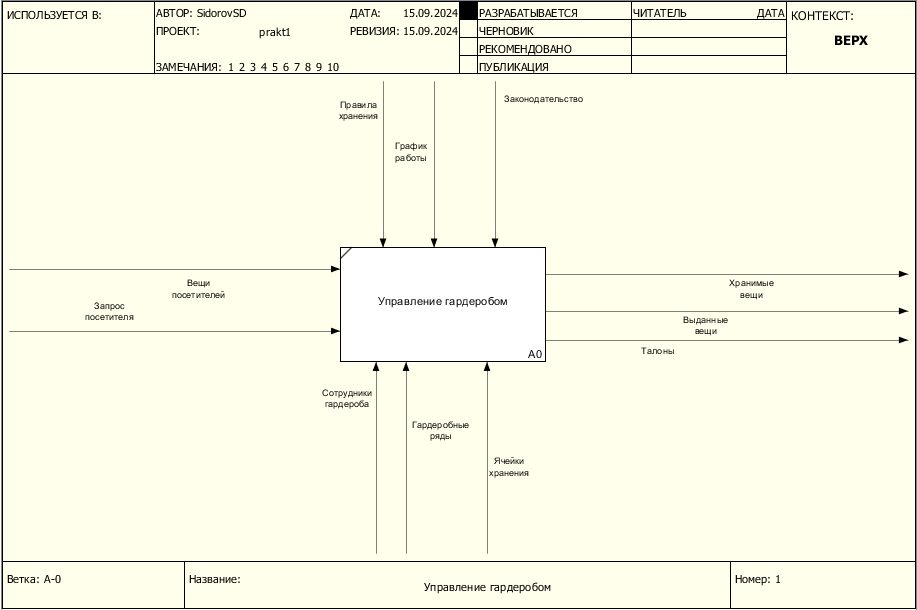


Рисунок 1 – Контекстная диаграмма А0

Далее была произведена декомпозиция основного функционального блока А0 (рисунок 2). Были получены следующие функциональные блоки:

* Обработка запроса посетителем – А1
* Передача вещей на хранение – А2
* Хранение вещей – А3
* Выдача вещей с хранения – А4

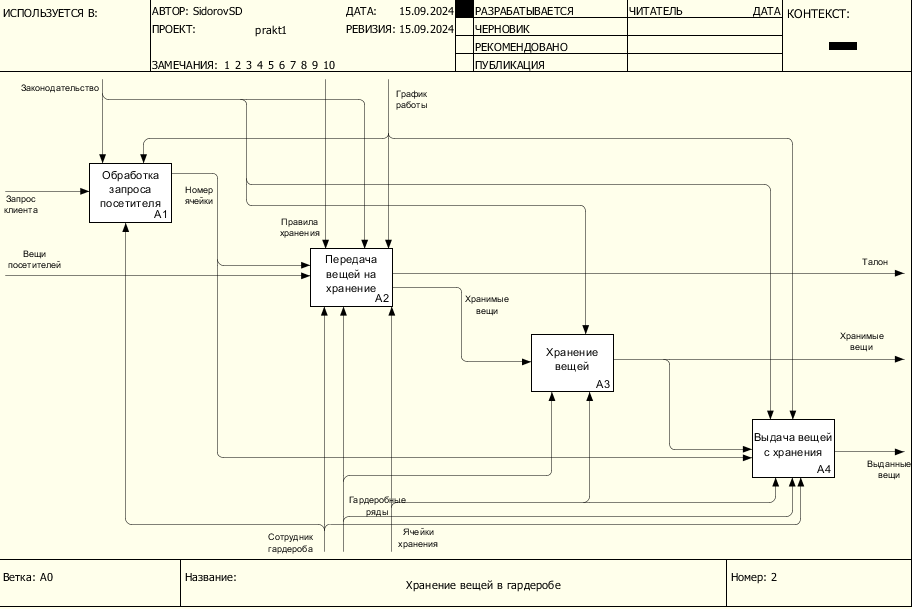


Рисунок 2 – Декомпозиция функционального блока

Декомпозиция блока A1:

* Получение данных клиента из запроса – А11
* Получение номера требуемой ячейки – А12

Диаграмма декомпозированного блока представлена на рисунке 3.

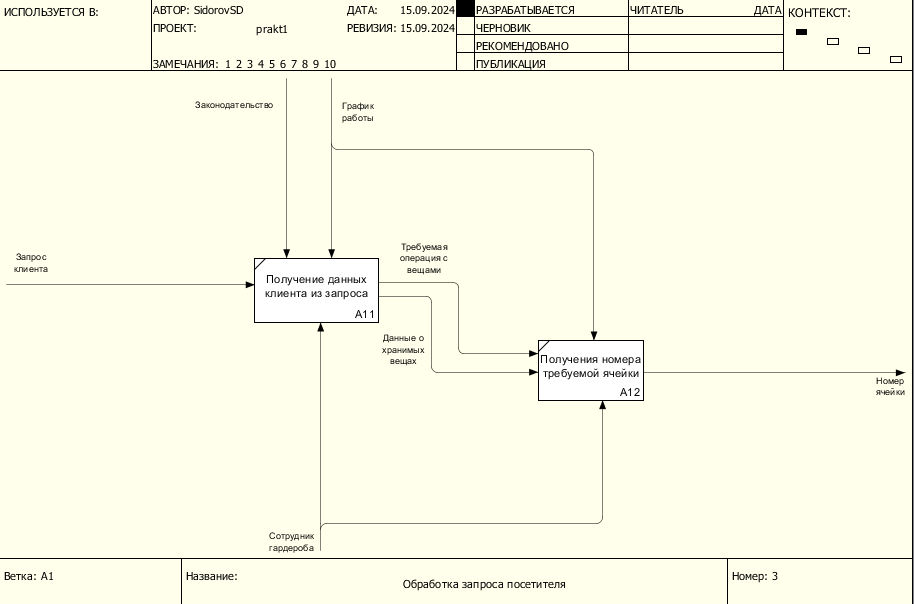


Рисунок 3 – Декомпозиция блока А1

Декомпозиция блока A2:

* Получение вещей – А21
* Перемещение вещей к месту хранения – А22

Диаграмма декомпозированного блока представлена на рисунке 4.

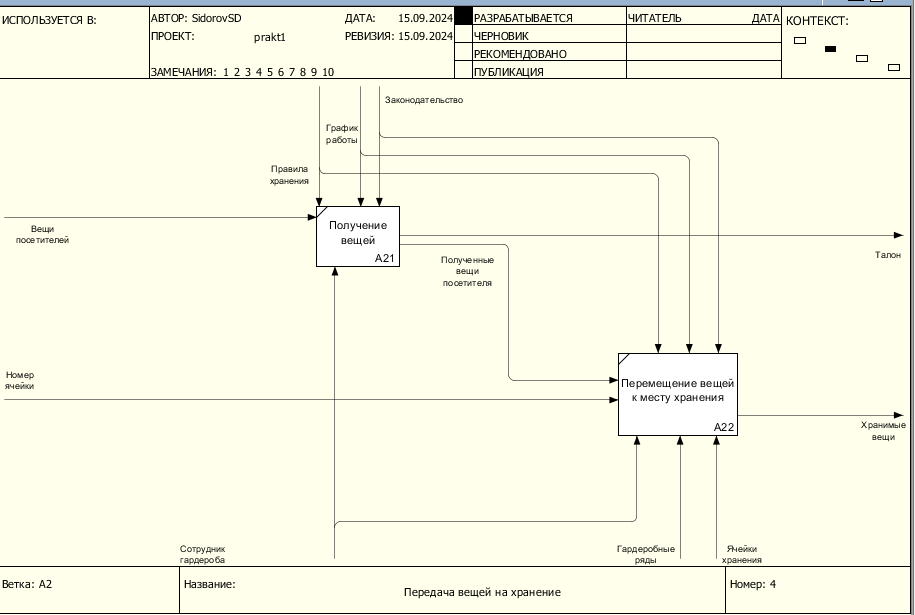


Рисунок 4 – Декомпозиция блока А2

Декомпозиция блока A3:

* Размещение вещей в ячейке – А31
* Хранение вещей – А32

Диаграмма декомпозированного блока представлена на рисунке 5.

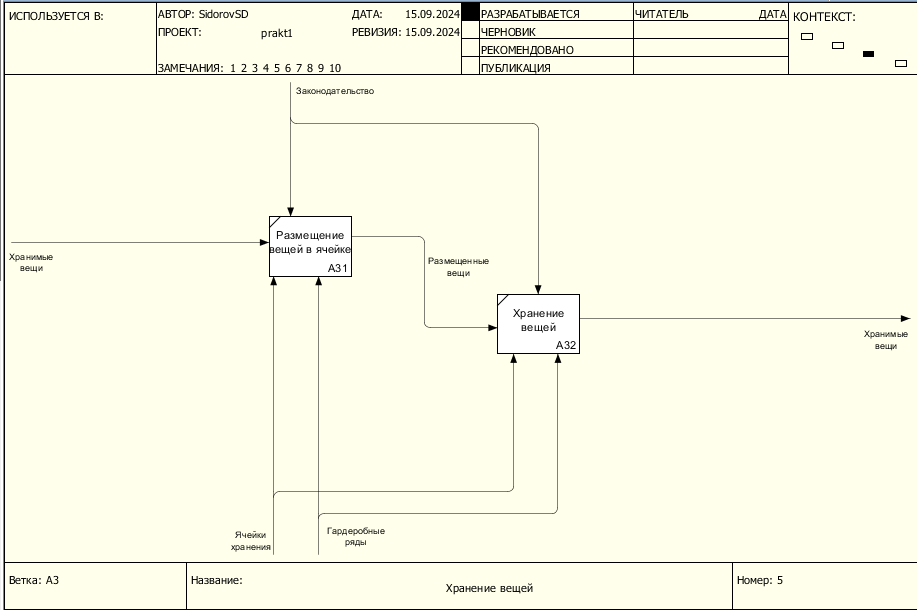


Рисунок 5 – Декомпозиция блока А3

Декомпозиция блока A4:

* Поиск необходимой ячейки – А41
* Получение вещей из ячейки – А42
* Передача вещей посетителю – А43

Диаграмма декомпозированного блока представлена на рисунке 6.

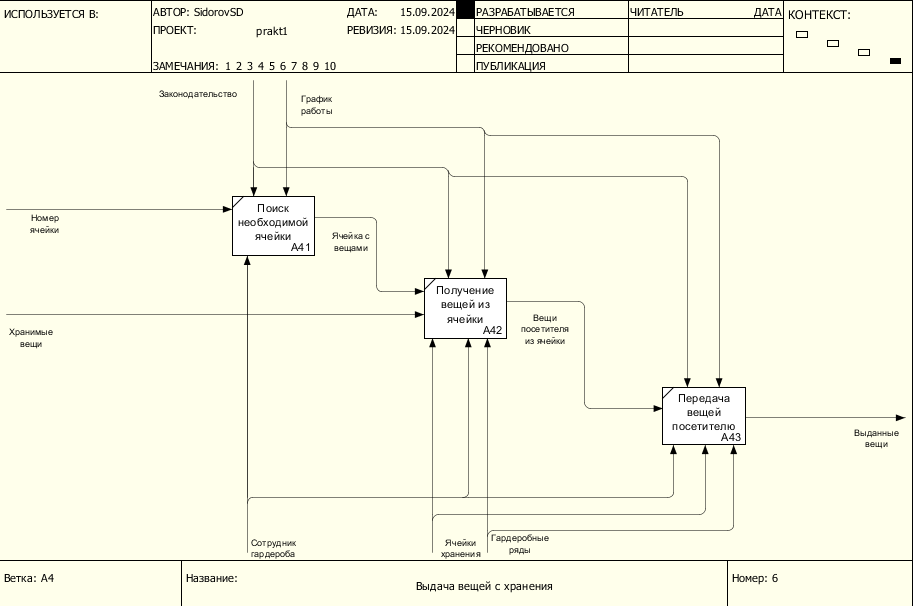


Рисунок 6 – Декомпозиция блока А4

**Вывод:**

В результате выполнения практической работы были получены теоретические знания в области диаграммы AS-IS, а также была разработана эта диаграмма в нотации IDEF-0.

# Практическая работа №2

**Цель работы:**

1. Знакомство с понятием функциональной модели TO-BE («как будет»).
2. Доработка созданной модели AS-IS с учетом выявленных недостатков в организации бизнес-процессов.

**Постановка задачи:**

Для заданной предметной области преобразовать созданную модель AS-IS в модель TO-BE. Внедрив информационную систему или клиент-серверную архитектуру.

**Ход работы:**

В результате анализа функциональной модели AS-IS, были сделаны выводы, как можно преобразовать модель в модель TO-BE.

Была добавлена автоматизированная система хранения в качестве механизма, позволяющая перенести процесс поиска необходимой ячейки, обработки пользовательских запросов, получение данных из уникального идентификатора пользователя, а также обработки результатов операций на вычислительные машины, что позволит разгрузить сотрудников гардероба и увеличить пропускную способность.

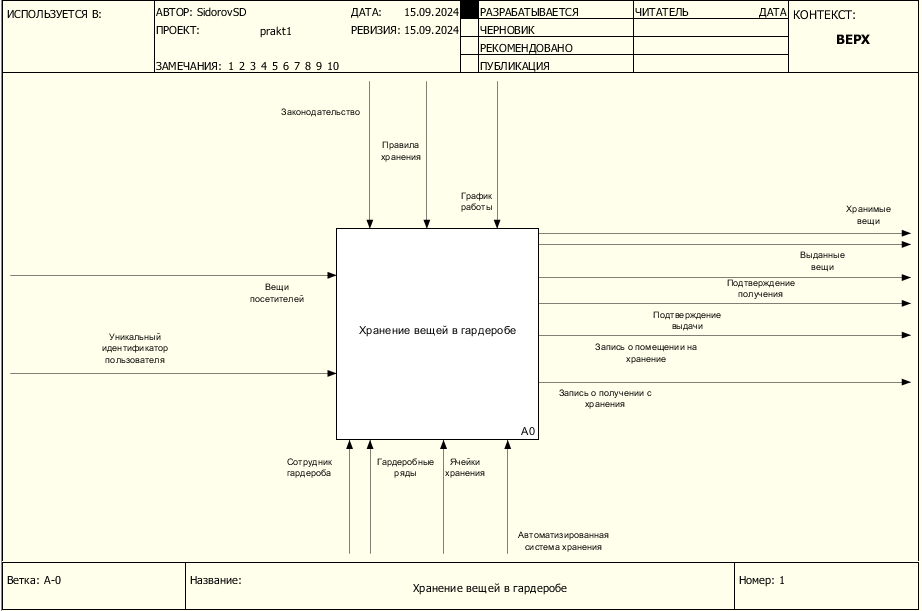


Рисунок 7 – Обновленный блок А0

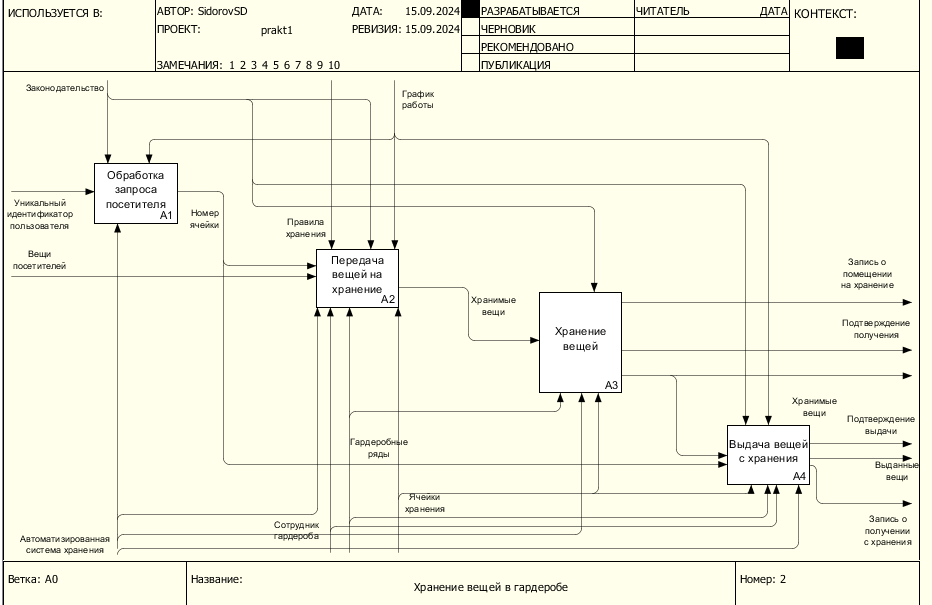


Рисунок 8 – Обновленная декомпозиция блока А0

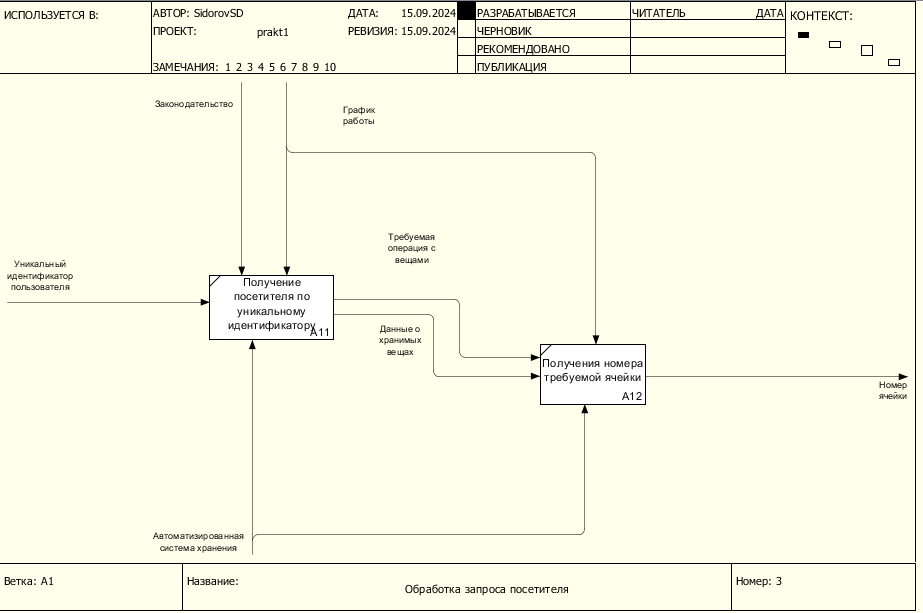


Рисунок 9 – Обновленная декомпозиция блока А1

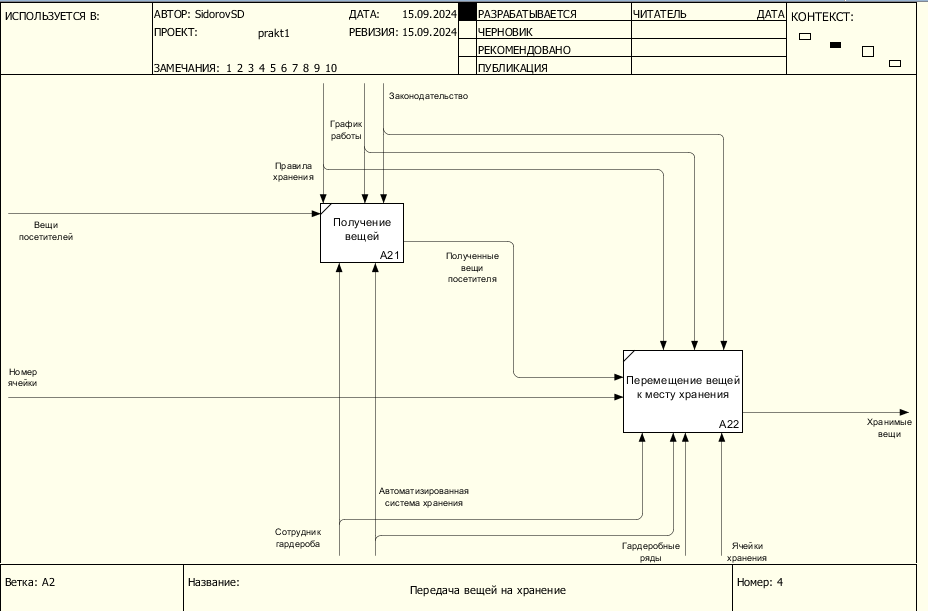


Рисунок 10 – Обновленная декомпозиция блока А2

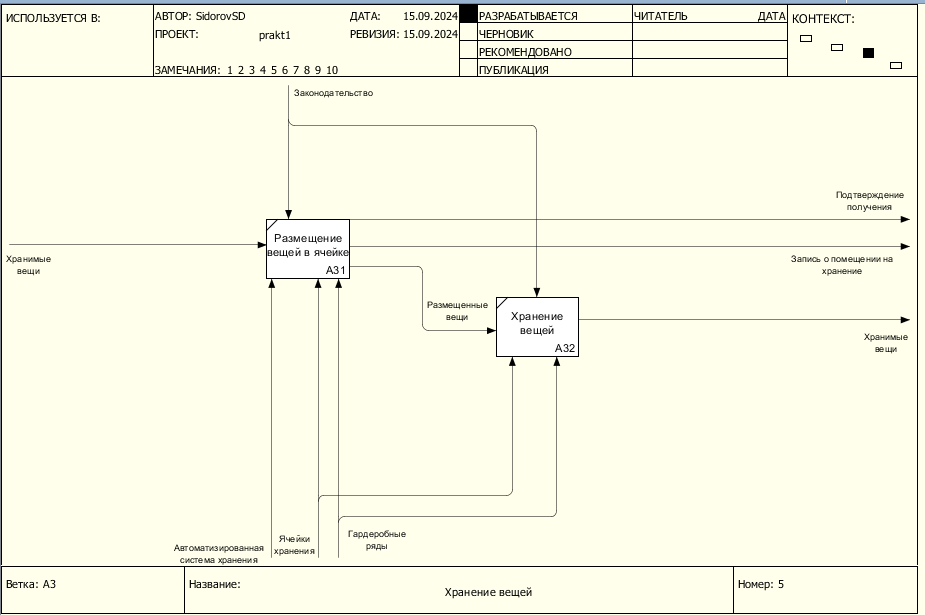


Рисунок 11 – Обновленная декомпозиция блока А3

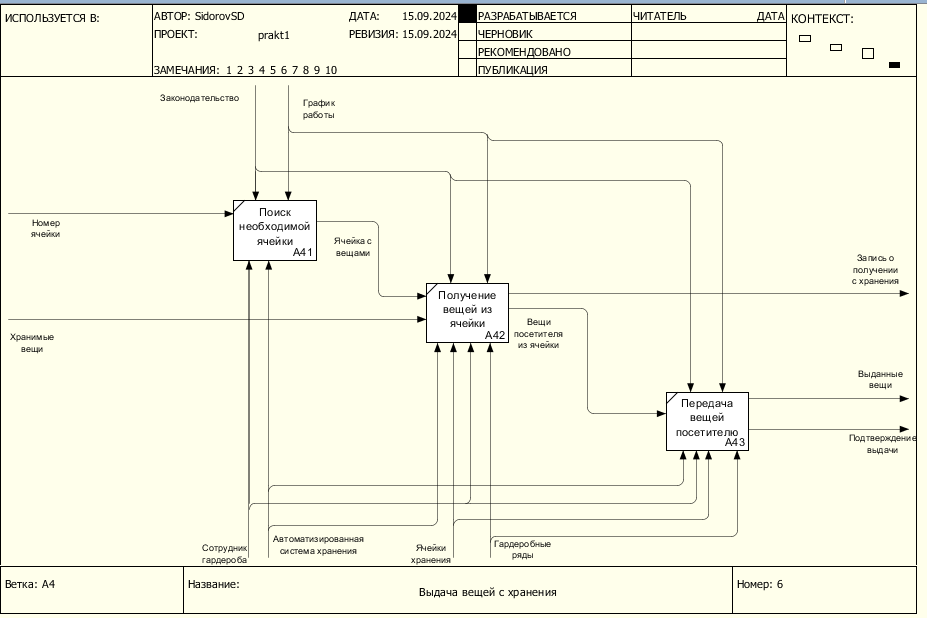


Рисунок 12 – Обновленная декомпозиция блока А4

**Вывод:**

В результате выполнения практической работы были получены теоретические знания в области диаграммы TO-BE, а также была разработана эта диаграмма в нотации IDEF-0.

# Практическая работа №3

**Цель работы:**

Получить практические навыки в построении IDEF3-модели процесса.

**Постановка задачи:**

С помощью методологии IDEF3 декомпозировать 1 из функциональных блоков модели окружения (A0), используя все типы перекрестков.

**Модель должна содержать:**

Перекрестки типа AND, OR, XOR

**Ход работы:**

Для выполнения практической работы был декомпозирован функциональный блок модели окружения под название “Обработка запроса пользователя”. Была получена модель информационных потоков и взаимоотношений между процессами обработки информации. Модель представлена на рисунке 13.

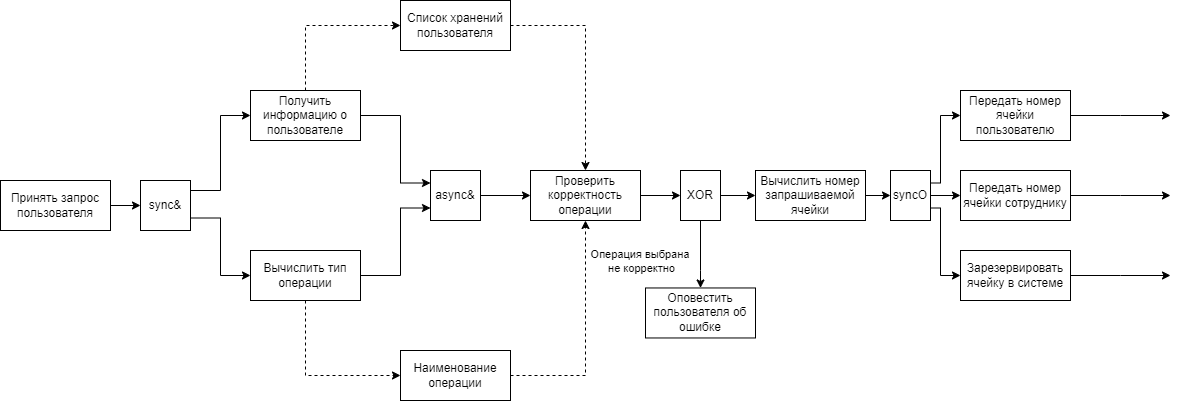


Рисунок 13 – Диаграмма WorkFlow, декомпозиция блока “Обработка запроса пользователя”

**Вывод:**

В результате выполнения практической работы были получены теоретические и практические знания в области диаграмм WorkFlow, а также была разработана эта диаграмма в нотации IDEF3.

# Практическая работа №4

**Цель работы:** получить практические навыки в построении DFD-модели бизнес-процесса.

**Постановка задачи:**

С помощью методологии DFD декомпозировать 1 из функциональных блоков. Можно выбрать часть процесса, который моделировался на предыдущих лабораторных работах. При выборе учтите, что процесс обязательно должен предусматривать обработку информации, лучше, чтобы это была автоматизированная обработка с использованием одной или нескольких информационных систем.

**Ход работы:**

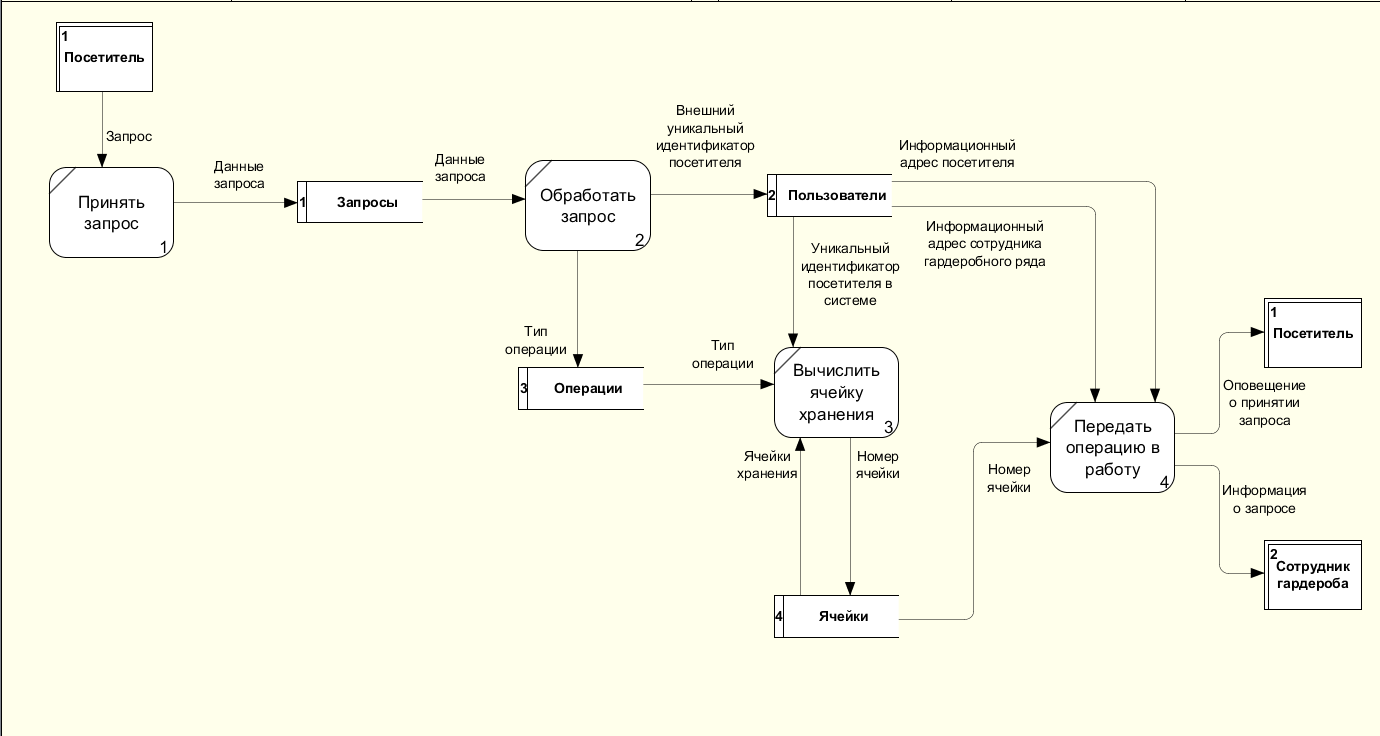
Для выполнения практической работы был декомпозирован процесс «Обработка запроса посетителя». В результате была получена модель бизнес-процесса, описывающая потоки информации, перемещающие между различными процессами в рамках обработки запроса пользователя. Модель представлена на рисунке 14.

Рисунок 14 – DFD-модель бизнес-процесса «Обработка запроса пользователя»

**Вывод:**

В результате выполнения практической работы были получены теоретические и практический знания в области диаграмм потоков данных, а также была разработана модель бизнес-процесса с помощью методологии DFD.

# Практическая работа №5

**Цель работы:** получить практические навыки в построении прецедентной UML-модели бизнес-процесса.

**Порядок выполнения работы.**

Выберите бизнес-процесс, для которого будете формировать модель. Вы можете выбрать один из вариантов процессов, описанных в приложении, или предложить свой вариант. Можно выбрать один из процессов, для которого на предыдущих лабораторных работах строилась модель по одной из структурных методологий. Желательно, чтобы процесс имел различные версии, т.е. альтернативные потоки событий.

Для заданной предметной области:

* построить диаграмму классов;
* построить диаграмму последовательности;
* построить диаграмму взаимодействий (диаграмму коммуникаций);
* построить диаграмму пакетов;

**Ход работы:**

Для выполнения данной практической работы был выбран процесс обработки запроса пользователя.

При выполнении данной работы была построена диаграмма классов, используемых в данном процессе, изображенная на рисунке 15.

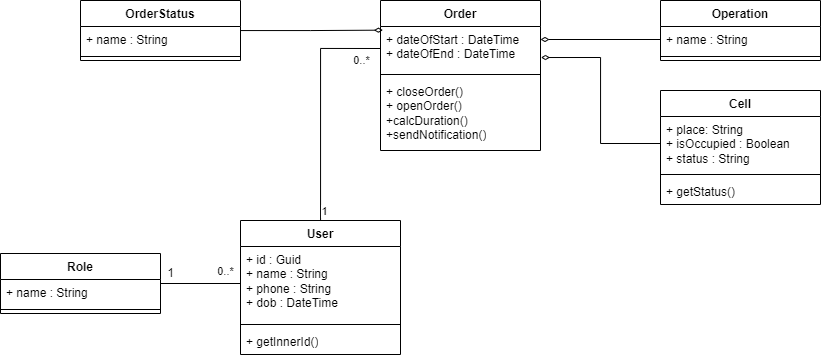


Рисунок 15 – Диаграмма классов

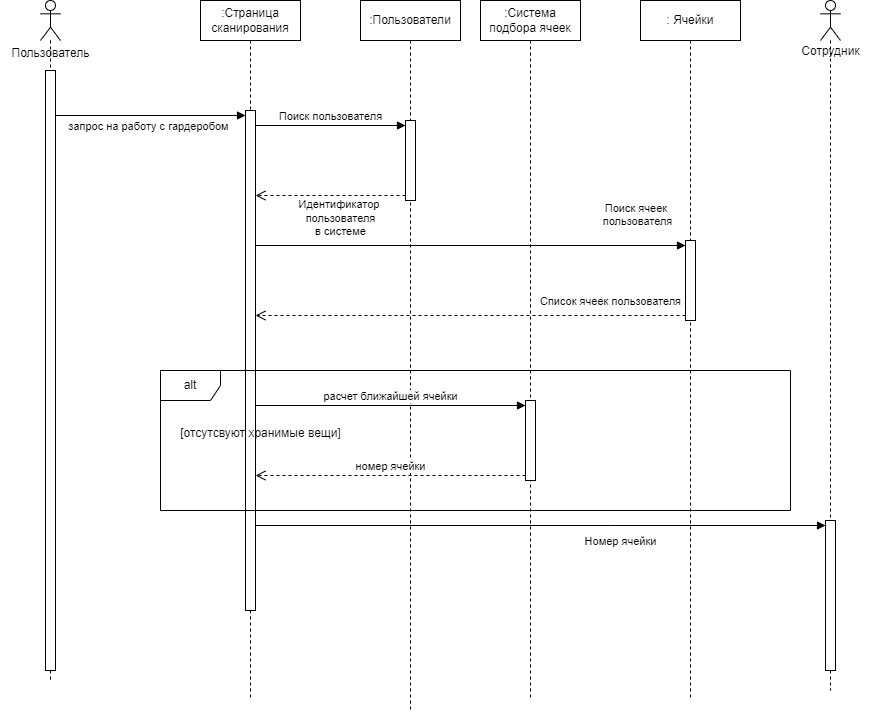
Также была построена диаграмма последовательности для этого процесса, изображенная на рисунке 16.

Рисунок 16 – Диаграмма последовательности процесса обработки запроса посетителя

Также для данного процесса были сформированы диаграммы обзора взаимодействий и пакетов, представленные на рисунках 17 – 18 соответственно.

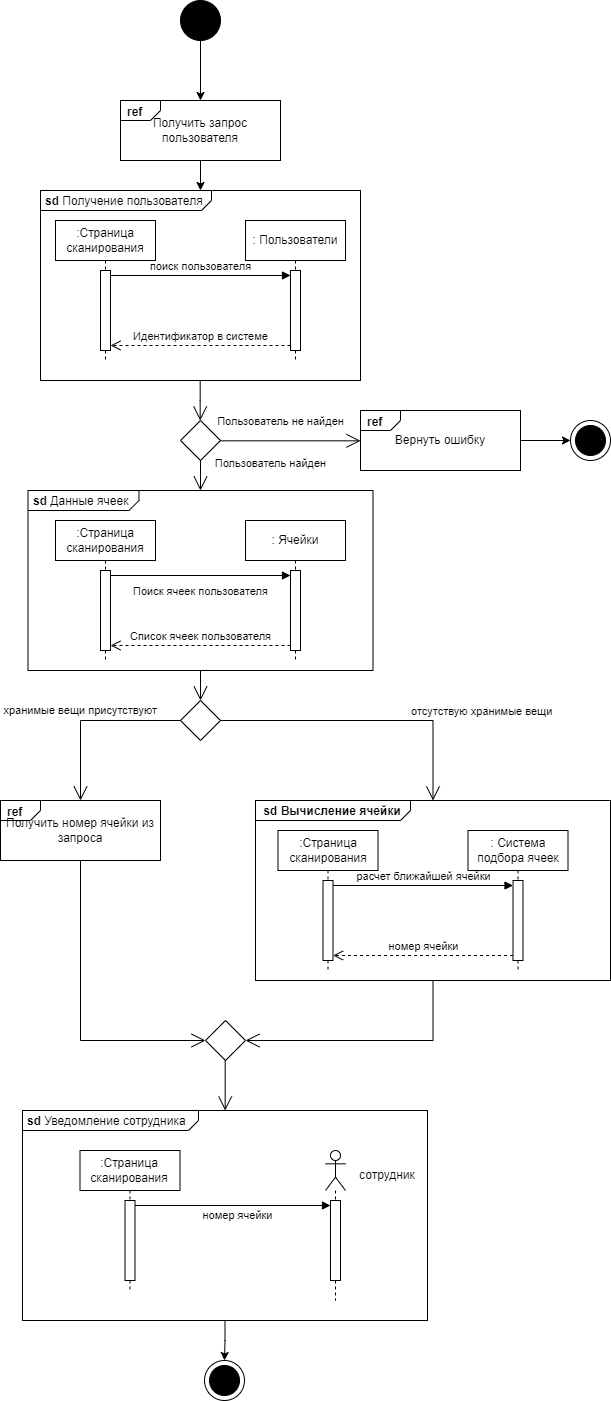


Рисунок 17 – Диаграмма обзора взаимодействий

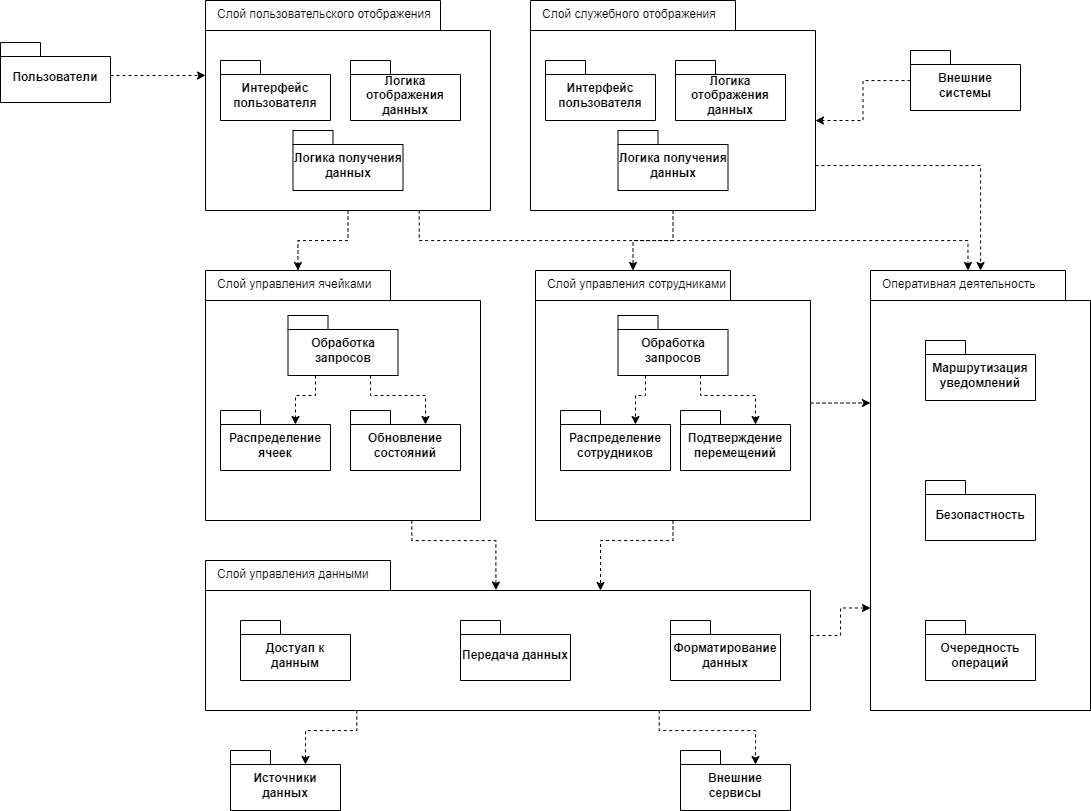


Рисунок 18 – Диаграмма пакетов

**Вывод:**

В ходе выполнения данной практической работы были получены навыки создания различных UML диаграмм, которые были применены на практике при создании диаграмм пакетов, обзора взаимодействий, последовательности, классов.

# Практическая работа №6

**Задание:**

Рассмотрите факторы, которые могут повлиять на масштабируемость системы, такие как архитектура, базы данных, алгоритмы, и другие.

Рассмотрите, какие изменения в архитектуре или инфраструктуре могут потребоваться для обеспечения требуемой масштабируемости.

По результатам анализа сделать мотивированный выбор архитектурной модели для проекта по выбранной тематике, описать возможности масштабирования проектируемой системы и связанные с этим риски.

Предложите стратегии масштабируемости, такие как использование облачных ресурсов, контейнеризация, кэширование, балансировка нагрузки и другие могут быть применены в вашем проекте.

Раскройте необходимые риски непосредственно в рамках вашего проекта.

Рассчитайте какое количество пользователей или запросов система должна обслуживать в вашем проекте?

**Ход работы:**

В рамках анализа были рассмотрены различные стратегии функционирования системы, а также произведен расчет предстоящих нагрузок на систему.

Было выявлено, что предложение в рамках одной организации среднего размера должно быть способно в течении суток обрабатывать от 5000 активных пользователей, совершающих операции по хранению и доставанию вещей. Также на каждый запрос посетителя приходится, как минимум два запроса сотрудника для принятия вещей и оповещении о совершении операции.

Для обеспечения комфортного уровня взаимодействия пользователя с системой в рамках организация среднего или крупного размера, требующих большой скорости обработка значительного потока посетителей в пиковые часы нагрузок, например ВУЗы или школы необходимо поддерживать обработку полного цикла взаимодействия с посетителем в рамках от 30 до 60 секунд, что позволит получить ощутимое преимущество над стандартной системой гардероба.

Основным фактором роста числа ежедневных пользователей системы в рамках данного проекта является подключение новых организаций, что приводит к скачкообразному росту пользователей, что может привести к неожиданному переполнению системы в случае недооценки посетителей новой организации. Основными факторами, которые уменьшают возможности к масштабированию системы является архитектура, критически влияющая на время исполнения запроса между различными элементами системы, базы данных не позволяющие оперативно получать необходимую информацию, а также различные физические ограничения масштабируемости в виде не хватки технического оснащения серверов.

Частично, большую часть из вышеописанных проблем может решить стратегия с использованием микросервисной архитектуры. Использование отдельных сервисов для задач распределения, аутентификации, контроля хранения и других позволит распределить нагрузку на отдельные вычислительные машины, что позволит уменьшить фактор недостатка технического оснащения.

Однако, микросервисная архитектура, при достаточно плотной взаимосвязи внутренних сервисов, должна быть устойчива к различным критическим ситуациям в виде выхода из строя одного из сервисов или недостатка вычислительных мощностей, выделенных под определенный сервис.   
 Для решения данных проблем предлагается использовать контейнеризацию, которая позволит быстро разворачивать различные контейнеры с сервисами, не позволяя им находится в не рабочем состоянии достаточно долго, а также улучшит возможности масштабирования привнеся возможность быстрого переезда или старта дополнительного экземпляра сервиса. Не мало важным фактором также является возможность каждого контейнера содержать базу данных, близость которой будет обеспечивать минимальное время отклика при запросах.

В качестве не менее важных вариантов развития масштабируемости системы предлагается в вести кэширование самых частых запросов с редко меняющимися данными, например запросы списка рядов, компаний или сотрудников, балансировку нагрузки между несколькими экземплярами одного и того же сервиса за счет различных оркестрантов в виде Kubernetes или Docker Swarm, а также как один из вариантов обеспечения приемлемого времени отклика для крупных компаний, размещение части или всех сервисов на вычислительных мощностях организации, благодаря чему удастся повысить безопасность личных данных компании, а также переложить нагрузку с серверов проекта на сервера заказчика, где они сами смогут регулировать пропускную способность.

Выбранные выше решения несут за собой риски связанные с увеличением трудозатрат на поддержание, обновление и обслуживание различных сервисов и вычислительных мощностей их поддерживающих, что особенно заметно на on-premise версии проекта, что может привести к ошибкам в обработке данных или большему времени цикла полного взаимодействия с пользователем, чем в стандартной не автоматизированной системе.  
 **Вывод:** В рамках выполнения данной практической работы были рассмотрены различные варианты масштабирования системы. В результате была выбрана микросервисная архитектура, с возможностью on-premise установки.

# Практическая работа №7

Для анализа архитектуры клиент-серверной системы автоматического гардероба были рассмотрены три типа архитектур: монолитная, модульная и микросервисная. Каждая из них имеет свои особенности, преимущества и недостатки, которые определяют потенциал для модификации и масштабирования системы.

Монолитная архитектура предполагает объединение всех компонентов системы (учет вещей, регистрация клиентов, управление ячейками) в одно приложение. Основной недостаток такого подхода заключается в сложности внесения изменений и ограничениях масштабирования. В случае увеличения нагрузки приходится развертывать несколько полных копий приложения, что приводит к перегрузке сервера и усложняет управление ресурсами. Это затрудняет модификацию отдельных функций и повышает вероятность возникновения ошибок при обновлении системы.

Модульная архитектура, в отличие от монолитной, позволяет разделить систему на отдельные модули, такие как управление пользователями, учет вещей и управление ячейками. Каждый модуль функционирует независимо, что упрощает внесение изменений и снижает риски, связанные с обновлением. Однако модули все еще остаются частью одного приложения, и их масштабирование требует запуска дополнительных копий всего приложения. Это ограничивает гибкость и приводит к неравномерному распределению нагрузки между модулями.

Микросервисная архитектура, которая была выбрана в рамках данного анализа, обеспечивает наибольшую гибкость и независимость компонентов системы. Суть данной архитектуры заключается в разделении системы на независимые микросервисы, каждый из которых отвечает за конкретную функциональность, например, регистрацию клиентов, учет вещей или управление ячейками. Микросервисы взаимодействуют между собой через API, что позволяет каждому из них масштабироваться независимо. Это дает возможность эффективно распределять ресурсы в зависимости от текущей нагрузки на конкретный сервис. Например, в случае увеличения количества запросов на регистрацию новых клиентов можно масштабировать только соответствующий микросервис, не затрагивая остальные компоненты системы.

Основной риск, связанный с микросервисной архитектурой, заключается в увеличении сложности координации и управления микросервисами. Для решения этой проблемы необходимо использовать оркестраторы, такие как Kubernetes или Docker Swarm, а также системы мониторинга и логирования для отслеживания состояния микросервисов. Дополнительно важно реализовать отказоустойчивую архитектуру, которая будет устойчива к сбоям отдельных сервисов.

Для успешного внедрения микросервисной архитектуры и обеспечения её масштабируемости предложены следующие стратегии: использование контейнеризации, что позволит быстро развертывать новые копии сервисов и снижать время простоя; применение кэширования для наиболее частых запросов; балансировка нагрузки между несколькими экземплярами сервисов с помощью оркестраторов; возможность on-premise установки на серверы клиента для крупных организаций, что обеспечит снижение времени отклика и повысит безопасность.

**Выводы:** Таким образом, микросервисная архитектура была выбрана как наилучшее решение для системы автоматического гардероба, так как она позволяет эффективно управлять ростом количества пользователей, поддерживает независимое масштабирование сервисов и упрощает модификацию системы.

# Практическая работа №8

**Цель задания:**

Провести анализ и изучение применения шаблонов и паттернов проектирования в контексте клиент-серверных систем.

**Задание:**

1. Рассмотрите следующие шаблоны и паттерны:

- Шаблоны клиент-серверных систем (например, шаблоны для взаимодействия клиента и сервера).

- Структурные паттерны (например, "Адаптер" (Adapter) для интеграции разных интерфейсов, "Мост" (Bridge) для разделения абстракции и реализации).

- Паттерны поведения (например, "Состояние" (State) для управления состоянием системы, "Стратегия" (Strategy) для замены алгоритмов).

- Порождающие паттерны (например, "Фабричный метод" (Factory Method) для со здания объектов, "Одиночка" (Singleton) для гарантированной единственной инстанциации).

2. Выбор конкретных шаблонов и паттернов: Выберите несколько конкретных шаблонов и паттернов, которые, наиболее подходят для применения в вашей клиент-серверной системе. Технически грамотно обоснуйте выбор этих паттернов.

3. Анализ применения: рассмотрите, как выбранные вами шаблоны и паттерны могут быть использованы для улучшения архитектуры и функциональности вашей системы. Опишите, как эти шаблоны и паттерны могут решать конкретные проблемы или улучшать производительность проектируемой клиент-серверной системы.

4. Проектирование и реализация: спроектируйте часть системы, используя выбранные шаблоны и паттерны. Результаты проектирования представьте в виде диаграмм и текстового описания, если потребуется добавьте глоссарий.

5. Отчет: подготовьте отчет, включающий ваш анализ применения шаблонов и паттернов, описания применения в выбранной системе и, результаты проектирования.

**Ход работы:**

Для анализа применения шаблонов и паттернов проектирования в контексте системы автоматизированного гардероба был рассмотрен ряд структурных, поведенческих и порождающих паттернов. Основной целью являлось выявление наиболее подходящих решений, которые помогут повысить гибкость системы, упростить взаимодействие между компонентами и обеспечить эффективное масштабирование.

Поскольку система автоматизированного гардероба построена на микросервисной архитектуре, основное внимание уделено паттернам, которые помогают управлять взаимодействием между независимыми сервисами и обеспечивают надежное и безопасное хранение данных.

1. **Шаблон "Singleton" (Одиночка):** Этот паттерн используется для управления подключением к базе данных, обеспечивая наличие единственного экземпляра подключения. Это позволяет исключить возможность создания нескольких подключений к базе данных, что минимизирует риск конфликтов и избыточных запросов. Реализация Singleton позволяет централизовать управление доступом к базе данных, повысить производительность системы и снизить затраты на ресурсы.
2. **Шаблон "Factory Method" (Фабричный метод):** В системе автоматического гардероба этот паттерн применяется для создания объектов, таких как учетные записи пользователей или записи о предметах одежды. Паттерн позволяет создавать объекты на основе переданных параметров, обеспечивая гибкость в расширении системы. Например, при необходимости добавления нового типа ячейки или категории предмета не потребуется вносить изменения в код логики, а достаточно будет добавить новую фабрику, которая будет создавать соответствующие объекты. Это упрощает расширение системы и повышает её гибкость.
3. **Шаблон "Observer" (Наблюдатель):** Данный паттерн используется для отслеживания изменения состояния ячеек в гардеробе и уведомления всех заинтересованных микросервисов о событиях. Например, при изменении состояния ячейки (занята или свободна) соответствующие микросервисы (управление учётом вещей, уведомление сотрудников) получают уведомления и могут предпринять дальнейшие действия. Паттерн "Наблюдатель" позволяет минимизировать взаимодействие между сервисами, так как каждый микросервис может подписаться только на те изменения, которые его касаются, что снижает нагрузку на систему и увеличивает её отзывчивость.
4. **Шаблон "Adapter" (Адаптер):** Использование паттерна "Adapter" позволяет интегрировать систему автоматического гардероба с внешними API и системами, такими как учетные системы пользователей или системы IoT для управления автоматическими ячейками и рядами. Адаптеры предоставляют единый интерфейс для взаимодействия с различными системами, что упрощает добавление новых интеграций и минимизирует изменения в исходном коде при замене внешних сервисов.
5. **Паттерн "Facade" (Фасад):** Для упрощения взаимодействия между различными микросервисами используется паттерн "Facade". Он объединяет несколько мелких сервисов в единую точку входа, предоставляя упрощенный интерфейс для выполнения комплексных операций. Это снижает количество прямых взаимодействий между микросервисами, упрощает координацию и уменьшает вероятность ошибок при вызове сервисов. Например, фасад может объединять операции регистрации новых пользователей и проверки доступности ячеек в одном запросе.
6. **Паттерн "Circuit Breaker" (Размыкатель цепи):** Паттерн "Circuit Breaker" используется для управления сбоями и ошибками при взаимодействии между микросервисами. Если один из сервисов временно недоступен или работает некорректно, "Circuit Breaker" позволяет временно остановить отправку запросов к этому сервису и перенаправить их на резервный сервис или вернуть сообщение об ошибке пользователю. Это снижает риск перегрузки системы и предотвращает цепную реакцию отказов.

Для реализации вышеописанных паттернов в системе автоматического гардероба предлагается следующая структура:

* **Singleton:** использовать для создания единого подключения к базе данных, которое будет управлять всеми операциями чтения и записи.
* **Factory Method:** применить для создания объектов при регистрации новых пользователей или добавлении новых вещей в гардероб.
* **Observer:** реализовать для уведомления микросервисов о событиях изменения состояния ячеек.
* **Adapter:** внедрить для взаимодействия с внешними системами, например, для подключения системы автоматических шкафов или учетных систем.
* **Facade:** создать фасадные сервисы, объединяющие несколько микросервисов, чтобы сократить количество запросов и улучшить координацию между ними.
* **Circuit Breaker:** включить в реализацию микросервисов для управления сбоями и повышения отказоустойчивости системы.

# Практическая работа №9

Задание:

1. Изучить возможности PlantUML по созданию продвинутых типов диаграмм.
2. Создать следующие диаграммы для своей системы с использованием PlantUML:

* Диаграмму состояний для ключевого компонента или объекта.
* Диаграмму компонентов, отражающую архитектуру системы.
* Диаграмму развертывания, показывающую физическое распределение системы.
* Диаграмму пакетов для организации классов и компонентов.

1. Применить стилизацию и темизацию к созданным диаграммам.
2. Документировать процесс создания диаграмм и обосновать принятые решения.

**Выполнение работы №9**

Для создания диаграммы состояний, представленной на рисунке 19, было выбран процесс обработки запроса пользователя.



Рисунок 21 – Диаграмма состояний

Листинг для данной диаграммы представлен в листинге 1.

Листинг 1 – plantUML код для диаграммы состояний

@startuml

[\*] --> ЗапросОбрабатывается

ЗапросОбрабатывается --> ПроверкаДанных : данные поступили

ПроверкаДанных --> НайденоМесто : данные валидны

ПроверкаДанных --> Ошибка : данные невалидны

НайденоМесто --> МестоВыбрано : подтверждение от клиента

МестоВыбрано --> СессияЗакрыта : завершение операции

Ошибка --> СессияЗакрыта : уведомление об ошибке

СессияЗакрыта --> [\*]

@enduml

Далее была построена диаграмма компонентов, представленная на рисунке 22, построенная кодом, представленным в листинге 2.

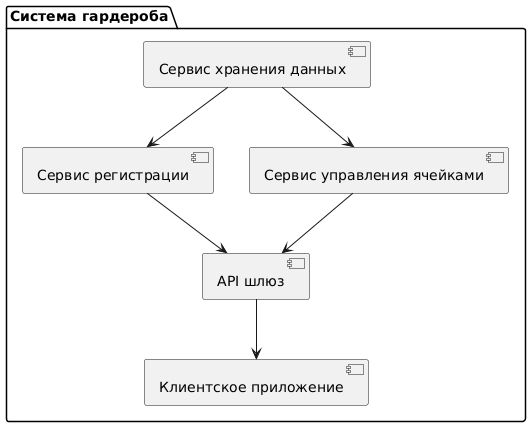


Рисунок 22 – Диаграмма компонентов

Листинг 2 – plantUML код для диаграммы компонентов

@startuml

package "Система гардероба" {

[Сервис регистрации]

[Сервис управления ячейками]

[Сервис хранения данных]

[API шлюз]

[Клиентское приложение]

}

[Сервис регистрации] --> [API шлюз]

[Сервис управления ячейками] --> [API шлюз]

[API шлюз] --> [Клиентское приложение]

[Сервис хранения данных] --> [Сервис управления ячейками]

[Сервис хранения данных] --> [Сервис регистрации]

@enduml

Также была построена диаграмма развертывания, которая показывает, как программные компоненты распределены на аппаратных узлах. Итоговая диаграмма продемонстрирована на рисунке 23, а код для её создания в листинге 3.



Рисунок 23 – Диаграмма развертывания

Листинг 3 – plantUML

@startuml

node "Центральный сервер" {

[API шлюз]

[Сервис регистрации]

[Сервис управления ячейками]

[Сервис хранения данных]

}

node "Клиентское устройство" {

[Клиентское приложение]

}

[Клиентское приложение] --> [API шлюз]

@enduml

Далее была создана диаграмма пакетов. Она помогает логически организовать элементы системы, группируя их по функциональным областям. Результат создания и используемый код представлен на рисунке 24 и листинге 4 соответственно.

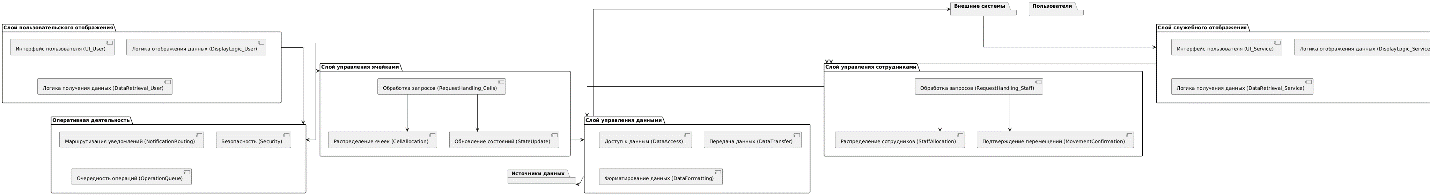


Рисунок 24 – Диаграмма пакетов

Листинг 4 – plantUML код для диаграммы пакетов

@startuml

skinparam linetype ortho

package "Пользователи" {

}

package "Источники данных" {

}

package "Внешние системы" {

}

package "Слой пользовательского отображения" {

[Интерфейс пользователя (UI\_User)]

[Логика отображения данных (DisplayLogic\_User)]

[Логика получения данных (DataRetrieval\_User)]

}

package "Слой служебного отображения" {

[Интерфейс пользователя (UI\_Service)]

[Логика отображения данных (DisplayLogic\_Service)]

[Логика получения данных (DataRetrieval\_Service)]

}

package "Слой управления ячейками" {

[Обработка запросов (RequestHandling\_Cells)] --> [Распределение ячеек (CellAllocation)]

[Обработка запросов (RequestHandling\_Cells)] --> [Обновление состояний (StateUpdate)]

}

package "Слой управления сотрудниками" {

[Обработка запросов (RequestHandling\_Staff)] --> [Распределение сотрудников (StaffAllocation)]

[Обработка запросов (RequestHandling\_Staff)] --> [Подтверждение перемещений (MovementConfirmation)]

}

package "Оперативная деятельность" {

[Маршрутизация уведомлений (NotificationRouting)]

[Безопасность (Security)]

[Очередность операций (OperationQueue)]

}

package "Слой управления данными" {

[Доступ к данным (DataAccess)]

[Передача данных (DataTransfer)]

[Форматирование данных (DataFormatting)]

}

[Внешние системы] --> [Слой служебного отображения]

[Слой пользовательского отображения] -down-> [Слой управления ячейками]

[Слой пользовательского отображения] -down-> [Слой управления сотрудниками]

[Слой пользовательского отображения] -down-> [Оперативная деятельность]

[Слой служебного отображения] -down-> [Слой управления сотрудниками]

[Слой служебного отображения] -down-> [Оперативная деятельность]

[Слой управления ячейками] -down-> [Слой управления данными]

[Слой управления сотрудниками] -down-> [Слой управления данными]

[Слой управления данными] -down-> [Источники данных]

[Слой управления данными] -up-> [Внешние системы]

@enduml

# Практическая работа №10

Задание:

* Освоить использование различных инструментов и нотаций для моделирования информационных систем.
* Научиться создавать сетевые диаграммы с помощью nwdiag.
* Познакомиться с архитектурным моделированием с использованием языка ArchiMate.
* Составить структуру разбиения работ (WBS) для планирования проекта.
* Применить эти инструменты при выполнении практических заданий и курсовой работы

**Выполнение работы №10**

Сетевая диаграмма помогает наглядно отразить сетевую инфраструктуру системы. Она показывает устройства, их IP-адреса, а также связи между ними. Итоговая диаграмма представлена на рисунке 25, а код к ней в листинге 5.

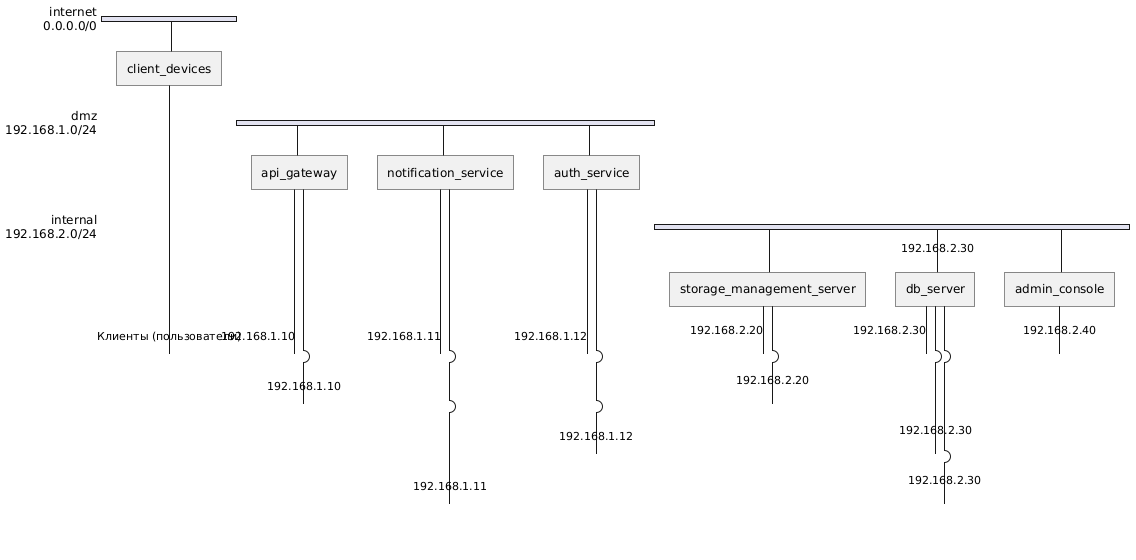


Рисунок 25 – Сетевая диаграмма

Листинг 5 – plantUML код для сетевой диаграммы

@startuml

nwdiag {

network internet {

address = "0.0.0.0/0";

client\_devices [address = "Клиенты (пользователи)"];

}

network dmz {

address = "192.168.1.0/24";

api\_gateway [address = "192.168.1.10"];

notification\_service [address = "192.168.1.11"];

auth\_service [address = "192.168.1.12"];

}

network internal {

address = "192.168.2.0/24";

storage\_management\_server [address = "192.168.2.20"];

db\_server [address = "192.168.2.30"];

admin\_console [address = "192.168.2.40"];

}

// Connections between components

client\_devices -- api\_gateway;

api\_gateway -- storage\_management\_server;

storage\_management\_server -- db\_server;

storage\_management\_server -- notification\_service;

storage\_management\_server -- auth\_service;

auth\_service -- db\_server;

notification\_service -- db\_server;

admin\_console -- db\_server;

}

@enduml

ArchiMate - это язык моделирования архитектуры предприятий, который поддерживается в PlantUML через специальные ключевые слова и макросы. Данный инструмент помогает отразить архитектуру системы на различных уровнях, а точнее показать то, как взаимодействуют между собой компоненты системы на уровне бизнес-процессов, приложений и технологий. Итоговая диаграмма представлена на рисунке 26, а код к ней в листинге 6.

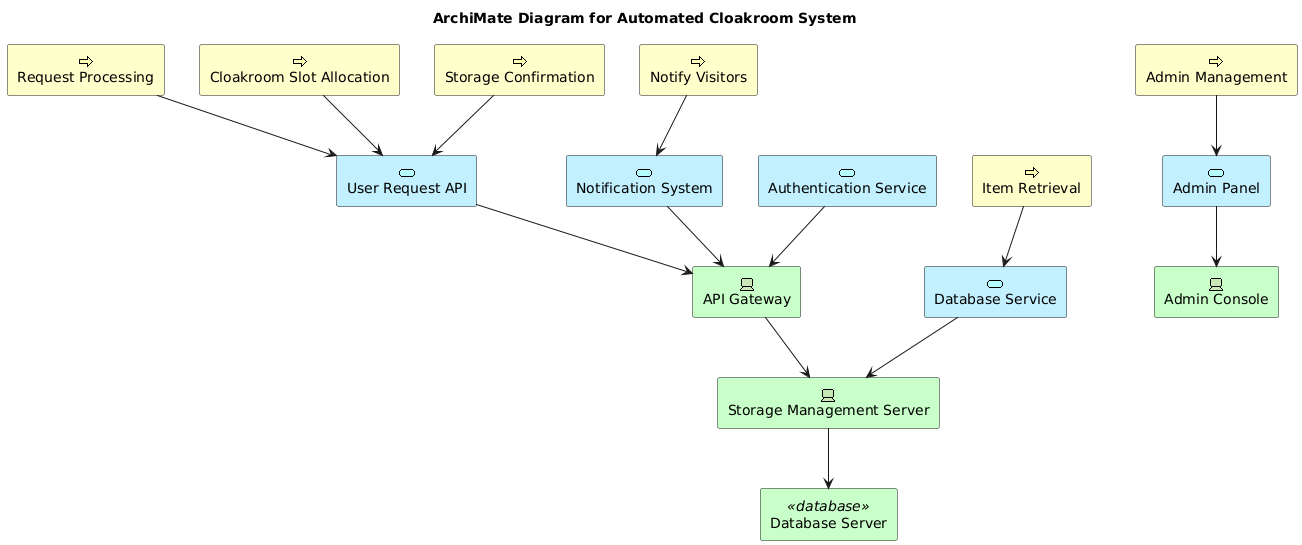


Рисунок 26 – Архитектурная диаграмма

Листинг 6 – plantUML код для архитектурной диаграммы

@startuml

title ArchiMate Diagram for Automated Cloakroom System

<style>

archimateDiagram {

.business-process {

BackgroundColor lightgreen

}

.application-service {

BackgroundColor lightblue

}

.technology-device {

BackgroundColor lightcoral

}

.database {

BackgroundColor plum

}

}

</style>

archimate #Business "Request Processing" as RequestProcessing <<business-process>>

archimate #Business "Cloakroom Slot Allocation" as SlotAllocation <<business-process>>

archimate #Business "Storage Confirmation" as StorageConfirmation <<business-process>>

archimate #Business "Notify Visitors" as NotifyVisitors <<business-process>>

archimate #Business "Item Retrieval" as ItemRetrieval <<business-process>>

archimate #Business "Admin Management" as AdminManagement <<business-process>>

archimate #Application "User Request API" as UserRequestAPI <<application-service>>

archimate #Application "Notification System" as NotificationSystem <<application-service>>

archimate #Application "Authentication Service" as AuthService <<application-service>>

archimate #Application "Database Service" as DatabaseService <<application-service>>

archimate #Application "Admin Panel" as AdminPanel <<application-service>>

archimate #Technology "API Gateway" as APIGateway <<technology-device>>

archimate #Technology "Storage Management Server" as StorageServer <<technology-device>>

archimate #Technology "Database Server" as DatabaseServer <<database>>

archimate #Technology "Admin Console" as AdminConsole <<technology-device>>

RequestProcessing --> UserRequestAPI

SlotAllocation --> UserRequestAPI

StorageConfirmation --> UserRequestAPI

NotifyVisitors --> NotificationSystem

ItemRetrieval --> DatabaseService

AdminManagement --> AdminPanel

UserRequestAPI --> APIGateway

NotificationSystem --> APIGateway

AuthService --> APIGateway

DatabaseService --> StorageServer

AdminPanel --> AdminConsole

APIGateway --> StorageServer

StorageServer --> DatabaseServer

@enduml

И наконец WBS диаграмма, которая помогает организовать и структурировать проект, разбивая его на более мелкие, управляемые компоненты. Итоговая диаграмма представлена на рисунках 27 - 29 и листинге 7.

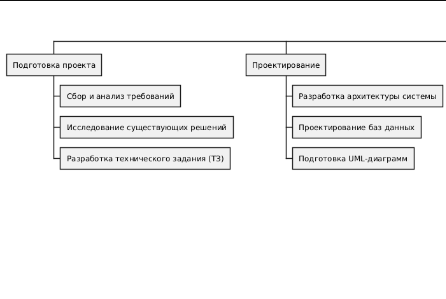


Рисунок 27 – Начало WBS диаграммы



Рисунок 28 – Середина WBS диаграммы

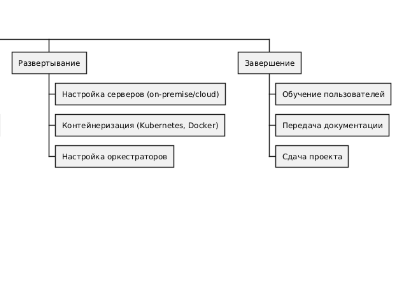


Рисунок 29 – Конец WBS диаграммы

Листинг 7 – plantUML код для WBS диаграммы

@startwbs

\* Проект: Система автоматизированного гардероба

\*\* Подготовка проекта

\*\*\* Сбор и анализ требований

\*\*\* Исследование существующих решений

\*\*\* Разработка технического задания (ТЗ)

\*\* Проектирование

\*\*\* Разработка архитектуры системы

\*\*\* Проектирование баз данных

\*\*\* Подготовка UML-диаграмм

\*\* Разработка

\*\*\* Создание микросервисов

\*\*\*\* Реализация сервиса аутентификации

\*\*\*\* Реализация сервиса управления ячейками

\*\*\*\* Реализация сервиса уведомлений

\*\*\* Настройка API-шлюза

\*\*\* Реализация модулей безопасности

\*\*\* Настройка базы данных

\*\* Интеграция и тестирование

\*\*\* Интеграция микросервисов

\*\*\* Функциональное тестирование

\*\*\* Нагрузочное тестирование

\*\* Развертывание

\*\*\* Настройка серверов (on-premise/cloud)

\*\*\* Контейнеризация (Kubernetes, Docker)

\*\*\* Настройка оркестраторов

\*\* Завершение

\*\*\* Обучение пользователей

\*\*\* Передача документации

\*\*\* Сдача проекта

@endwbs

# Практическая работа №11

Задание:

* Изучить теоретические основы C4-модели и ее уровни.
* Настроить среду разработки для использования C4-модели в PlantUML.
* Создать диаграммы уровней Context, Container и Component для своей системы.
* Разработать диаграммы уровня Code
* Документировать процесс работы, обосновать архитектурные решения и представить результаты.

**Выполнение работы №11**

C4-модель — это простой и эффективный способ визуализации архитектуры программных систем. Она предлагает иерархический подход к описанию системы на разных уровнях абстракции.

Диаграмма уровня Context позволяет показать систему в окружении с действующими лицами (пользователями) и внешними системами. Итоговая диаграмма, представленная на рисунке 30 с кодом в листинге 8.

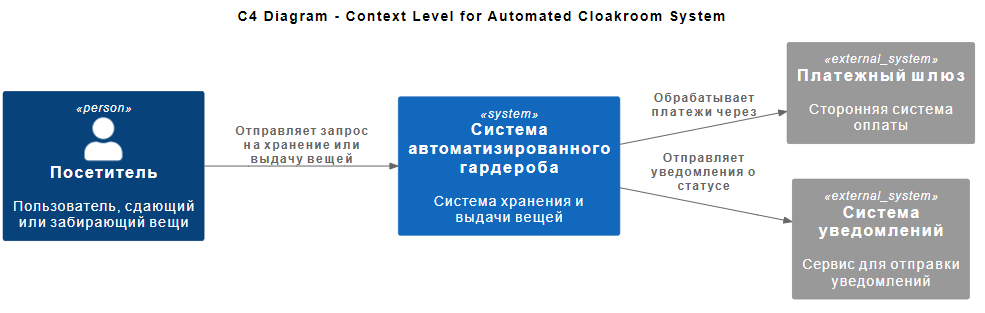


Рисунок 30 – Диаграмма уровня Context

Листинг 8 – plantUML код для диаграммы Context

@startuml

!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4\_Context.puml

LAYOUT\_LEFT\_RIGHT()

title C4 Diagram - Context Level for Automated Cloakroom System

Person(visitor, "Посетитель", "Пользователь, сдающий или забирающий вещи")

System(acs, "Система автоматизированного гардероба", "Система хранения и выдачи вещей")

System\_Ext(paymentGateway, "Платежный шлюз", "Сторонняя система оплаты")

System\_Ext(notificationService, "Система уведомлений", "Сервис для отправки уведомлений")

Rel(visitor, acs, "Отправляет запрос на хранение или выдачу вещей")

Rel(acs, paymentGateway, "Обрабатывает платежи через")

Rel(acs, notificationService, "Отправляет уведомления о статусе")

@enduml

Диаграмма уровня Container позволяет чуть больше углубиться в специфику нашей системы. Она разбивает нашу систему на основные компоненты, которые могут быть отдельными приложениями, сервисами или базами данных. Итоговая диаграмма представлена на рисунке 31 с кодом в листинге 9.

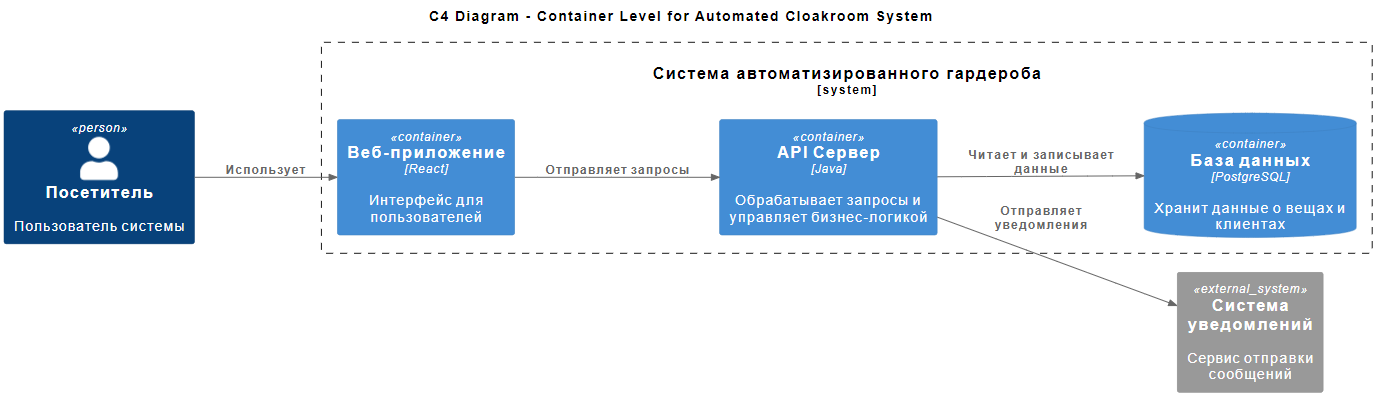


Рисунок 31 – диаграмма уровня Container

Листинг 9 – plantUML код для диаграммы уровня Container

@startuml

!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4\_Container.puml

LAYOUT\_LEFT\_RIGHT()

title C4 Diagram - Container Level for Automated Cloakroom System

Person(visitor, "Посетитель", "Пользователь системы")

System\_Boundary(acs\_boundary, "Система автоматизированного гардероба") {

Container(webApp, "Веб-приложение", "React", "Интерфейс для пользователей")

Container(apiServer, "API Сервер", "Java", "Обрабатывает запросы и управляет бизнес-логикой")

ContainerDb(database, "База данных", "PostgreSQL", "Хранит данные о вещах и клиентах")

}

System\_Ext(notificationService, "Система уведомлений", "Сервис отправки сообщений")

Rel(visitor, webApp, "Использует")

Rel(webApp, apiServer, "Отправляет запросы")

Rel(apiServer, database, "Читает и записывает данные")

Rel(apiServer, notificationService, "Отправляет уведомления")

@enduml

Диаграмма уровня Component детализирует выбранный контейнер, показывая его внутренние компоненты. Данный подход позволяет более детально рассмотреть интересующие нас моменты в конкретном аспекте системы. Итоговая диаграмма представлена на рисунке 32 с кодом в листинге 10.

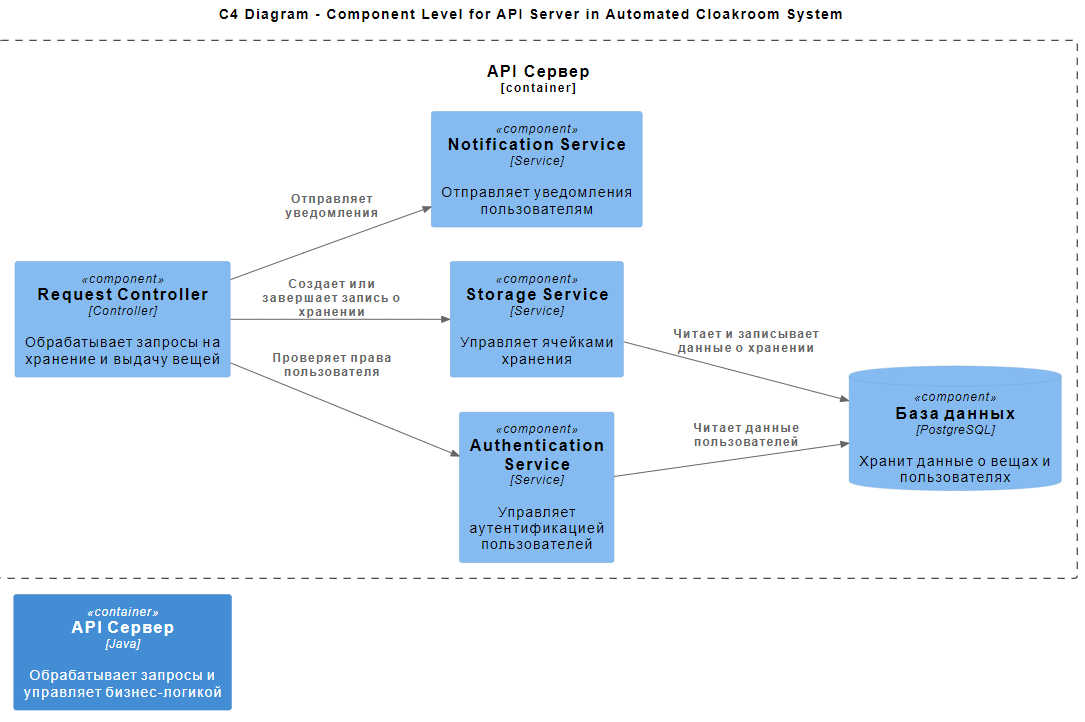


Рисунок 32 – Диаграмма уровня Container

Листинг 10 – plantUML код для диаграммы уровня Container

@startuml

!include https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4\_Component.puml

LAYOUT\_LEFT\_RIGHT()

title C4 Diagram - Component Level for API Server in Automated Cloakroom System

Container(apiServer, "API Сервер", "Java", "Обрабатывает запросы и управляет бизнес-логикой")

Container\_Boundary(apiServer\_boundary, "API Сервер") {

Component(requestController, "Request Controller", "Controller", "Обрабатывает запросы на хранение и выдачу вещей")

Component(authService, "Authentication Service", "Service", "Управляет аутентификацией пользователей")

Component(storageService, "Storage Service", "Service", "Управляет ячейками хранения")

Component(notificationService, "Notification Service", "Service", "Отправляет уведомления пользователям")

ComponentDb(database, "База данных", "PostgreSQL", "Хранит данные о вещах и пользователях")

}

Rel(requestController, authService, "Проверяет права пользователя")

Rel(requestController, storageService, "Создает или завершает запись о хранении")

Rel(requestController, notificationService, "Отправляет уведомления")

Rel(storageService, database, "Читает и записывает данные о хранении")

Rel(authService, database, "Читает данные пользователей")

@enduml

И наиболее низкий уровень детализации – диаграмма уровня Code. Она позволяет рассмотреть конкретный компонент системы с помощью уже знакомых нам UML-диаграмм классов. Итоговая диаграмма представлена на рисунке 33 с кодом в листинге 11.

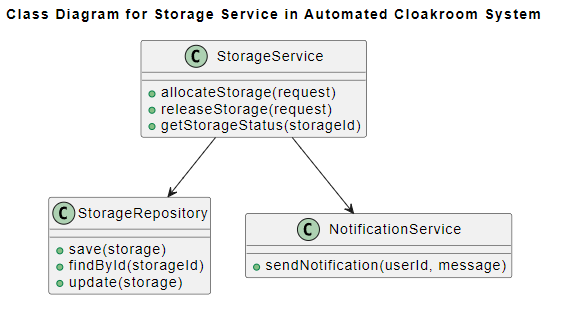


Рисунок 33 – Диаграмма уровня Code

Листинг 11 – plantUML код для диаграммы уровня Code

@startuml

title Class Diagram for Storage Service in Automated Cloakroom System

class StorageService {

+allocateStorage(request)

+releaseStorage(request)

+getStorageStatus(storageId)

}

class StorageRepository {

+save(storage)

+findById(storageId)

+update(storage)

}

class NotificationService {

+sendNotification(userId, message)

}

StorageService --> StorageRepository

StorageService --> NotificationService

@enduml

# Практическая работа №12

Задание:

* Освоить синтаксис и возможности Mermaid для создания различных типов диаграмм в документах Markdown.
* Научиться интегрировать диаграммы Mermaid в Markdown-документы.
* Применить полученные знания для визуализации структур и процессов в рамках своей индивидуальной темы проекта, используя новые типы диаграмм.

**Выполнения работы №12**

Проект был дополнен тремя диаграммами Mind Map, User Journey Diagram, QuadrantChart. Диаграмма **Mind Map** отображает структуру системы автоматизированного гардероба с разделением на роли пользователей, архитектуру и функции. **User Journey Diagram** показывает путь пользователя через этапы взаимодействия с системой, такие как хранение и выдача вещей, включая уведомления. **Quadrant Chart** иллюстрирует приоритеты задач разработки по важности и сложности, выделяя ключевые и второстепенные направления.

Итоговые диаграммы представлены на рисунках 34 - 36, а код к ним находится в листингах 12 – 14.



Рисунок 34 – Диаграмма Mind Map

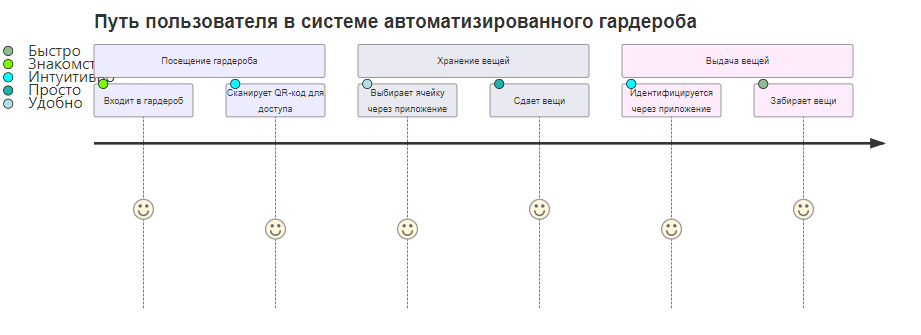


Рисунок 35 – Диаграмма User Journey



Рисунок 36 – Диаграмма Quadrant Chart

Листинг 12 – mermaid код для Диаграммы Mind Map

mindmap

root((Система автоматизированного гардероба))

Клиенты

Пользователи

: Сдают вещи

: Забирают вещи

Администраторы

: Управляют ячейками

: Настраивают систему

Архитектура

Веб-приложение

: React

: Интуитивный интерфейс

API-сервер

: Node.js

: Обработка запросов

База данных

: PostgreSQL

: Хранение данных

Функции

Хранение вещей

: Резервирование ячейки

: Контроль доступности

Выдача вещей

: Аутентификация

: Уведомления

Управление

: Администрирование системы

: Мониторинг

Листинг 13 – mermaid код для диаграммы User Journey

journey

title Путь пользователя в системе автоматизированного гардероба

section Посещение гардероба

Входит в гардероб: 5: Знакомство

Сканирует QR-код для доступа: 4: Интуитивно

section Хранение вещей

Выбирает ячейку через приложение: 4: Удобно

Сдает вещи: 5: Просто

section Выдача вещей

Идентифицируется через приложение: 4: Интуитивно

Забирает вещи: 5: Быстро

Листинг 14 – mermaid код для диаграммы Quadrant Chart

quadrantChart

title "Приоритеты задач для автоматизированного гардероба"

x-axis "Низкая важность" --> "Высокая важность"

y-axis "Простота реализации" --> "Сложность реализации"

quadrant-1 "Выполнить в первую очередь"

quadrant-2 "Стратегически важные"

"Настройка базы данных": [0.8, 0.3]

"Реализация API": [0.7, 0.4]

"Система уведомлений": [0.6, 0.7]

"Модули безопасности": [0.9, 0.8]

"Интерфейс администратора": [0.4, 0.5]

"Интеграция IoT": [0.3, 0.9]

"Оптимизация производительности": [0.5, 0.8]

Для демонстрации работы в рамках markdown бы использован Obsidian, результат представлен на рисунке 37.

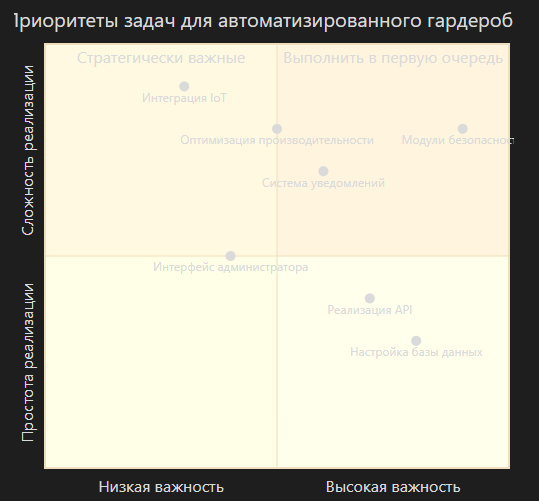


Рисунок 37 – Диаграмма Quadrant Chart в Obsidian

**Вывод:** Применение данных паттернов позволит существенно упростить архитектуру системы, обеспечить её гибкость и масштабируемость, а также снизить риски при интеграции с внешними сервисами и разработке новых функциональных компонентов. В результате выполнения всех практических работ была спроектирована система для проекта автоматизированного гардероба, ядром которой станет микросервисная архитектура с использованием различных паттернов, позволяющих для облегчения функционирования, поддерживания и обновления.

# Список литературы

1. Гвоздева Т. В., Баллод Б. А.. Проектирование информационных систем. Стандартизация [Электронный ресурс]:учебное пособие. - Санкт-Петербург: Лань, 2019. - 252 с. – Режим доступа: https://e.lanbook.com/book/115515
2. Рочев К. В.. Информационные технологии. Анализ и проектирование информационных систем [Электронный ресурс]:учебное пособие. - Санкт-Петербург: Лань, 2019. - 128 с. – Режим доступа: https://e.lanbook.com/book/122181
3. Вейцман В. М.. Проектирование информационных систем [Электронный ресурс]:учебное пособие. - Санкт- Петербург: Лань, 2019. - 316 с. – Режим доступа: https://e.lanbook.com/book/122172
4. Остроух А. В., Суркова Н. Е.. Проектирование информационных систем [Электронный ресурс]:монография. - Санкт-Петербург: Лань, 2019. - 164 с. – Режим доступа: https://e.lanbook.com/book/118650
5. Информационно-правовой портал ГАРАНТ http:// www.garant.ru
6. Консультант Плюс http:// www.consultant.ru
7. Научная электронная библиотека eLIBRARY.ru
8. Российский технологический журнал
9. Тестирование и контроль программных систем: сайт. – URL: https://xreferat.com/33/2759-1-sushnost-i-osobennosti-ispol-zovaniya-instrumental-nogo-programmnogo-obespecheniya.html (дата обращения 15.03.2020). – Текст: электронный.
10. Контроль и корректировка кода: сайт. – URL: https://studfile.net/preview/2790134/page:3/ (дата обращения 15.03.2020). – Текст: электронный.
11. Object Constraints Language: сайт. – URL: https://ami.nstu.ru/~vms/lecture/lecture12/lecture12.htm (дата обращения 15.03.2020). – Текст: электронный.
12. Эксплуатация и сопровождение проекта: сайт. – URL: http://kgau.ru/istiki/umk/pis/l7.htm (дата обращения 15.03.2020). – Текст: электронный.
13. Типовые модели систем: сайт. – URL: http://kgau.ru/istiki/umk/pis/l37.htm (дата обращения 15.03.2020). – Текст: электронный.
14. Проектирование и разработка Java приложений и систем: сайт. – URL: https://dic.academic.ru/dic.nsf/ruwiki/608820 (дата обращения 15.03.2020). – Текст: электронный.
15. Современные CASE средства проектирования систем: сайт. – URL: http://window.edu.ru/resource/616/73616/files/kulyabov-korolkova\_formal-methods.pdf (дата обращения 15.03.2020). – Текст: электронный.
16. Реляционные СУБД: сайт. – URL: https://compress.ru/article.aspx?id=10082 (дата обращения 15.03.2020). – Текст: электронный.
17. Проектирование ER-диаграмм: сайт. – URL: http://citforum.ru/cfin/prcorpsys/infsistpr\_09.shtml (дата обращения 15.03.2020). – Текст: электронный.
18. Стандарты разработки: сайт. – URL: http://venec.ulstu.ru/lib/disk/2017/460.pdf (дата обращения 15.03.2020). – Текст: электронный.
19. Архитектура ЭВМ и систем: сайт. – URL: https://arxiv.org/ftp/arxiv/papers/1802/1802.06769.pdf (дата обращения 15.03.2020). – Текст: электронный.