СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данный отчёт представляет собой результат выполнения производственной практики в компании «Andersen». Основной целью этой практики было приобретение профессиональных навыков, соответствующих специализации, а также получение опыта работы внутри компании. Помимо этого, практика также предоставила возможность закрепить, расширить и систематизировать знания, полученные в ходе обучения по специальным дисциплинам.

В процессе практики необходимо было ознакомиться с организационной структурой предприятия, изучить функциональные обязанности различных подразделений, оценить степень автоматизации бизнес-процессов компании, а также ознакомиться с предоставляемыми компанией услугами.

Кроме того, одной из важных задач практики было разработать специализированное программное приложение, которое было создано с учетом особенностей предметной области и результатов анализа теоретического материала.

В рамках практики было спроектировано и разработано микросервисное приложение. Данное приложение было разработано для изучения микросервисной архитектуры. Оно позволяет пополнять счет.

Таким образом, прохождение производственной практики в компании «Andersen» позволило не только приобрести ценный опыт, но и внести вклад в развитие программного обеспечения, удовлетворяющего потребностям современного бизнеса и повседневной жизни.

1 ХАРАКТЕРИСТИКИ ПРЕДПРИЯТИЯ

1.1 История развития, общие сведения о предприятии

Andersen — международная IT-компания, которая специализируется на разработке программного обеспечения, предоставлении консультационных услуг и цифровой трансформации. Компания Andersen была основана в 2007 году и в настоящее время имеет офисы и представительства в различных странах. История развития компании Andersen связана с ее стремлением к постоянному инновационному росту и внедрению новых технологий. Компания активно занималась разработкой программного обеспечения, предоставлением услуг по тестированию и контролю качества, а также консалтинговыми услугами.

Со временем, Andersen стала шире внедрять искусственный интеллект, автоматизацию бизнес-процессов, машинное обучение и другие передовые технологии в свои проекты. Компания активно работает с клиентами из различных отраслей, включая финансы, здравоохранение, розничную торговлю, производство и другие, помогая им добиться цифровой трансформации и достичь конкурентного преимущества на рынке.

Сегодня Andersen является одной из ведущих IT-компаний, предоставляющих техническую экспертизу и консультирование в области разработки ПО и цифровой трансформации. Компания постоянно развивается, и их целью является опережающее предоставление инновационных решений для своих клиентов.

1.2 Организационная структура предприятия

В настоящее время штат постоянно растет и на данный момент насчитывает более 3500 сотрудников, а каждый из новых специалистов принес в команду свой уникальный опыт и наработки. Около 70% сотрудников Andersen заняты непосредственно разработкой, а еще 15% - тестированием. Работа компании построена в полном соответствии с методологией Agile и практикой непрерывной интеграции. Это означает, что каждый член команды всегда знает свои обязанности, они не пересекаются с обязанностями других сотрудников и не мешают их выполнению.

Благодаря собственному отделу тестирования обеспечивается неизменно высокое качество разрабатываемых решений. Кроме того, предлагается тестирование сторонних проектов как отдельная услуга.

В активе Andersen есть проекты различных типов: разработка приложений для предприятий госсектора, внедрение электронных платежных систем, ряд приложений для мобильных устройств и многие другие.

Основные задачи компании:

– Разработка приложений от настольных до веб, работа с торговой и производственной сферами, а также со сферой рекламы и развлечений. Интегрирование разнородного ПО. Сведение различных информационных систем в единую платформу.

– Проведение в плановом и экстренном режиме проверки работоспособности web-ресурсов, приложений и электронных игр, где Andersen выступает как независимый эксперт.

– Создание стильных, красочных, адаптивных и мобильных решений с удобным интерфейсом и уникальным дизайном.

– Настройка и доработка типовых конфигураций 1С. Разработка новых отчетов и обработок 1С. Разработка конфигураций с нуля под заказ.[1]

2 АНАЛИЗ ФУНКЦИОНИРОВАНИЯ СТРУКТУРНЫХ ПОДРАЗДЕЛЕНИЙ

2.1 Анализ функций предприятия

ООО «Andersen» основывает свою деятельность на богатом опыте и глубоких знаниях, создавая инновационные технологии и предоставляя клиентам решения для глобальных бизнес-задач. Достижения и экспертиза охватывают различные области, включая:

– внедрение ERP, PLM, CRM, SCM решений и систем аналитики, стратегического планирования и бюджетирования в ряде отраслей;

– анализ инфраструктуры и информационных ресурсов, проектирование и реинжиниринг бизнес-процессов, управление проектами модернизации и развития информационных систем

– разработка по заказам производителей программного обеспечения для систем корпоративного планирования (ERP), управления жизненным циклом изделий (PLM), корпоративных информационных порталов (EIP), систем управления отношениями с клиентами (CRM);

– разработка приложений, соответствующих требованиям новейших сервис-ориентированных архитектур (SOA - service oriented architecture);

– создание и развертывание электронных систем управления закупками и сбытом;

– построение порталов крупных предприятий и холдингов с развитыми средствами анализа данных и управления знаниями;

– интеграция приложений в распределенных системах (в том числе насчитывающих сотни производственных площадок, сотни унаследованных приложений и десятки ERP-систем), проектирование, консолидация и настройка корпоративных справочников и каталогов;

Высококвалифицированные сотрудники и эффективные процессы разработки позволяют предоставлять клиентам наилучшие IT-решения, объединяя в себе преимущества индивидуальных заказных проектов и масштабируемых продуктов.

2.2 Анализ функций и задач IT-отдела

Одно из основных направлений деятельности компании – разработка программного обеспечения на заказ (аутсорсинг). Типичный заказчик Andersen обращается к в компанию, когда для его задач не существует достаточно удобного и функционально соответствующего готового универсального решения. Команда Andersen полностью берет процесс под свой контроль. Создание решения

начинается с четкого формулирования требований и анализа потребностей клиента. Далее обязательно следует создание визуальных моделей и предоставление их заказчику на согласование. Специалисты Andersen всегда активно участвуют в процессе согласования и дают клиенту экспертные рекомендации.

Разработка проекта ведется с использованием безопасной и гибкой методики Agile. При планировании разработки используется декомпозиция с оптимальным размером фрагментов. Методология непрерывной интеграции позволяет управлять рисками, что существенно ускоряет выход продукта. Эффективные коммуникации и отслеживание ошибок обеспечиваются использованием системы JIRA, а хранение всех промежуточных версий кода – применением Git. Для различных уровней (разработка, продакшн, поддержка, тестирование) применяются отдельные среды, что исключает возникновение проблем и обеспечивает поступление заказчику исключительно рабочих сборок ПО.

Для тестирования разрабатываемого программного обеспечения в Andersen выделен специальный отдел, что позволяет обеспечить не только полное соответствие разрабатываемого решения требованиям технического задания, но и снизить риски, связанные с его поведением в нештатных ситуациях.

3 НОРМАТИВНАЯ ДОКУМЕНТАЦИЯ

3.1 Содержание должностных инструкций

Должностная инструкция инженера-программиста. Общие положения:

1. Вакансия ведущего инженера-программиста оформляется приказом руководителя при приеме на работу и также уведомлением при увольнении.

2. Для занятия должности ведущего инженера-программиста требуется наличие высшего образования в соответствующей области квалификации.

3. В своей деятельности инженер-программист руководствуется:

– нормативными документами по вопросам выполняемой работы;

– уставом организации;

– правилами трудового распорядка;

– приказами и распоряжениями руководителя организации (непосредственного руководителя).

4. Ведущий инженер-программист должен знать:

– руководящие и нормативные материалы, регламентирующие методы разработки алгоритмов, программ и использования вычислительной техники при обработке информации;

– основные принципы структурного и объектно-ориентированного программирования;

– виды программного обеспечения;

– стандарты программной документации;

– основные методы, средства и методологии разработки программного обеспечения;

– стандартные алгоритмы;

– языки программирования;

– технологию автоматизированной обработки информации;

– виды и порядок оформления технической документации;

– стандарты качества программного обеспечения;

– основные общепринятые технологии в области разработки программных продуктов;

– технико-эксплуатационные характеристики, конструктивные особенности, назначение и режимы работы оборудования, правила его технической эксплуатации;

– основы организации труда и управления;

– основы законодательства о труде;

– правила и нормы охраны труда и пожарной безопасности.

5. Во время отсутствия ведущего инженера-программиста его обязанности выполняет в установленном порядке назначаемый заместитель, несущий полную ответственность за надлежащее исполнение возложенных на него обязанностей.

Должностные обязанности. Для выполнения возложенных на него функций ведущий инженер-программист обязан:

– На основе анализа математических моделей и алгоритмов решения научных, прикладных, экономических и других задач разрабатывать программы, обеспечивающие возможность выполнения средствами вычислительной техники алгоритма и поставленной задачи.

– Участвовать в выборе языка программирования для описания алгоритмов и структур данных.

– Разрабатывать технологию решения задачи на всех этапах.

– Определять информацию, подлежащую обработке средствами вычислительной техники, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и вывода, методы ее контроля.

– Определять объем и содержание данных контрольных примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению.

– Осуществлять запуск отлаженных программ и ввод исходных данных, определяемых условиями поставленных задач.

– Определять возможность использования готовых программных продуктов.

– Разрабатывать, отлаживать, анализировать и оптимизировать программный код на основе готовых спецификаций.

– Интегрировать программные компоненты.

– Проводить проверку программ на основе логического анализа.

– Проводить отладку разработанных программ, корректировать их в процессе стабилизации и сопровождения.

– Выполнять работу по унификации процессов разработки программы.

– Разрабатывать инструкции по работе с программами, оформлять необходимую техническую документацию.

– Разрабатывать и внедрять методы автоматизации программирования.

– Участвовать в сопровождении программного обеспечения.

– Оказывать помощь специалистам на различных стадиях разработки программного обеспечения при сборе и документировании требований пользователя, в разработке спецификации.

– Вести и представлять установленную отчетность.

– Своевременно и точно исполнять производственные приказы, задания, указания и распоряжения руководства.

– Соблюдать установленные на предприятии Правила внутреннего трудового распорядка.

– Оказывать содействие и сотрудничать с нанимателем в деле обеспечения здоровых и безопасных условий труда, немедленно сообщать непосредственному руководителю о каждом случае производственного травматизма и профессионального заболевания, а также о чрезвычайных ситуациях, которые

создают угрозу здоровью и жизни для него и окружающих, обнаруженных недостатках и нарушениях охраны.

– Принимать необходимые меры по ограничению развития аварийной ситуации и ее ликвидации.

Права. Ведущий инженер-программист имеет право:

– Знакомиться с проектами решений руководства организации, касающимися его деятельности.

– Вносить на рассмотрение руководства по совершенствованию информационной системы организации в целом.

– Получать от руководителей структурных подразделений, специалистов информацию и документы, необходимые для выполнения своих должностных обязанностей.

– Привлекать специалистов организации для решения возложенных на него обязанностей (если это предусмотрено положениями о структурных подразделениях, если нет – с разрешения руководителя организации).

– Требовать от руководства организации оказания содействия в исполнении своих должностных обязанностей и прав.

– Принимать участие в обсуждении вопросов охраны труда, выносимых на рассмотрение собраний (конференций) трудового коллектива (профсоюзной организации)

Оценка работы и ответственность. Работу ведущего инженера-программиста оценивает непосредственный руководитель (иное должностное лицо) Ведущий инженер-программист несёт ответственность:

– За неисполнение (ненадлежащее поведение) своих должностных обязанностей, предусмотренных настоящей должностной инструкцией, – в пределах, определенных действующим трудовым законодательством РБ.

– За совершенные в процессе осуществления своей деятельности правонарушения – в пределах, определенных действующим административным, уголовным и гражданским законодательством РБ.

– За причинение материального ущерба – в пределах, определенных действующим трудовым, уголовным и гражданским законодательством РБ.

– За несоблюдение правил и норм охраны труда, техники безопасности, производственной санитарии и противопожарной защиты – в соответствии с требованиями нормативных правовых актов РБ и локальных актов.

3.2 Модели жизненного цикла программного обеспечения

Жизненный цикл программного обеспечения (ПО) представляет собой последовательность событий, которые происходят при создании и использовании программных продуктов. Он может быть представлен в виде различных моделей.

Эти модели включают инженерный подход, учет спецификации задачи и современные технологии быстрой разработки.

В прежние времена, программные приложения были едиными и для их разработки применялся каскадный метод. Этот метод предполагал разделение разработки на этапы, с переходом к следующему этапу только после завершения текущего. Каждый этап завершался созданием полного комплекта документации.

Стандарт ISO 12207 является международным стандартом, который полно описывает процессы жизненного цикла программного обеспечения, технологию разработки и обеспечения качества сложных программных средств. Жизненный цикл программного обеспечения включает этапы от подготовки технического задания до завершения эксплуатации.

Процессы разработки программного обеспечения начинаются с инициации проекта, анализа концепции и требований, и включают создание плана проекта, выбор средств разработки и документирование всех этапов.

Процессы сопровождения программного обеспечения включают анализ сообщений об ошибках, предложений по модификации программы, их оценку и тестирование изменений.

Процессы документирования включают планирование и создание документации, а также обеспечение качества программного обеспечения.

Верификация и аттестация обеспечивают соответствие программного продукта требованиям и гарантируют его безопасное и надежное использование.

Таким образом, жизненный цикл программного обеспечения включает в себя множество процессов, начиная с разработки и заканчивая сопровождением и обеспечением качества, что позволяет создавать и поддерживать сложные программные продукты.

4 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ИСПОЛЬЗУЕМОЕ В ОРГАНИЗАЦИИ

4.1 Среды разработки и языки программирования, используемые в организации

Для разработки программного обеспечения в отделе используются такие среды разработки как: Microsoft Visual Studio, 1C-Bitrix, IntelliJ IDEA, VS Code, PhpStorm.

Средства и технологии, которые используются для разработки ПО:

1. Языки программирования (C, C++, C#, PHP, Java, JavaScript, Python, Ruby, Scala, Golang)
2. Языки разметки (HTML, XML)
3. Фреймворки (Angular, React, Vue, ASP.NET Core, Unity, Xamarin, iOS, Android)
4. СУБД (MS SQL, MySQL, LINQ to SQL)

4.2 Операционные системы, используемые в организации

Компания Andersen использует операционные системы семейства Windows, Mac OS и Linux, на которые оформлены соответствующие лицензии.

4.3 Требования, предъявляемые к разрабатываемому программному продукту

К главным требованиям, предъявляемым к разрабатываемому продукту, относятся:

– Правильность – функционирование в соответствии с техническим заданием.

– Универсальность – обеспечение правильной работы при любых допустимых данных и защиты от неправильных данных.

– Надежность (помехозащищенность) – обеспечение полной повторяемости результатов, т. е. обеспечение их правильности при наличии различного рода сбоев.

– Защищенность – обеспечение конфиденциальности информации.

– Аппаратная совместимость – возможность совместного функционирования с некоторым оборудованием.

5 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

5.1 Постановка задачи

За время прохождения практики необходимо было разработать микросервисное приложение, а именно приложение для пополнение счетов пользователя.

5.2 Выбор и обоснование выбора языка программирования

Согласно требованиям к программному средству, данная разрабатываемая программа должна представлять только правильный функционал работы. Не обеспечивая удобный интерфейс для пользователя.

Для создания программы был использован язык Java.

Java – широко используемый язык программирования для написания интернет-приложений. Язык Java широко использовался на протяжении более двух десятилетий. Миллионы приложений Java используются и сегодня. Java – это многоплатформенный, объектно-ориентированный и сетецентрический язык, который сам по себе может использоваться как платформа. Это быстрый, безопасный и надежный язык программирования для всего: от мобильных приложений и корпоративного ПО до приложений для работы с большими данными и серверных технологий[2].

5.3 Выбор средств разработки программы

В качестве интегрированной среды разработки была выбрана Intellij IDEA. IntelliJ IDEA — это IDE, интегрированная среда разработки (комплекс программных средств, который используется для написания, исполнения, отладки и оптимизации кода) для Java, JavaScript, Python и других языков программирования от компании JetBrains. Отличается обширным набором инструментов для рефакторинга (перепроектирования) и оптимизации кода[3].

5.4 Проектирование функциональной структуры

Проектирование функциональной структуры программы включает в себя определение функций и их взаимодействие для обеспечения работы приложения,

а также требует тщательного анализа потребностей пользователей и бизнес-логики для создания эффективного и интуитивно понятного интерфейса.

Вот функциональная структура для такого приложения:

- Пополнение счета пользователя;
- Отправка уведомления посредством RabbitMQ;
- Получение уведомлений для других сервисов;
- Единый вход в программу для всех сервисов;
- Добавление docker container для правильного старта программы;
- Регистрация сервисов в Eureka.

5.5 Реализация и тестирование программы

Для реализации приложения потребовались Java, фреймворк Spring Boot. Код файла docker-compose.yml, для правильного порядка запуска сервисов представлен в листинге 5.1.

Листинг 5.1 – Код файла docker-compose.yml

```
version: '3.5'
services:
  config-service:
    container_name: config-service
    build: config-service
    ports:
      - 8001:8001

  registry:
    container_name: registry
    restart: always
    build: registry
    ports:
      - 8761:8761
    depends_on:
      - config-service

  gateway:
    container_name: gateway
    restart: always
    build: gateway
    ports:
      - 8989:8989
    depends_on:
      - config-service
      - registry
```

```
account-service:
  container_name: account-service
  restart: on-failure
  build: account-service
  ports:
    - 8081:8081
  depends_on:
    - config-service
    - registry

bill-service:
  restart: on-failure
  container_name: bill-service
  build: bill-service
  ports:
    - 8082:8082
  depends_on:
    - config-service
    - registry

deposit-service:
  restart: on-failure
  container_name: deposit-service
  build: deposit-service
  ports:
    - 9090:9090
  depends_on:
    - config-service
    - registry

notification-service:
  restart: on-failure
  container_name: notification-service
  build: notification-service
  depends_on:
    - config-service
    - registry

rabbitmq:
  image: rabbitmq:3-management
  hostname: rabbitmq
  labels:
    NAME: "rabbitmq"
  ports:
    - 5672:5672
    - 15672:15672
  depends_on:
    - config-service
```

```
      - registry
      - account-service
```

Для пополнения счёта и отправки уведомления о данном события использовался разработанный класс DepositService. Код файла DepositService.java представлен в листинге 5.2.

Листинг 5.2 – Код файла DepositService.java

```
package com.javastart.deposit.service;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.javastart.deposit.controller.dto.DepositResponseDTO;
import com.javastart.deposit.entity.Deposit;
import com.javastart.deposit.exception.DepositServiceException;
import com.javastart.deposit.repository.DepositRepository;
import com.javastart.deposit.rest.*;
import lombok.AllArgsConstructor;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.time.OffsetDateTime;

@Service
public class DepositService {

    private static final String TOPIC_EXCHANGE_DEPOSIT =
"js.deposit.notify.exchange";
    private static final String ROUTING_KEY_DEPOSIT =
"js.key.deposit";

    private final DepositRepository depositRepository;

    private final AccountServiceClient accountServiceClient;

    private final BillServiceClient billServiceClient;

    private final RabbitTemplate rabbitTemplate;

    @Autowired
    public DepositService(DepositRepository depositRepository,
AccountServiceClient accountServiceClient, BillServiceClient
billServiceClient, RabbitTemplate rabbitTemplate) {
        this.depositRepository = depositRepository;
        this.accountServiceClient = accountServiceClient;
        this.billServiceClient = billServiceClient;
        this.rabbitTemplate = rabbitTemplate;
    }
```

```
    public DepositResponseDTO deposit(Long accountId, Long billId,
BigDecimal amount) {
        if(accountId == null && billId == null){
            throw new DepositServiceException("Account id and bill
id cannot be null");
        }

        if(billId != null){
            BillResponseDTO billResponseDTO =
billServiceClient.getBillById(billId);
            BillRequestDTO billRequestDTO =
createBillRequest(amount, billResponseDTO);

            billServiceClient.update(billId, billRequestDTO);

            AccountResponseDTO accountResponseDTO =
accountServiceClient.getAccountById(billResponseDTO.getAccountId())
;
            depositRepository.save(new Deposit(amount, billId,
OffsetDateTime.now(), accountResponseDTO.getEmail()));

            return createResponse(amount, accountResponseDTO);
        }
        BillResponseDTO defaultBill = getDefaultBill(accountId);
        BillRequestDTO billRequestDTO = createBillRequest(amount,
defaultBill);
        billServiceClient.update(defaultBill.getBillId(),
billRequestDTO);

        AccountResponseDTO accountResponseDTO =
accountServiceClient.getAccountById(accountId);
        depositRepository.save(new Deposit(amount,
defaultBill.getBillId(), OffsetDateTime.now(),
accountResponseDTO.getEmail()));

        return createResponse(amount, accountResponseDTO);

    }

    private DepositResponseDTO createResponse(BigDecimal amount,
AccountResponseDTO accountResponseDTO) {
        DepositResponseDTO depositResponseDTO = new
DepositResponseDTO(amount, accountResponseDTO.getEmail());

        ObjectMapper objectMapper = new ObjectMapper();
        try {
            rabbitTemplate
                    .convertAndSend(TOPIC_EXCHANGE_DEPOSIT,
ROUTING_KEY_DEPOSIT,
```

Продолжение листинга 5.2

```
objectMapper.writeValueAsString(depositResponseDTO));
        } catch (JsonProcessingException e) {
            e.printStackTrace();
            throw new DepositServiceException("Can't send a message
to RabbitMQ");

        }
        return                                          depositResponseDTO;
    }

    private   static   BillRequestDTO   createBillRequest(BigDecimal
amount,          BillResponseDTO          billResponseDTO)          {
        BillRequestDTO   billRequestDTO   =   new   BillRequestDTO();

billRequestDTO.setAccountId(billResponseDTO.getAccountId());

billRequestDTO.setCreationDate(billResponseDTO.getCreationDate());

billRequestDTO.setIsDefault(billResponseDTO.getIsDefault());

billRequestDTO.setOverdraftEnabled(billResponseDTO.getOverdraftEnab
led());

billRequestDTO.setAmount(billResponseDTO.getAmount().add(amount));
        return                                              billRequestDTO;
    }

    private    BillResponseDTO    getDefaultBill(Long    accountId){
        return      billServiceClient.getBillsByAccountId(accountId)
                .stream()
                .filter(BillResponseDTO::getIsDefault)
                .findAny()
                .orElseThrow(()                  ->                  new
DepositServiceException("Unable to find default bill by account id "
+                                                       accountId));
    }
}
```

Для работы RabbitMQ был создан класс конфигурации RabbitMQConfig. Код файла RabbitMQConfig.java представлен в листинге 5.3.

Листинг 5.3 – Реализация веб-сайта

```
package com.javastart.notification.config;

import org.springframework.amqp.core.*;
import org.springframework.amqp.rabbit.annotation.EnableRabbit;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

Продолжение листинга 5.3

```
@Configuration
@EnableRabbit
public class RabbitMQConfig {

    public static final String QUEUE_DEPOSIT = "js.deposit.notify";

    private static final String TOPIC_EXCHANGE_DEPOSIT =
"js.deposit.notify.exchange";
    private static final String ROUTING_KEY_DEPOSIT =
"js.key.deposit";

    @Autowired
    private AmqpAdmin amqpAdmin;

    @Bean
    public TopicExchange depositExchange() {
        return new TopicExchange(TOPIC_EXCHANGE_DEPOSIT);
    }

    @Bean
    public Queue depositQueue() {
        return new Queue(QUEUE_DEPOSIT);
    }

    @Bean
    public Binding depositBinding() {
        return BindingBuilder
                .bind(depositQueue())
                .to(depositExchange())
                .with(ROUTING_KEY_DEPOSIT);
    }
}
```

В ходе технологической практики было разработано приложение согласно требованиям, изложенным в главе 5.1. Однако, для того чтобы разработку можно было считать завершенной, необходимо провести тестирование. Тестирование программного обеспечения – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. Главная цель проведения тестирование – проверка выполнения программой поставленных задач, способность решать задачи пользователей с необходимой точностью при использовании в заданном контексте, способность предоставлять определенные результаты в рамках ожидаемых затрат ресурсов и способность приносить удовлетворение пользователям при использовании в заданном контексте. Для проверки выполнения программой всех вышеуказанных требований, были разработаны unit тесты для проверки корректной работы программы, среди которых можно выделить следующие:

– Создание депозита, в контроллере программы
– Создание депозита на уровне сервиса
– Проверка возникновения исключения, при некорректной передачи параметров в метод для создания депозита.

Для проверки данных ситуаций, было разработано 2 класса. DepositServiceTest, в котором происходит проверка создание депозита и проверка возникновения исключения, и DepositControllerTest, в котором проверяется создание депозита в контроллере.

Приведем листинг данных классов.

Листинг 5.4 – код файла DepositServiceTest.java

```
package com.javastart.deposit.service;

import com.javastart.deposit.controller.dto.DepositResponseDTO;
import com.javastart.deposit.exception.DepositServiceException;
import com.javastart.deposit.repository.DepositRepository;
import com.javastart.deposit.rest.AccountResponseDTO;
import com.javastart.deposit.rest.AccountServiceClient;
import com.javastart.deposit.rest.BillResponseDTO;
import com.javastart.deposit.rest.BillServiceClient;
import org.assertj.core.api.Assertions;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.MockitoJUnitRunner;
import org.springframework.amqp.rabbit.core.RabbitTemplate;

import java.math.BigDecimal;
import java.time.OffsetDateTime;
import java.util.Arrays;

@RunWith(MockitoJUnitRunner.class)
public class DepositServiceTest {

    @Mock
    private DepositRepository depositRepository;

    @Mock
    private AccountServiceClient accountServiceClient;

    @Mock
    private BillServiceClient billServiceClient;

    @Mock
    private RabbitTemplate rabbitTemplate;

    @InjectMocks
    private DepositService depositService;
```

```
    @Test
    public void depositServiceTest_withBillId() {
        BillResponseDTO billResponseDTO = createBillResponseDTO();

Mockito.when(billServiceClient.getBillById(Mockito.anyLong())).then
Return(billResponseDTO);

Mockito.when(accountServiceClient.getAccountById(Mockito.anyLong())
).thenReturn(createAccountResponseDTO());
        DepositResponseDTO deposit = depositService.deposit(null,
1L, BigDecimal.valueOf(1000));

Assertions.assertThat(deposit.getMail()).isEqualTo("nothing@nothing
.com");
    }

    @Test(expected = DepositServiceException.class)
    public void depositServiceTest_exception() {
        depositService.deposit(null, null,
BigDecimal.valueOf(1000));
    }

    private AccountResponseDTO createAccountResponseDTO(){
        AccountResponseDTO accountResponseDTO = new
AccountResponseDTO();
        accountResponseDTO.setAccountId(1L);
        accountResponseDTO.setBills(Arrays.asList(1L, 2L, 3L));
        accountResponseDTO.setCreationDate(OffsetDateTime.now());
        accountResponseDTO.setEmail("nothing@nothing.com");
        accountResponseDTO.setName("nothing");
        accountResponseDTO.setPhone("+123456");
        return accountResponseDTO;
    }

    private BillResponseDTO createBillResponseDTO() {
        BillResponseDTO billResponseDTO = new BillResponseDTO();
        billResponseDTO.setAccountId(1L);
        billResponseDTO.setAmount(BigDecimal.valueOf(1000));
        billResponseDTO.setBillId(1L);
        billResponseDTO.setCreationDate(OffsetDateTime.now());
        billResponseDTO.setIsDefault(true);
        billResponseDTO.setOverdraftEnabled(true);
        return billResponseDTO;
    }
}
```

Листинг 5.5 – код файла DepositControllerTest.java

```
package com.javastart.deposit.controller;
```

```
import com.fasterxml.jackson.databind.ObjectMapper;
import com.javastart.deposit.DepositApplication;
import com.javastart.deposit.config.SpringH2DatabaseConfig;
import com.javastart.deposit.controller.dto.DepositResponseDTO;
import com.javastart.deposit.entity.Deposit;
import com.javastart.deposit.repository.DepositRepository;
import com.javastart.deposit.rest.AccountResponseDTO;
import com.javastart.deposit.rest.AccountServiceClient;
import com.javastart.deposit.rest.BillResponseDTO;
import com.javastart.deposit.rest.BillServiceClient;
import org.assertj.core.api.Assertions;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import java.math.BigDecimal;
import java.time.OffsetDateTime;
import java.util.Arrays;
import java.util.List;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders
.post;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.s
tatus;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = {DepositApplication.class,
SpringH2DatabaseConfig.class})
public class DepositControllerTest {

    private MockMvc mockMvc;

    @Autowired
    private WebApplicationContext context;
```

```
    @Autowired
    private DepositRepository depositRepository;

    @MockBean
    private BillServiceClient billServiceClient;

    @MockBean
    private AccountServiceClient accountServiceClient;

    @MockBean
    private RabbitTemplate rabbitTemplate;

    @Before
    public void setup() {
        this.mockMvc =
MockMvcBuilders.webAppContextSetup(context).build();
    }

    private static final String REQUEST = "{\n" +
            "    \"billId\": 1,\n" +
            "    \"amount\": 3000\n" +
            "}";

    @Test
    public void createDeposit() throws Exception {
        BillResponseDTO billResponseDTO = createBillResponseDTO();

Mockito.when(billServiceClient.getBillById(Mockito.anyLong())).then
Return(billResponseDTO);

Mockito.when(accountServiceClient.getAccountById(Mockito.anyLong())
).thenReturn(createAccountResponseDTO());
        MvcResult mvcResult = mockMvc.perform(post("/deposits")
                        .content(REQUEST)
                        .contentType(MediaType.APPLICATION_JSON)

.accept(MediaType.APPLICATION_JSON)).andExpect(status()
                        .isOk())
                .andReturn();
        String body = mvcResult.getResponse().getContentAsString();
        List<Deposit> deposits =
depositRepository.findDepositsByEmail("nothing@nothing.com");
        ObjectMapper objectMapper = new ObjectMapper();
        DepositResponseDTO depositResponseDTO =
objectMapper.readValue(body, DepositResponseDTO.class);


Assertions.assertThat(depositResponseDTO.getMail()).isEqualTo(depos
its.get(0).getEmail());
```

```
Assertions.assertThat(depositResponseDTO.getAmount()).isEqualTo(dep
osits.get(0).getAmount());
    }
    private AccountResponseDTO createAccountResponseDTO(){
        AccountResponseDTO accountResponseDTO = new

AccountResponseDTO();
        accountResponseDTO.setAccountId(1L);
        accountResponseDTO.setBills(Arrays.asList(1L, 2L, 3L));
        accountResponseDTO.setCreationDate(OffsetDateTime.now());
        accountResponseDTO.setEmail("nothing@nothing.com");
        accountResponseDTO.setName("nothing");
        accountResponseDTO.setPhone("+123456");
        return accountResponseDTO;
    }

    private BillResponseDTO createBillResponseDTO() {
        BillResponseDTO billResponseDTO = new BillResponseDTO();
        billResponseDTO.setAccountId(1L);
        billResponseDTO.setAmount(BigDecimal.valueOf(1000));
        billResponseDTO.setBillId(1L);
        billResponseDTO.setCreationDate(OffsetDateTime.now());
        billResponseDTO.setIsDefault(true);
        billResponseDTO.setOverdraftEnabled(true);
        return billResponseDTO;
    }
}
```

Подводя итог, требуется отметить, что программное обеспечение стабильно работает. Скорость работы программы достаточно быстрая. Все команды работают без ошибок. Качество данного приложения находится на высоком уровне. Дефекты функционала не обнаружены. Таким образом, разработанная программа соответствует всем заявленным характеристикам.

ЗАКЛЮЧЕНИЕ

По итогам прохождения производственной практики была проведена оценка эффективности взаимодействия между различными подразделениями предприятия, что позволило выявить ключевые моменты для оптимизации процессов. Были изучены нормативные документации подразделения, в том числе должностная инструкция инженера-программиста.

Особое внимание было уделено интеграции микросервисов, что способствовало более глубокому пониманию принципов работы распределенных систем. Полученные знания и навыки в области разработки программного обеспечения на платформе Java, а также использование современных инструментов и технологий, таких как Spring Boot и RabbitMQ, значительно повысили уровень моей квалификации и подготовили к будущей профессиональной деятельности в сфере IT.

Технологическая (производственная) практика сыграла огромную роль в приобретении практических навыков. В ходе неё были закреплены теоретические знания по изученным дисциплинам и было получено много новых.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Электронный ресурс «JavaScript – чтение данных» [Электронный ресурс]. Режим доступа https://coderlessons.com/tutorials/veb-razrabotka/izuchite-firebase/firebase-chtenie-dannykh. Дата доступа – 12.08.2024 г.

2 Что такое Java? [Электронный ресурс] – Режим доступа: https://aws.amazon.com/ru/what-is/java/. Дата доступа: 20.08.2024 г.

3 Intellij IDEA [Электронный ресурс]. Режим доступа https://blog.skillfactory.ru/glossary/intellij-idea/. Дата доступа – 24.08.2024 г.