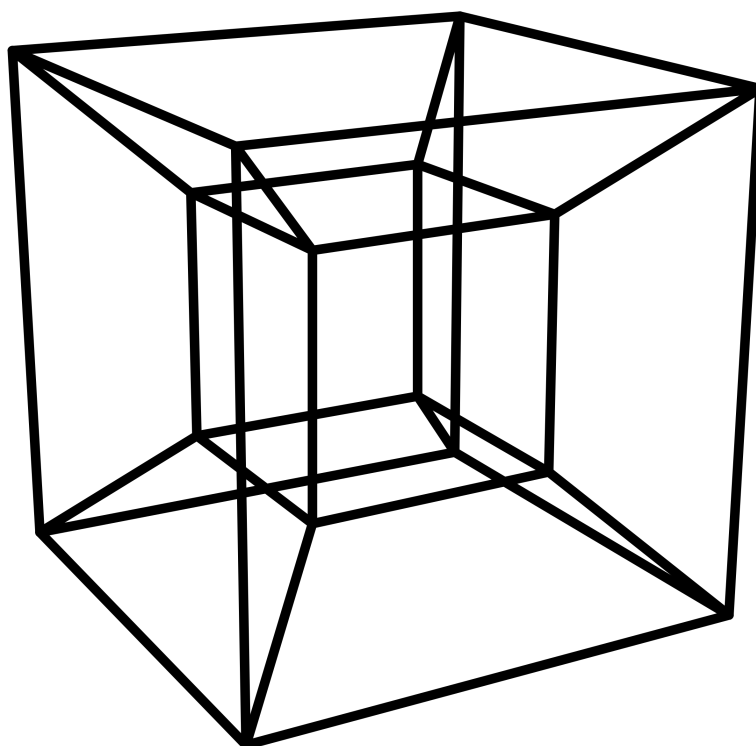


ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ
ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОТЧЕТ ПО СИСТЕМНОМУ ТЕСТИРОВАНИЮ

АГРЕГАТОР ЦИФРОВЫХ ФИНАНСОВЫХ АКТИВОВ «ТЕССЕРАКТ»

по дисциплине «Технологии разработки качественного программного обеспечения»



Выполнили студенты группы: 5130904/00104:

Почернин В. С.
Шиляев В. С.
Мурзаканов И. М.
Разукрантов В. Е.

Преподаватель:

Маслаков А. П.

Санкт-Петербург
2024

Содержание

1	Постановка задачи	2
1.1	Обязательные требования	2
1.2	Содержание отчета	2
2	Ход работы	2
2.1	Отчет о выполненной работе, использованных инструментах	2
2.1.1	Использованные инструменты	2
2.2	Тест-план со словесным описанием тестовых сценариев	2
2.2.1	AssetTest	3
2.2.2	FavoriteTest	3
2.2.3	DiversificationTest	3
2.2.4	ChangePasswordTest	4
2.2.5	LoginTest	4
2.2.6	RegisterTest	5
2.3	Отчет о прохождении тестов с результатами на сервере непрерыв- ной интеграции	5

1 Постановка задачи

На данном уровне необходимо протестировать готовый продукт по бизнес-требованиям, сформированным перед началом непосредственного проектирования программного продукта. В качестве тестовых сценариев выбираются основные сценарии использования ПО в полностью рабочем окружении. Целесообразно использование инструментов веб-тестирования или UI-тестирования, таких как selenium, puppeteer и т. д.

Минимальное количество сценариев - 10.

1.1 Обязательные требования

- 1) Предварительное формирование документа, описывающего тестовые сценарии.
- 2) Поднятие сервера непрерывной интеграции и запуск задачи системного тестирования по временному триггеру или в ручном режиме.

1.2 Содержание отчета

Отчет по интеграционному тестированию должен содержать:

- 1) Отчет о выполненной работе, использованных инструментах.
- 2) Тест-план со словесным описанием тестовых сценариев.
- 3) Отчет о прохождении тестов с результатами на сервере непрерывной интеграции.

2 Ход работы

2.1 Отчет о выполненной работе, использованных инструментах

В рамках проведенной работы, код клиентской части приложения был покрыт системными тестами.

2.1.1 Используемые инструменты

Были использованы следующие инструменты:

- **JUnit** - фреймворк для языков программирования Java и Kotlin, предназначенный для автоматического unit-тестирования. Данный фреймворк позволяет удобно создавать, организовывать и выполнять тесты, благодаря широкому набору аннотаций и встроенной поддержки в популярных IDE. Для клиентской части применялся JUnit4.
- **Compose UI Test** - это библиотека, предоставляемая Android Jetpack для тестирования пользовательского интерфейса (UI) в приложениях, использующих Jetpack Compose. Эта библиотека предоставляет набор инструментов и API для написания и выполнения автоматизированных тестов, которые проверяют корректность работы пользовательского интерфейса приложений, созданных с использованием Jetpack Compose.
- **GitHub Actions** - система непрерывной интеграции (CI), используемая нами для автоматического выполнения тестов при открытии PR или мерже в мастер ветку. Позволяет автоматизировать процесс интеграционного тестирования и не только.

2.2 Тест-план со словесным описанием тестовых сценариев

Описание тест-плана удобно разделить на несколько категорий.

Для залогиненного пользователя

2.2.1 AssetTest

В данном разделе описаны тесты, связанные с Активами.

- 1) `whenGoToAsset_thenAssetIsLoadedSuccessfully` - получение информации по выбранному активу.

Порядок действий:

1. Нажать на любой актив.
2. Проверить, что описание отображается корректно.

- 2) `whenSwipeUp_thenAllAssetsAreShown` - загрузка активов и проверка отображаемого количества.

Порядок действий:

1. Выполнять свайп вверх, пока не будет достигнут конец списка.
2. Проверить, что количество активов соответствует количеству активов в тестовой базе.

2.2.2 FavoriteTest

В данном разделе описаны тесты, связанные с Избранным.

- 1) `whenAddFavorite_thenAssetIsAddedToFavorites` - добавление актива в избранные.

Порядок действий:

1. Нажать на кнопку «В избранное» около актива.
2. Перейти на вкладку «Избранное».
3. Проверить, что количество активов равно 1.

2.2.3 DiversificationTest

В данном разделе описаны тесты, связанные с Диверсификациями.

- 1) `whenGoToDiversification_thenDiversificationAssetsAreShown` - получить активы своей диверсификации.

Порядок действий:

1. Перейти на вкладку «Диверсификации».
2. Нажать на диверсификацию.
3. Проверить, что активы показываются.

- 2) `givenValidAmount_whenCreateDiversification_thenDiversificationIsCreated` - создать диверсификацию с корректными данными.

Порядок действий:

1. Перейти на вкладку «Диверсификации».
2. Нажать на кнопку «Создать диверсификацию».
3. Ввести сумму «10000».
4. Нажать на кнопку «Создать диверсификацию».
5. Проверить, что количество диверсификаций равно 2.

- 3) `givenAmountSmallerThanMinPrice_whenCreateDiversification_thenErrorDialogIsShown` - создание диверсификации с указанием некорректной суммы диверсификации.

Порядок действий:

1. Перейти на вкладку «Диверсификации».
2. Нажать на кнопку «Создать диверсификацию».

3. Ввести сумму «1».
4. Нажать на кнопку «Создать диверсификацию».
5. Проверить, что показывается сообщение об ошибке.

2.2.4 ChangePasswordTest

В данном разделе описаны тесты, связанные с Изменением пароля.

- 1) `givenValidDetails_whenChangePassword_thenPasswordIsChanged` - изменение пароля с указанием корректных данных.

Порядок действий:

1. Перейти на вкладку «Настройки».
2. Ввести корректный старый пароль.
3. Ввести корректный новый пароль.
4. Ввести корректное подтверждение пароля.
5. Нажать на кнопку «Изменить пароль».
6. Проверить, что показывается сообщение об успехе.
7. Закрыть сообщение.
8. Нажать на кнопку «Выйти из аккаунта».
9. Ввести корректный логин.
10. Ввести старый пароль.
11. Нажать на кнопку «Войти».
12. Проверить, что показывается сообщение об ошибке.
13. Закрыть сообщение.
14. Ввести новый пароль.
15. Нажать на кнопку «Войти».
16. Проверить, что вход выполнен успешно.

- 2) `givenInvalidOldPassword_whenChangePassword_thenErrorDialogIsShown` - изменение пароля с указанием некорректных данных.

Порядок действий:

1. Перейти на вкладку «Настройки».
2. Ввести некорректный старый пароль.
3. Ввести корректный новый пароль.
4. Ввести корректное подтверждение пароля.
5. Нажать на кнопку «Изменить пароль».
6. Проверить, что показывается сообщение об ошибке.

Для незалогиненного пользователя

2.2.5 LoginTest

В данном разделе описаны тесты, связанные с входом в приложение.

- 1) `givenValidDetails_whenLogin_thenLoginIsSuccessful` - вход в приложение с использованием корректного логина и пароля.

Порядок действий:

1. Ввести корректный логин.
2. Ввести корректный пароль.
3. Проверить, что вход выполнен успешно.

- 2) `givenInvalidPassword_whenLogin_thenLoginIsUnsuccessful` - вход в приложение с использованием некорректного пароля.

Порядок действий:

1. Ввести корректный логин.
2. Ввести некорректный пароль.
3. Проверить, что показывается сообщение об ошибке.

2.2.6 RegisterTest

В данном разделе описаны тесты, связанные с регистрацией.

- 1) `givenValidDetails_whenRegister_thenRegistrationIsSuccessful` - регистрация в приложении с использованием корректных данных.

Порядок действий:

1. Перейти на страницу регистрации.
2. Ввести корректный логин.
3. Ввести корректный email.
4. Ввести корректный пароль.
5. Ввести корректное подтверждение пароля.
6. Нажать на кнопку «Зарегистрироваться».
7. Проверить, что показывается сообщение об успехе.
8. Ввести корректный логин.
9. Ввести корректный пароль.
10. Проверить, что вход выполнен успешно.

- 2) `givenDuplicateLogin_whenRegister_thenErrorDialogIsShown` - регистрация в приложении с использованием логина-дубликата.

Порядок действий:

1. Перейти на страницу регистрации.
2. Ввести логин-дубликат.
3. Ввести корректный email.
4. Ввести корректный пароль.
5. Ввести корректное подтверждение пароля.
6. Нажать на кнопку «Зарегистрироваться».
7. Проверить, что показывается сообщение об ошибке.

2.3 Отчет о прохождении тестов с результатами на сервере непрерывной интеграции

За работу сервера непрерывной интеграции отвечает конфигурационный файл `e2e_tests.yml`:

```
name: End-to-End tests
on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]
jobs:
  e2e-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Enable KVM
```

```

run: |
  echo 'KERNEL=="kvm", GROUP="kvm", MODE="0666", OPTIONS+="static_node=kvm"' | sudo tee /etc/udev/rules.d/99-kvm4all.rules
  sudo udevadm control --reload-rules
  sudo udevadm trigger --name-match=kvm

- name: Setup Java
  uses: actions/setup-java@v4
  with:
    distribution: 'temurin'
    java-version: '17'

- name: Setup Gradle
  uses: gradle/actions/setup-gradle@v3

- name: Restore AVD cache
  uses: actions/cache@v4
  id: avd-cache
  with:
    path: |
      ~/.android/avd/*
      ~/.android/adb*
    key: avd-33

- name: Create AVD
  if: steps.avd-cache.outputs.cache-hit != 'true'
  uses: reactivecircus/android-emulator-runner@v2
  with:
    api-level: 33
    arch: x86_64
    profile: pixel
    force-avd-creation: false
    emulator-options: -no-window -gpu swiftshader_indirect -noaudio -no-boot-anim -camera-back none -skin 1080x2400
    disable-animations: false
    script: echo "Generated AVD snapshot for caching."

- name: Build backend
  run: |
    cd server
    ./gradlew bootJar

- name: Prepare migration files
  run: |
    rm -rf server/src/main/resources/db/migration/*
    cp -r server/src/test/resources/integration/migration/* server/src/main/resources/db/migration/

- name: Run postgres
  uses: isbang/compose-action@v1.5.1
  with:
    compose-file: "server/src/main/resources/docker/database/docker-compose.yml"

- name: Run backend
  run: java -jar server/build/libs/tesseract-0.0.1-SNAPSHOT.jar &

- name: Run tests
  uses: reactivecircus/android-emulator-runner@v2
  with:
    api-level: 33
    arch: x86_64
    profile: pixel
    force-avd-creation: false
    emulator-options: -no-snapshot-save -no-window -gpu swiftshader_indirect -noaudio -no-boot-anim -camera-back none -skin 1080x2400
    disable-animations: true
    script: android/gradlew -p android connectedCheck

- name: Upload e2e test reports
  if: always()
  uses: actions/upload-artifact@v3
  with:
    name: e2e-test-reports
    path: android/app/build/reports

```

Как можно заметить, мы выполняем следующие шаги:

- Копируем наш репозиторий в систему непрерывной интеграции.
- Включаем поддержку KVM для ускорения запуска виртуальных машин.
- Устанавливаем виртуальную машину Java.
- Устанавливаем Gradle.
- Восстанавливаем кэш Android Virtual Device для ускорения создания эмуляторов Android.
- Создаем Android Virtual Device.
- Собираем backend с помощью Gradle.

- Заменяем реальные файлы миграций тестовыми.
- Запускаем PostgreSQL базу данных с помощью Docker Compose.
- Запускаем backend приложение.
- Запускаем end-to-end тесты для Android приложения.
- Загружаем артефакты тестов в систему интеграции.

Как можно заметить, запуск данного сценария происходит при следующих условиях:

- Произошел пуш в мастер-ветку.
- Открылся PR в мастер-ветку.

Страница с пройденным CI выглядит следующим образом:

The screenshot shows a GitHub Actions workflow run for the file `e2e_tests.yml` triggered by a push to the master branch. The status is 'Success' with a total duration of 4m 31s and 1 artifact. The job 'e2e-test' completed successfully in 4m 20s. There is one warning in the annotations: 'Node.js 16 actions are deprecated'. The artifacts section shows files produced during runtime.

Рис. 1: Пройденный CI

Ниже можно скачать архив с артефактами тестирования. Он содержит в себе HTML-страницу с подробным отчетом о пройденных тестах.

The Test Summary report shows the following statistics:

- 12 tests
- 0 failures
- 0 skipped
- 30.651s duration
- 100% successful

The report also includes a table of test results by package:

Package	Tests	Failures	Skipped	Duration	Success rate
ru.tesseract.assets	3	0	0	6.656s	100%
ru.tesseract.diversifications	3	0	0	6.273s	100%
ru.tesseract.login	4	0	0	9.493s	100%
ru.tesseract.settings	2	0	0	8.229s	100%

Generated by Gradle 8.3 at Feb 29, 2024, 6:48:34 PM

Рис. 2: Отчет о пройденных тестах