OTUS

Kotlin QA Engineer

Курс по активной прокачке навыков автоматизации тестирования на Kotlin для QA-инженеров и разработчиков на Kotlin

Длительность курса: 118 академических часов

1 Введение в Kotlin

1 **Эволюция языка Kotlin. Сравнение с другими языками.**

Цели занятия:

реализовывать типовые алгоритмические конструкции и создавать функции и классы на языке Kotlin;
использовать существующие библиотеки Java и Kotlin.

Краткое содержание:

основы Kotlin;
Kotlin Playground;
Gradle;
установка IDE IntelliJ IDEA Community Edition;

---

2 **Коллекции, Generic-типы, лямбда выражения**

Цели занятия:

Создавать расширения к существующим классам;
использовать библиотеки для работы с коллекциями объектов;
создавать делегаты к классам.

Краткое содержание:

kotlin.collections;
Android KTX;
Generics;

---

3 **Функции-расширения, делегаты**

Цели занятия:

создавать расширения к существующим классам;
использовать библиотеки для работы с коллекциями объектов;
создавать делегаты к классам.

Краткое содержание:

Реализация функций-расширений;
Работа с делегатами;

Домашние задания

1 Test Runner

Цель: Научиться работать с основными конструкциями языка и применять их на практике.

Дан интерфейс interface TestRunner { fun <T> runTest(steps: T, test: () -> Unit) }. Класс, передаваемый в steps, содержит методы before*/after*, которые задают предусловия/чистят данные перед/после теста.
Напишите свою реализацию интерфейса TestRunner, а именно:
1) Создайте класс реализующий интерфейс
2) Внутри класса переопределите метод runTest
3) Внутри этого метода необходимо сначала вызвать методы с before* из steps, далее прогнать тест (запустить передаваемую функцию test), и после вызвать методы с after* из steps.
4) Вызовы before* и after* нужно обернуть в лог/печать в консоль.
5) Проверьте работоспобоность написанного кода (можно в методе main).

Код залейте на github. Название репозитория - otus-qa-kotlin-hw, ветка hw01-testrunner. В ДЗ отправьте ссылку на pull request.

Вам потребуется 90-120 минут на выполнения задания.

Если возникнут сложности, вы всегда можете обсудить их с одногруппниками или задать вопрос преподавателю Slack

4 **Мультиплатформенная разработка, модули Gradle**

Цели занятия:

разрабатывать сценарии для автоматизации действий;
декомпозировать приложение на модули.

Краткое содержание:

Kotlin Scripting;
Kotlin Multiplatform;
перенос логики приложения в multiplatform-модуль;
преобразование gradle-файла в kts.

---

5 **Запуск и анализ результатов тестов с помощью Gradle**

Цели занятия:

создавать конфигурации gradle для выполнения тестов при сборке;
получать отчёты о тестировании и покрытии тестами кода.

Краткое содержание:

kotlin.test;
Allure Framework;
JaCoCo;
Codecov;
Allure;
интеграция запуска тестов со сборкой, создание отчёта покрытия кодом.

---

6 **Взаимодействие с СУБД для заполнения тестовых данных**

Цели занятия:

подключаться к реляционным и нереляционным базам данных;
заполнять тестовыми наборами данных, выполнять запросы и агрегации.

Краткое содержание:

основы SQL;
встраиваемые базы данных (SQLite, H2);
сохранение (на диск) и восстановление состояния для H2;
Exposed, Room, kmongo, SQLDelight;
создание классов для базы данных (H2);
сохранение и восстановление с диска;

2 Тестирование компонентов

1 Модульные тесты

Цели занятия:

создавать модульные (unit) тесты для функций и объектов классов;
использовать утверждения для проверки примитивных и составных типов результата.

Краткое содержание:

JUnit5;
TestNG;
kluent;
atrium;
assertK;

2 **Тестирование спецификаций**

Цели занятия:

разрабатывать и выполнять тесты на основе спецификаций;

Краткое содержание:

Spek;
Kotest;

Домашние задания

1 Расширение Kotest

Цель: Получить практику расширения тестовых фреймворков

Реализуйте расширение для Kotest, которое позволить повторить запуск тестов, в которых произошла ошибка.
1. Создайте класс RepeatOnFailureExtension, который импементирует TestCaseExtension
2. Создайте внутри класса переменную максимального количества перезапусков. Значение - любое на ваш выбор.
3. Переопределите метод intercept так, чтобы если результат выполнения теста - ошибка или неудача - тест перепрогонялся заново заданное в пункте 2 количество раз.
4. Примените данное расширение ко всем тестам в проекте.
5. Напишите минимум 1 позитивный и 1 негативный тест с использованием данного расширения

Вам понадобится 120-180 минут на выполнение домашнего задания.

Если возникнут сложности, вы всегда можете обсудить их с одногруппниками или задать вопрос преподавателю Slack

3	**BDD-тестирование**	**Цели занятия:** разрабатывать и исполнять сценарии тестирования (feature-файлы). **Краткое содержание:** Serenity; Cucumber;
4	**Статический анализ кода и проверка безопасности**	**Цели занятия:** определять правила тестирования стиля кода; обнаруживать отклонения от стиля; находить потенциальные уязвимости в коде. **Краткое содержание:** detekt; Horusec; insider; подключение зависимости (артефакт com.object-refinery:orsoncharts); выполнение аудита безопасности; контроль стиля кода через detekt.
5	**Тестирование производительности и профилирование**	**Цели занятия:** использовать measure-функции; разрабатывать правила для smoke-тестов и бенчмарков; собирать и анализировать информацию от JVM (через Java Flight Recorder). **Краткое содержание:** kotlinx.benchmark; Java Flight Recorder.

6 **Тестирование Enterprise-приложений на Spring**

Цели занятия:

создавать и выполнять тесты для компонентов на Java Spring;
создавать тестовые реализации компонентов.

Краткое содержание:

основы DI (IoC);
использование mock для тестирования;
Spring Boot (основы DI, MockBean, Spring Boot Testing).

3 Мобильная разработка и тестирование

1 Тестирование UI на основе Material

Цели занятия:

определять селекторы для элементов на экране;
разрабатывать автоматические тесты для мобильных приложений на Android;
использовать механизмы синхронизации для корректной обработки асинхронных операций и анимаций;
выполнять тестирование многоэкранных приложения с внешними вызовами (интентами);
создавать скриншоты при выполнении теста.

Краткое содержание:

Android Emulator;
Espresso (основная библиотека и расширения IdlingResources и Intent);
Kakao;
Hamcrest;
Barista;
ознакомление с мобильным приложением "Мои привычки" с хранением данных в переменных, вариант на Android Material (каталог привычек, дневник выполнения, профиль - здоровье виртуального персонажа, имя и фотография авторизованного пользователя), доступ через GitHub;
обсуждение сценариев тестирования и возможных затруднений.

2 **Тестирование UI (на основе Material) с использованием Espresso/Kakao и Hamcrest Часть 2**

Цели занятия:

определять селекторы для элементов на экране;
разрабатывать автоматические тесты для мобильных приложений на Android;
использовать механизмы синхронизации для корректной обработки асинхронных операций и анимаций;
выполнять тестирование многоэкранных приложения с внешними вызовами (интентами);
создавать скриншоты при выполнении теста.

---

3 **Тестирование UI на основе Jetpack Compose**

Цели занятия:

использовать семантические модификаторы для виджетов;
разрабатывать автоматические тесты для взаимодействия с элементами и проверки наличия/содержания;
использовать правила (rules) для создания скриншотов и навигации.

Краткое содержание:

Jetpack Compose UI Testing;
Multiplatform Compose;
разбор мобильного приложения "Мои привычки" с хранением данных в переменных, вариант на Jetpack Compose, доступ через GitHub;
обсуждение семантического дерева, сравнение методов тестирования с Android Material.

Домашние задания

1 Разработать автоматические тесты для сценариев: просмотр списка привычек, создания новой привычки, выбор привычки и просмотр страницы подробностей привычки, отметка выполнения привычки, просмотр персонального профиля, проверка значения здоровья, вызов вне

Цель: В результате выполнения задания вы создадите инструментальные тесты для автоматического тестирования функций мобильного приложения с точки зрения пользователя.
В данном задании тренируются навыки:
- определения способа идентификации вида (view) / виджета (composable) в мобильном приложении;
- проверки наличия/отображения вида (view) / виджета (composable) и его содержимого;
- взаимодействия с видом (view) / виджетом (composable);
- синхронизации при ожидании анимации или результатов сетевых запросов.

Необходимо реализовать:
- переход на страницу списка привычек, создание новой привычки с названием "Каждый день писать тесты";
- переход на страницу списка привычек, прокрутки до элемента ""Каждый день писать тесты"", нажатие на этот элемент списка;
- тест перехода на страницу с описанием привычки (проверка наличия элемента с названием, проверка содержимого текстового элемента);
- отметка выполнения привычки, проверка изменения состояния страницу привычки;
- переход на страницу профиля, проверка значения состояния здоровья;
- нажатие на кнопку "Поделиться" и проверка вызова Intent.ACTION_SEND;
- создать скриншоты для состояния после каждого пункта, перечисленного выше.

Тесты должны быть реализованы дважды - для варианта Material Components (в GitHub репозитории, ветка qa-kotlin-5-1) и для варианта Jetpack Compose (в GitHub репозитории, ветка qa-kotlin-5-2).
По вопросам обращаться в Slack к студентам, преподавателям и наставникам в канал группы.

4	**Использование mock-объектов для тестирования приложений**	**Цели занятия:** разрабатывать mock-реализации (значений и методов) для использования в unit-тестах и в тестах UI; давать рекомендации по созданию кода приложения для возможного тестирования через mock. **Краткое содержание:** mockito; mockk; разбор варианта мобильного приложения "Мои привычки" (Jetpack Compose) с хранением данных в SQLite; обсуждение использования mock для устойчивого выполнения тестов.
5	**Тестирование сетевых приложений (на OkHttp MockWebServer и Retrofit)**	**Цели занятия:** создавать mock API для тестирования сетевых мобильных приложений; запускать тесты с ограничением скорости сети, имитировать неустойчивую связь. **Краткое содержание:** основы сетевого взаимодействия для Android/iOS; Ktor client; OkHttp; OkHttp MockWebServer; Retrofit; Network Link Conditioner (MacOS); tc (Linux); clumsy (Windows); разбор варианта мобильного приложения "Мои привычки" (Jetpack Compose) с использованием; обсуждение использования имитации API для реализации тестов.

Домашние задания

1 Разработать автоматические тесты с использованием mock для приложения с хранением данных в SQLite.

Цель: В результате выполнения задания вы расширите набор тестов мобильного приложения и сможете выполнить тестирование локального кэширования и сетевого взаимодействия без необходимости устанавливать тестовый сервер.
В данном задании тренируются навыки:
- реализации mock для структур данных и веб-запросов;
- оценки возможности тестирования сетевого приложения;
- оценки архитектуры компонентов приложения с точки зрения возможности использования тестовых данных.

Разработать автоматические тесты с использованием mock для приложения с хранением данных в SQLite. Сценарии тестирования: просмотр страницы списка привычек, добавление новой привычки (с названием "Не забывать про состояние базы данных"), проверка наличия нового элемента в списке, удаление созданной привычки, проверка отсутствия элемента. Аналогично сделать реализацию API для варианта с сетевым взаимодействием.

Необходимо реализовать:
- создать mock для объекта взаимодействия с SQLite и (для тестового исполнения) подменить методы получения списка привычек и создания новой (для работы со списком в памяти) - для вариант с SQLite;
- создать MockWebServer (метод GET /habit возвращает значение из списка в памяти, список инициализируется константой, методы POST /habit и DELETE /habit/:id взаимодействуют со списком в памяти) и настроить OkHttp-клиент на взаимодействие с mock-сервером;
- разработать автоматические тесты для просмотра списка привычек, создания новой

привычки, проверки наличия, удаления привычки, проверки отсутствия;
- сформировать отчёты в Allure по результатам выполнения тестов. Отчёты необходимо загрузить в каталог reports github-репозитория проекта.

В репозитории проекта должно быть две ветки - для mock в SQLite варианте (в GitHub репозитории, ветка qa-kotlin-6-1) и для MockWebServer (в GitHub репозитории, ветка qa-kotlin-6-2).
По вопросам обращаться в Slack к студентам, преподавателям и наставникам в канал группы.

6 **Кроссплатформенное тестирование интерфейса методом "черного ящика"**

Цели занятия:

выбрать тестовый движок TestRunner;
настроить автоматическое выполнение тестов (как в модели "белого ящика" при наличии доступа к коду, так и в модели "чёрного ящика");
создать автоматические тесты для существующих приложений без доступа к коду;

Краткое содержание:

UI Automator;
Appium;
Calabash;
RoboElectric TestRunner;
Firebase TestLab;
обсуждение тестирования "чёрного ящика" для мобильного приложения "Мои привычки";
тестирование произвольных приложений.

Домашние задания

1 Опциональное творческое задание

Цель: подготовить автоматический тест для любого выбранного приложения из Play Store / AppStore.

1. Найдите интересное для вас приложение в Play Store / AppStore
2. Напишите тесты инструментальной проверки не менее чем для двух экранов (необходимо проверить все доступные элементы интерфейса и переход между экранами)
3. Подготовьте отчёт о тестировании в виде текстового документа, включающего перечисление тестируемых элементов и сценариев

4 Сложные вопросы тестирования реальных приложений

1 Dependency Injection для тестирования фрагментов

Цели занятия:

создавать реализацию тестовых классов и интегрировать в приложение с использованием инъекции зависимостей;
создавать и регистрировать тестовые модули в точке входа мобильного приложения.

Краткое содержание:

основы clean-архитектуры;
Dagger;
Hilt;
Koin;
разбор мобильного приложения "Мои привычки" с сетевым подключением и локальным кэшированием (в SQLite);
замена компонентов сетевого варианта "Мои привычки" на Hilt-компоненты;
создание тестового модуля и замена репозитория на тестовый.

---

2 Корутины и тестирование асинхронных приложений

Цели занятия:

использовать механизмы синхронизации (очереди, семафоры);
создавать диспетчеры и scope для основного потока и ввода-вывода;
выполнять мониторинг состояния корутин;
выполнять тестирование API.

Краткое содержание:

основы асинхронного выполнения;
реализация корутин в kotlin;
kotlinx-coroutines-test;
потоки выполнения в мобильных приложениях (на примере Main / IO);

тестирование API-клиентов (корутины) в сетевом варианте "Мои привычки".

Домашние задания

1. Выполнить замену всех компонентов мобильного приложения "Мои привычки" (сетевой вариант с локальным кэшированием) на Hilt, разработать тестовую реализацию для всех модулей и для асинхронных методов репозитория (взаимодействует с SQLite и сетью).

   Цель: В результате выполнения задания вы улучшите тестовое покрытие и проверите корректность работы логики репозитория мобильного приложения.
   В данном задании тренируются навыки:
   - адаптации приложения для тестирования;
   - использования инъекций зависимостей для создания тестовых реализаций компонентов;
   - тестирования асинхронных функций и корректного применения контекстов корутин.

   Необходимо реализовать:
   - создать конфигурацию для основного и тестового модуля Hilt;
   - перенести все компоненты в Hilt (ViewModel, Repository, DataSource, Service);
   - разработать тесты взаимодействия с сервером через класс источника данных MyHabitsDataSourceOnline (тесты должны возвращать систему в исходное состояние и проверять все методы - список привычек, описание привычки, создание привычки, выполнение привычки, отмена выполнения привычки, удаление привычки);
   - реализовать тестовую реализацию для источника данных MyHabitsDataSourceOffline (кэш) и описать тесты для проверки функционирования кэша;
   - сформировать отчёты в Allure по результатам выполнения тестов. Отчёты необходимо загрузить в каталог reports github-репозитория проекта.

   Для доступа к API необходимо использовать персональный токен, отправленный в личном кабинете.
   Домашнее задание отправляется ссылкой на github-репозиторий, в котором должна присутствовать ветка qa-kotlin-7.

По вопросам обращаться в Slack к студентам, преподавателям и наставникам в канал группы.

3	**Тестирование изменяемых данных и потоков состояний**	**Цели занятия:** разрабатывать тесты для реактивных приложений на RxJava; разрабатывать тесты для приложений на основе наблюдаемого потока (StateFlow). **Краткое содержание:** основы реактивного взаимодействия и flow; понятие потока, модель publish/subscribe; RxKotlin; Flow/SharedFlow в Kotlin; Turbine; разбор Rx-варианта приложения "Мои привычки" (с web-socket для обновления списка привычек и состояния здоровья); разработка теста для асинхронного обновления списка привычек.

5 Мультиплатформенная разработка и тестирование

1 **KotlinJS и тестирование взаимодействия с JS-библиотеками и веб-приложениями**

Цели занятия:

разрабатывать автоматические тесты для веб-приложений (в т.ч. созданных с использованием KotlinJS);
создавать отчёты о тестировании web-приложений в Allure.

Краткое содержание:

основы разработки веб-приложений на Kotlin;
взаимодействие с JS;
KVision;
Selenium;
Selenide;
Kirk;
разбор веб-приложения "Мои привычки" (KotlinJS);
обсуждение сценариев тестирования и возможных сложностей;
создание теста для сценария "создать привычку".

---

2 **Разработка и тестирование backend на Ktor**

Цели занятия:

разрабатывать автоматические тесты для веб-серверов (API);
выполнять запросы от авторизованных пользователей/сервисов;
создавать тестовые реализации бизнес-логики и слоя данных;
создавать отчёты о тестировании API в Allure.

Краткое содержание:

REST;
Swagger;
Ktor;
Ktor Testing;
разбор реализации API "Мои привычки;

создание тестового компонента доступа к данным (используется DI koin);
разработка теста для метода "Получить список привычек".

Домашние задания

1. Разработать тестовую реализацию компонента доступа к данным. Реализовать тесты для методов API управления списком привычек и их выполнением.

   Цель: В результате выполнения задания будут созданы тесты для серверной реализации API на Kotlin.
   В данном задании тренируются навыки:
   - создания компонентов Koin для замены базы данных на состояние в памяти;
   - вызовов API с использованием Ktor Testing;
   - проверка корректности обработки ошибок и изменения состояния при вызове методов API.

   Необходимо реализовать:
   - тестовый класс для доступа к данным (с использованием локального состояния в памяти);
   - тестовый модуль Koin для связывания с тестовыми реализациями интерфейса доступа к данным;
   - зарегистрировать тестовый модуль при запуске тестов;
   - создать серию тестов для всех методов с префиксом /habit;
   - сформировать отчёт в Allure по результатам выполнения тестов в двух вариантах: при использовании доступа к данным с базой данных, при использовании доступа к данным для локального состояния. Отчёт необходимо загрузить в каталог reports github-репозитория проекта.

   Для доступа к API необходимо использовать персональный токен, отправленный в личном кабинете.
   Домашнее задание отправляется ссылкой на github-репозиторий, в котором должна присутствовать ветка qa-kotlin-9.
   По вопросам обращаться в Slack к студентам, преподавателям и наставникам в канал группы.

3 **Нагрузочное тестирование веб-сервера на Gatling**

Цели занятия:

создать наборы правил для оценки производительности сервера;
интегрировать проверку скорости работы сайта и сценариев для пользователя;
создавать нагрузку и оценивать производительность сервера при заданном количестве запросов в секунду.

Краткое содержание:

Gatling;
конфигурирование тестов
нагрузочное тестирование API
анализ результатов HT

4 **Kotlin Native и тестирование нативных приложений**

Цели занятия:

разрабатывать тесты для нативных приложений и их взаимодействия со сторонними библиотеками и возможностями целевой платформы;
создавать тесты приложений для микроконтроллеров.

Краткое содержание:

LLVM;
взаимодействие с C;
кросскомпиляция в iOS, Linux, MacOS;
микроконтроллеры и эмулятор STM32;
особенности создания unit-тестов для native-приложений.

Домашние задания

1 Создать тесты LightHouse для всех страниц интерфейса приложения ""Мои привычки"". Выполнить нагрузочное тестирование всех методов GET в API (при 10, 100, 1000 и 10000 запросов в секунду)

Цель: В результате выполнения задания будут созданы автоматические тесты для нефункциональной спецификации

(быстродействие, адаптивность, визуальная привлекательность)
В данном задании тренируются навыки:
- проектирования и выполнения нагрузочного тестирования API;
- оценки быстродействия веб-страниц и сценариев;
- создания плана тестирования веб-приложений (включая авторизованную область).

Необходимо реализовать:
- правила интеграции LightHouse в сборку gradle;
- набор тестов LightHouse для анализа страниц веб-сайта при сборке из git-репозитория;
- код теста для анализа времени ответа API в зависимости от количества запросов в секунду;
- сформировать отчёт в Allure по результатам выполнения тестов. Отчёт необходимо загрузить в каталог reports github-репозитория проекта.

Для доступа к защищенным страницам необходимо использовать персональный токен, отправленный в личном кабинете.
Домашнее задание отправляется ссылкой на github-репозиторий, в котором должна присутствовать ветка qa-kotlin-10.
По вопросам обращаться в Slack к студентам, преподавателям и наставникам в канал группы.

6 Инфраструктура тестирования

1 **Настройка Jenkins и проекта для автоматической сборки full-stack приложения**

Цели занятия:

настраивать конвейер сборки (с запуском тестов) в Jenkins;
отслеживать изменение метрик покрытия, результатов тестов.

Краткое содержание:

Jenkins;
настройка расширений в проекте;
сбор результатов выполнения тестов;
накопление метрик покрытия тестами.

---

2 **Использование контейнеров и систем оркестрации для тестового окружения**

Цели занятия:

создавать контейнеры для тестирования продукта в изолированном окружении;
интегрировать выполнение тестов в контейнеры;
извлекать отчёты о тестировании из контейнеров.

Краткое содержание:

основы технологий контейнеризации;
OCI;
создание Dockerfile;
Docker Compose;
Docker Swarm;
Kubernetes (kube-mini);
практикум по созданию Dockerfile для приложения "3D-морской бой" и API для "Мои привычки".

---

3 **Включение тестов в сборочный конвейнер Github Actions, Gitlab + Docker, Atlassian Bitbucket**

Цели занятия:

настраивать конвейер сборки (с запуском тестов) в Gitlab / Github / Bitbucket;
отслеживать изменение метрик покрытия, результатов тестов.

Краткое содержание:

тестовые сценарии в контейнерах и конвейере Github Actions, Bitbucket;
запуск тестовых сервисов; артефакты сборки.

Домашние задания

1. Реализовать сборку jar-артефакта игры ""3D-морской бой"" с использованием Jenkins. Создать конфигурации сборки для API с использованием Github Actions (должен быть доступен в виде образа контейнера в Github Container Registry) и веб-интерфейса приложения

   Цель: В результате выполнения задания будут созданы рабочие сценарии, которые могут быть использованы в дальнейшей в качестве основы для интеграции тестирования в конвейер CI/CD в рабочих процессах.
   В данном задании тренируются навыки:
   - создания, выбора и использования образов контейнеров;
   - разработки сценариев сборки и тестирования с использованием контейнеризации;
   - запуска вспомогательных сервисов и заполнения тестовыми данными;
   - интеграции тестирования в CI/CD.

   Необходимо реализовать:
   - проект Jenkins для сборки и тестирования приложения ""3D-морской бой"" с прерыванием сборки при обнаружении ошибки при выполнении теста и снижении покрытия тестами величины в 50%;
   - конфигурация сборки и тестирования API проекта для Github Actions (в ветке qa-kotlin-11 github-репозитория проекта, тесты API с подключением к тестовому сервисы базы данных, нагрузочные тесты);
   - конфигурация .gitlab-ci.yml для сборки и автоматического тестирования веб-приложения ""Мои привычки"" (тесты взаимодействия с интерфейсом, lighthouse).

   Для доступа к API необходимо использовать персональный токен, отправленный в личном кабинете.
   Домашнее задание отправляется в виде XML-

файла экспорта проекта Jenkins, ссылки на репозиторий github (ветка qa-kotlin-11) и ссылки на публичный репозиторий gitlab.

По вопросам обращаться в Slack к студентам, преподавателям и наставникам в канал группы.

---

4 **Создание тестовой инфраструктуры из контейнеров и заполнение тестовыми данными**

Цели занятия:

применять полученные знания по разработке и автоматизации тестов для всестороннего тестирования проекта.

Краткое содержание:

обсуждение возможных сложностей и разных сценариев в реальных проектах.

7 Итоговый проект

1 Выбор темы и организация проектной работы

Цели занятия:

выбрать и обсудить тему проектной работы;
спланировать работу над проектом;
ознакомиться с регламентом работы над проектом.

Краткое содержание:

правила работы над проектом и специфика проведения итоговой защиты;
требования к результату проекта и итоговой документации.

Домашние задания

1 Проектная работа

---

2 Консультация по проектам и домашним заданиям

Цели занятия:

получить ответы на вопросы по проекту, ДЗ и по курсу.

Краткое содержание:

вопросы по улучшению и оптимизации работы над проектом;
затруднения при выполнении ДЗ;
вопросы по программе."

3	**Защита проектных работ**	**Цели занятия:** защитить проект и получить рекомендации экспертов. **Краткое содержание:** презентация проектов перед комиссией; вопросы и комментарии по проектам. * в защите могут участвовать и студенты, не выполняющие собственного проекта, но желающие принять участие в обсуждении проектов своих коллег.
4	**Подведение итогов курса**	**Цели занятия:** узнать, как получить сертификат об окончании курса, как взаимодействовать после окончания курса с OTUS и преподавателями, какие вакансии и позиции есть для выпускников (опционально - в России и за рубежом) и на какие компании стоит обратить внимание **Краткое содержание:** организационные вопросы; рынок вакансий по направлению; статистика курса и вопросы по курсу.