Федеральное государственное автономное образовательное учреждение
высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий
Кафедра «Инфокогнитивных технологий»

Направление подготовки/ специальность: системная и программная инженерия

ОТЧЕТ

по проектной практике

Студент: Первухин Роман Александрович Группа: 241–3211

Место прохождения практики: Университет «Сириус», научно-практический интенсив по мобильной разработке от компании «Яндекс»

Отчет принят с оценкой _____ Дата _____

Руководитель практики: Мария Журавлёва, менеджер молодёжных программ Яндекс

Москва 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Интенсив по мобильной разработке от компании «Яндекс» в университете «Сириус»

Интенсив проходил в течение двух недель, по результатам работы мной был получен сертификат об успешном освоении программы в 88 академических часов.

Я вёл проектную работу по разработке мобильного приложения на *Flutter* (***Flutter*** – **кроссплатформенный фреймворк** для создания **нативных** мобильных, веб- и десктопных приложений из **единой кодовой базы** на языке ***Dart***).

Нашим ментором из команды *ЯндексПро* было предложено абстрактное видение задачи. Мы, доработав её до полноценного ТЗ, приступили к реализации superapp **"GymBro"**.

Основной репозиторий проектной практики: https://github.com/PervuhinRoman/Mospolytech-practice-2025-1.

GymBro superapp

Подробнее: https://github.com/PervuhinRoman/GymBroForPress

Некоторые скриншоты экрана-карты:

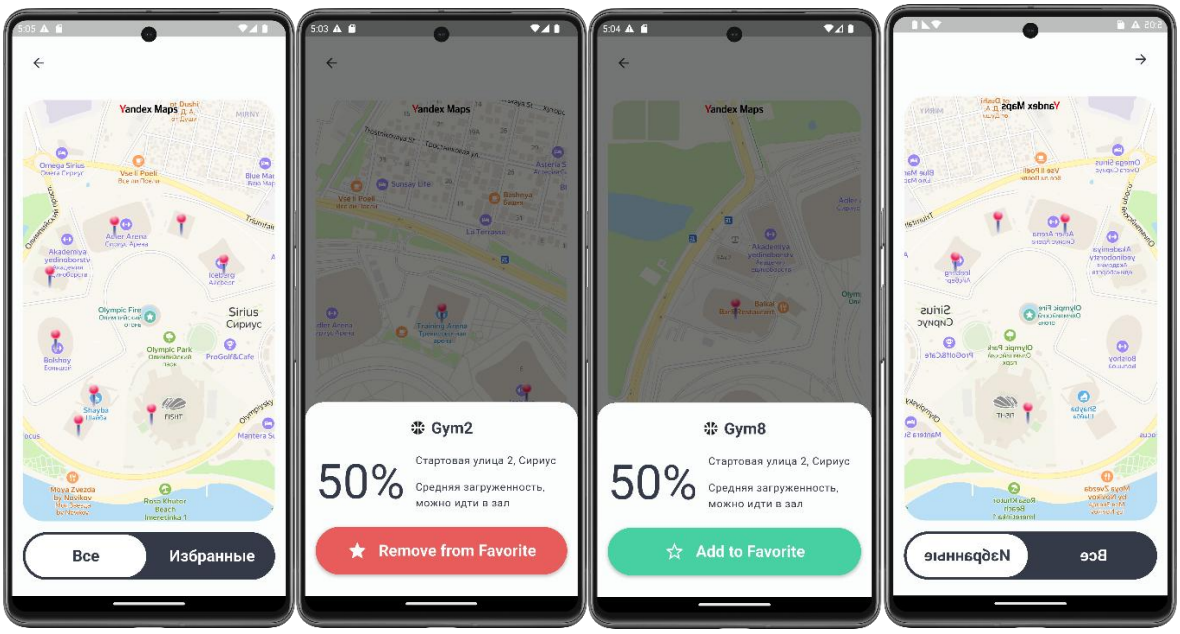


Рис. 1. Скриншоты экрана функциональности выбора локации зала

GymBro – приложение для всех, кто занимается спортом, сочетающее в себе:

- **AI-ассистента** для составления расписания и наполнения тренировок
- **Карту** залов с возможностью оценки их загруженности
- Поиск твоего *Gym-Bro* для совместных тренировок в ***"Tinder-like"*** стиле
- **Календарь** тренировок для контроля режима занятий спортом

Приобретенные знания

Dart

В ходе обучения мы ознакомились со всеми концепциями языка Dart. Некоторые из них подавались сжато в целях экономии времени. Тем не менее были полностью разъяснены следующие темы:

- Базовый синтаксис
- Типы данных
- Null-safety
- Управляющие операторы
- «Синтаксический сахар» языка
- Реализация ООП в Dart

Подробного описания заслуживают:

Null-Safety

Null-safety в Dart — это механизм, который помогает избежать ошибок, связанных с обращением к null (нулевым ссылкам). Он делает типы данных по умолчанию не nullable (не могут быть null), если явно не указано обратное.
Познакомились с механизмами работы операторов:
! («bang»-оператор) – утверждение, что значение не null (опасно, если ошибочно)
?. (conditional access) — безопасный доступ
?? (null-coalescing) — значение по умолчанию
Null-Safety стала важным нововведением в Dart, т. к. теперь все сущности, плагины, пакеты должны следовать этой парадигме, а все предыдущие версии без null-safety не обратно совместимы.

ООП

Язык Dart реализует классическую схему ООП, но имеет обширный ряд особенностей и полезных фич, с которыми мы познакомились:

1. Инкапсуляция. Доступ к членам класса регулируется только двумя модификаторами:
    - public (по умолчанию) — доступ отовсюду внутри библиотеки
    - private (с подчеркиванием) — только внутри файла
2. Наследование. Dart поддерживает одиночное наследование (один родительский класс) через ключевое слово extends. Для переопределения методов используется @override. Миксины (mixin, with) позволяют добавлять функциональность без множественного наследования.
3. Миксины (Mixins). Уникальная черта Dart — миксины, которые позволяют подмешивать функциональность в классы без наследования. Определяются через mixin и подключаются к классу через with. Отличаются от интерфейсов тем, что могут содержать реализацию методов.
4. Полиморфизм. Переопределение методов (override) — дочерние классы могут менять поведение родительских методов. **Интерфейсы — все классы неявно определяют интерфейс, который можно реализовать через implements.** Абстрактные классы (abstract) — могут содержать абстрактные методы (без реализации), требующие переопределения в дочерних классах.
5. Интерфейсы. В Dart нет отдельного ключевого слова для интерфейсов — любой класс может быть интерфейсом. Класс может реализовывать (implements) несколько интерфейсов, но наследовать (extends) — только один.
6. Фабричные конструкторы. Dart позволяет создавать объекты через фабричные конструкторы (factory), которые могут возвращать экземпляры из кэша или подклассов, нарушая стандартное правило о возврате нового объекта.

7. Каскадная нотация (Cascade). Синтаксис «..» (двойная точка) позволяет выполнять несколько операций над одним объектом без повторного обращения к нему. Это особенно полезно для цепочек вызовов методов.
8. Null-safety и ООП. С включенной null-безопасностью все типы по умолчанию non-nullable. Это влияет на поля классов, методы и наследование, требуя явного указания «?» для nullable-типов.
9. Расширения (Extension Methods). Позволяют добавлять методы к существующим классам без их модификации, даже к стандартным типам (String, int).
10. Иммутабельность и const-конструкторы. Dart поощряет иммутабельность: классы могут иметь const-конструкторы для создания неизменяемых объектов, которые кэшируются на этапе компиляции.

Асинхронность

Является важной составляющей любого языка программирования. Тема асинхронности рассматривалась сразу в парадигме Flutter. Так, асинхронность в Flutter имеет ряд особенностей:

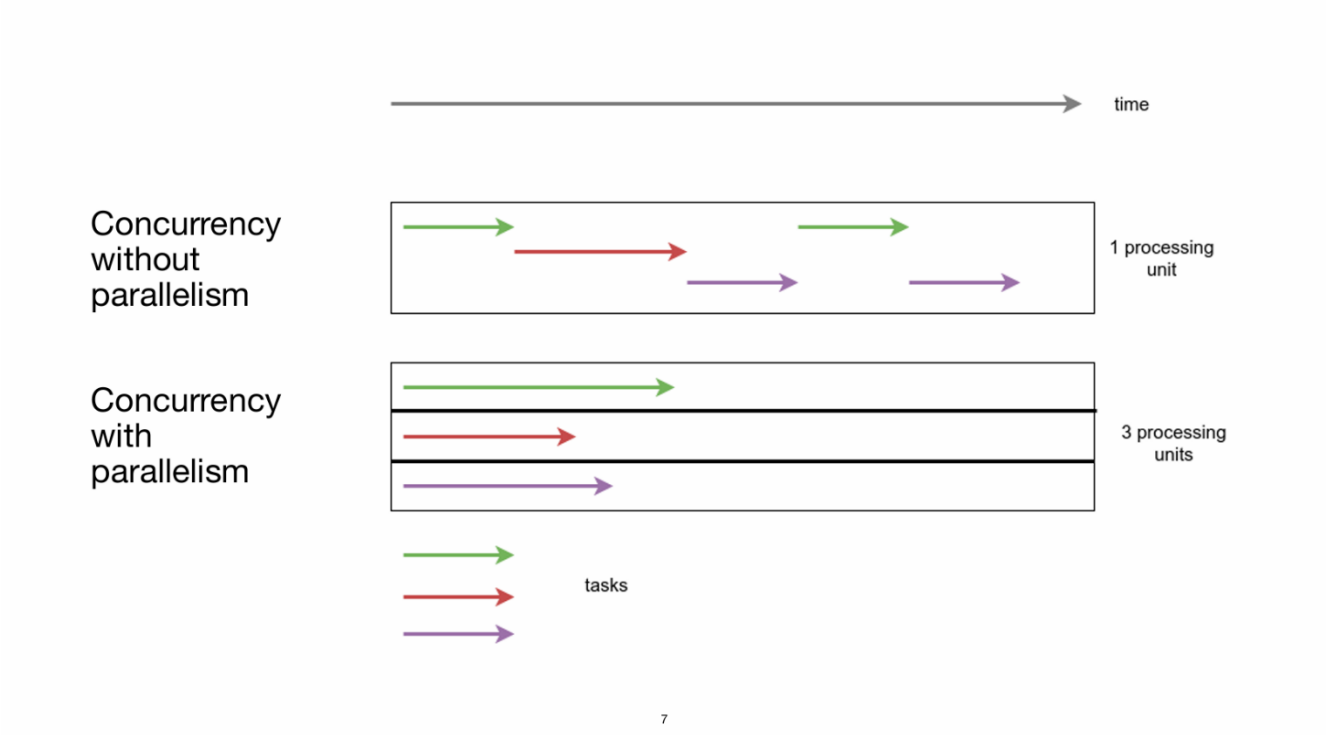
- В данных технологиях есть чёткое разграничение понятий: синхронный код, асинхронный код и параллелизм.
- «Асинхронность» осуществляется на уровне виртуальной машины Dart.
- «Параллелизм» осуществляется на нативном уровне системы (использование «изолятов»).

Асинхронность

«Весь код в Dart и Flutter синхронный». Dart VM использует однопоточную модель. Это значит, что весь код выполняется внутри одного потока (изолята) системы. Когда мы реализуем «асинхронную» реализацию, на самом деле мы запускаем процесс синхронного переключения между процессами, в итоге достигается эффект асинхронного выполнения. Данный процесс контролирует Event Loop.

Параллелизм

Достичь действительно параллельного выполнения кода для по-настоящему тяжеловесных операций можно с помощью механизма изолятов. На практике нам не пришлось обращаться к нему, но основные методы работы были рассмотрены.



Flutter

Разумеется, были рассмотрены основы программирования на самом Flutter. Перечень тем, которые были изучены с наставниками и самостоятельно:

1. Понятие Widget, виды виджетов
2. Понятие состояния
3. WidgetTree
4. Понятие Element
5. ElementTree
6. Ключи
7. InheretedWidget

8. Анимации
9. Навигация
10. Хранение данных

State Management

State Management (управление состоянием) — это подход к организации данных, которые изменяются во время работы приложения, и их синхронизации между различными частями программы. В любом приложении есть состояние (например, данные пользователя, настройки, UI-состояния), и важно управлять им эффективно.

Основные принципы:

- Единый источник истины (Single Source of Truth) — состояние хранится в одном месте.
- Реактивность — автоматическое обновление зависимых частей при изменении состояния.
- Инкапсуляция логики — отделение бизнес-логики от UI.

Мы выделили и изучили 3 вида State Management в Flutter:

1. Встроенный
2. Рекомендуемый Flutter Team
3. Сторонний

Поддерживать выполнение основных принципов можно с помощью стандартных инструментов фреймворка: setState(), InheretedWidget и т. д., но этот путь громоздкий и плохо масштабируемый в рамках больших проектов.

Тогда можно использовать более продвинутый способ – Provider. По сути, это удобная обёртка над вышеописанным не всегда удобным способом.

И тем не менее существуют множество других подходов и путей для организации State Management в Flutter.

Для нашего проекта был применён Riverpod. Ещё более глубокая модернизация Provider. Данный фреймворк был выбран также потому, что реализует механизмы DI (Dependency Injection (DI) — механизм, который автоматически предоставляет зависимости (например, сервисы, репозитории, конфиги) там, где они нужны, без ручного создания экземпляров), который стал необходим в нашем superapp.

Архитектура

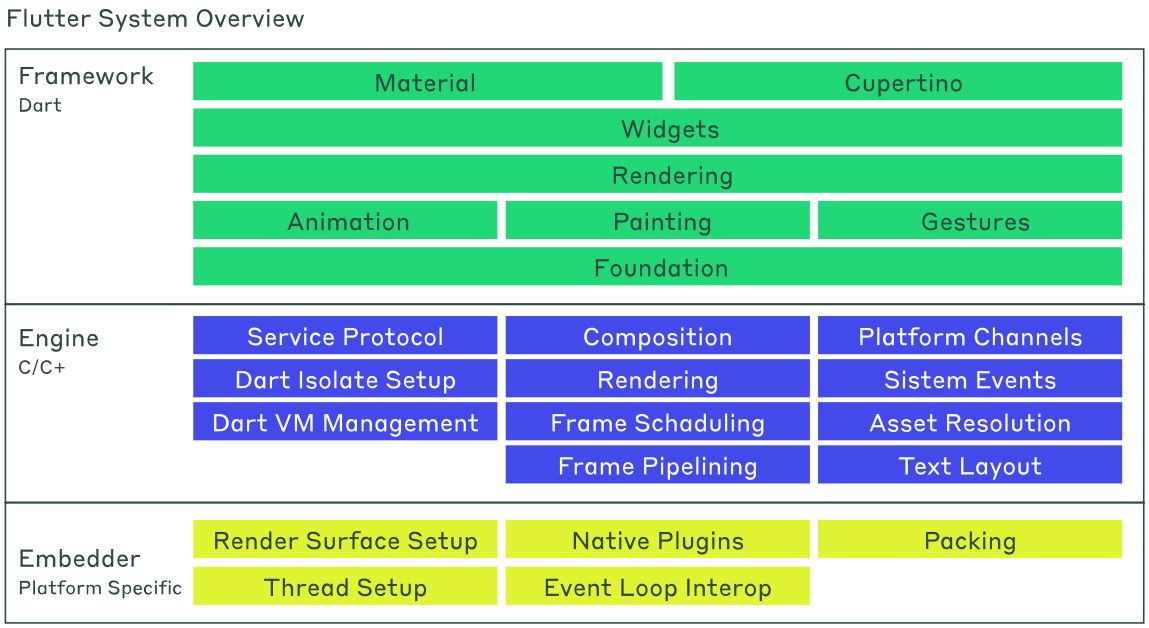
Нами была изучена архитектура мобильных приложений. Мы выделили несколько уровней архитектуры:

1. Уровень кода (code style, effective dart, linter, чистый код)
2. Уровень классов (принципы SOLID, DRY, KISS, YAGNI)
3. Организация компонентов (rep, ccp, crp, adp, sdp, sap; паттерны проектирования)
4. Уровень приложения (арх. шаблоны, dependencies & state management)

В общих чертах были рассмотрены все уровни. Особое внимание мы уделили уровню классов (за соблюдением принципов следил наш ментор в качестве ревьюера кода в GitHub) и уровню приложения. (Так или иначе при разработке задеваются все уровни, разница лишь в том, насколько много осознанного внимания уделяется каждому из них)

В итоге наше приложение реализует:
MVVM, Riverpod specific архитектуру с разделением на слои упрощённого clean-a и Feature-fisrt подходом.

Устройство нативных частей iOS и Android



Сложной, но необходимой темой является изучение нативных особенностей платформ, на которые будет осуществляться деплой продукта.

Главная особенность и отличие Flutter от других кроссплатформенных фреймворков в том, что Flutter сам рисует каждый пиксель на экране, в архитектуре нет "моста" между Flutter и нативом, Flutter не просит платформу отобразить кнопку, а сам рисует её. Наиболее точно это можно сравнить с игровым движком (игры выглядят одинаково на разных платформах). В большинстве задач нам не приходится обращаться к нативной части, но бывают и исключения, когда одни и те же механизмы (например, взаимодействие с геоданными, как будет показано дальше) имеют разную нативную реализацию и не поддаются централизованному замещению движком Dart.

В GymBro я также отвечал за интеграцию карты от Яндекс для реализации функционала выбора наименее загруженного зала, для этого было необходимо:

1. Написать некоторый объём кода на нативных языках для платформ Android и iOS (Kotlin и Swift соответственно)

2. В настройках нативных частей проекта предоставить соответсвующие разрешения приложения для доступа к геоданным
3. В системах сборки каждой платформы выполнить подключение SDK карт Яндекс
4. Решить сопутствующие проблемы

Пункт 4 почётно находится в этом списке, потому что он один занимает уверенные 75% от всего объёма задачи по интеграции карт.

Для решения этой задачи пригодились следующие изученные темы:

- Работа Gradle (в т.ч. Gradle.kts) для Android
- Работа Cocoapods для iOS
- Понимание Manifest для Android
- Понимание info.plist для iOS
- Принципы работы Dart VM

Как видим, совсем чуть-чуть (нет)

Тестирование и Deploy

Тестирование

Ни один настоящий проект не может обойтись без тестирования. Методы контроля качества продукта, которые мы применяли в проекте:

1. Ревью кода опытным ментором
2. Использование linter-а
3. Осуществление CI/CD тестов в GitHub
4. Создание Unit-тестов

Изучили, но на практике не применили:

1. Создание Widget/Golden-тестов
2. Создание E2E-тестов

Deploy

Нами были изучены процессы получения артефакта приложения для двух платформ: Android и iOS. На практике удалось создать только артефакт Android. Это связано с тем, что лицензия Apple для публикации приложения намного более труднодоступна в силу в том числе политических факторов.
Для деплоя вновь стало необходимым углубиться в особенности работы Dart VM и некоторых других нативных процессов в Flutter-фреймворке.

Не mobile development

Т. к. в рамках проектной работы на интенсиве нам было необходимо осуществить полный цикл разработки приложения, были затронуты темы, которые на прямую не связаны с преподаваемым материалом.
Так я познакомился с технологией быстрой разработки backend-части (в т. ч. хранилища данных) средствами **Firestore**.
В проекте также применялась технология **Firebase** и сервер на GoLang.

Итоги

Итогом работы стала бета-версия приложения GymBro – superapp-сервиса для совместных тренировок. Проект был представлен на общей сессии защиты проектов в финальный день интенсива, где получил положительные отзывы менторского жюри и других команд.

В ходе разработки и обучения мы охватили множество тем, часть из которых была описана выше. Объёма этих знаний достаточно для самостоятельной разработки мобильных кроссплатформенных приложений на технологии Flutter. Разумеется, данные знания необходимо углублять и совершенствовать, после интенсива я составил план по их развитию и в настоящий момент приступаю к его выполнению.

Основной репозиторий проектной практики: https://github.com/PervuhinRoman/Mospolytech-practice-2025-1.