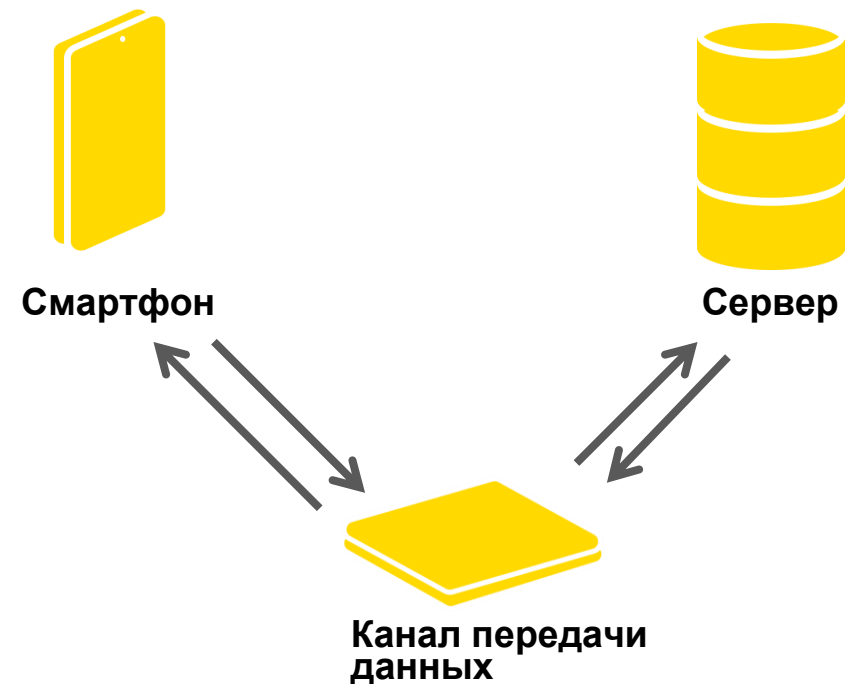


# Сетевое взаимодействие

# Введение

**Сетевое взаимодействие** позволяет приложениям обмениваться данными с удаленными серверами. Это ключевой аспект для большинства современных мобильных приложений, обеспечивающий динамическое обновление информации и взаимодействие с пользователями в реальном времени.



# Важность сетевого взаимодействия

В современных мобильных приложениях сетевое взаимодействие используется повсеместно. Оно необходимо для:

- получения данных с серверов
- отправки пользовательской информации
- синхронизации данных между устройствами и серверами

## Примеры использования:

- социальные сети загружают новые посты и фотографии
- мессенджеры отправляют и получают сообщения
- приложения для погоды получают актуальную информацию о погодных условиях


**Без сетевого взаимодействия большинство приложений потеряли бы свою функциональность и привлекательность для пользователей.**

# Что такое бэкенд?

**Бэкенд** — это серверная часть приложения, отвечающая за обработку данных и бизнес-логику.

Он принимает запросы от клиентских приложений, обрабатывает их, взаимодействует с базой данных, другими сервисами и возвращает результаты обратно клиенту. Бэкенд обеспечивает безопасность данных, управление пользователями, выполнение сложных вычислений и многое другое.

**В мобильных приложениях бэкенд играет важную роль, обеспечивая связь между клиентским приложением и серверами.**



# Компоненты бэкенда

Примеры технологий:

Node.js, Django,  
Flask, Spring

## Серверы

Серверы принимают и обрабатывают запросы от клиентских приложений. Они выполняют бизнес-логику и взаимодействуют с базой данных.

1

## Базы данных

Базы данных используются для хранения и управления данными приложения. Они обеспечивают быстрый доступ к данным и их надежное хранение.

2

## API

API (интерфейсы программирования приложений) предоставляют методы для взаимодействия клиентских приложений с сервером. Они определяют, как клиент может запрашивать данные и отправлять информацию на сервер.

3



# Клиент-серверная архитектура

Клиент-серверная архитектура предполагает взаимодействие между клиентским приложением и сервером. Клиент отправляет запросы на сервер, сервер обрабатывает их и возвращает ответы клиенту.

**Например,** мобильное приложение может отправить запрос на сервер для получения списка пользователей, сервер обработает запрос, получит данные из базы данных и отправит их обратно клиенту.

**Этот подход позволяет разделить логику приложения на клиентскую и серверную части, обеспечивая гибкость и масштабируемость.**



# Сетевые операции в Android

В Android приложения часто взаимодействуют с удаленными серверами для получения и отправки данных.

Сетевые операции должны выполняться правильно и эффективно, чтобы **обеспечить безопасность, производительность и надежность.**

Например, запросы к серверу должны выполняться асинхронно, чтобы не блокировать пользовательский интерфейс.

**Важно также обрабатывать ошибки, возникающие при сетевых операциях, и обеспечивать безопасность данных, передаваемых по сети.**



# Популярные библиотеки

Каждая библиотека имеет свои преимущества и недостатки.

***OkHttp*** и ***Retrofit*** широко используются благодаря своей гибкости и простоте использования.

## Retrofit

это библиотека для взаимодействия с REST API. Она упрощает создание HTTP-запросов, обработку ответов и сериализацию данных. Retrofit использует OkHttp под капотом.

## OkHttp

это мощная и гибкая библиотека для выполнения HTTP-запросов. Она поддерживает синхронные и асинхронные запросы, обработку ответов, кэширование и многое другое.

## Volley

это библиотека, разработанная Google для выполнения сетевых операций в Android. Она поддерживает выполнение запросов, кэширование, обработку ошибок и более сложные функции, такие как загрузка изображений.





# Обработка сетевых операций

## Асинхронное программирование:

В Android важно выполнять сетевые операции асинхронно, чтобы не блокировать основной поток и не ухудшать пользовательский опыт. Для этого можно использовать AsyncTask, Handler, Executor или современные подходы, такие как Kotlin Coroutines и RxJava.

## Обработка ошибок:

При выполнении сетевых операций могут возникать различные ошибки, такие как отсутствие сети, тайм-ауты, ошибки сервера. Эти ошибки должны обрабатываться корректно, чтобы приложение могло информировать пользователя и предпринимать соответствующие действия.

## Безопасность:

Передача данных по сети должна быть безопасной. Для этого используется HTTPS, а также дополнительные методы шифрования и аутентификации.



# Что такое REST API

REST API (Representational State Transfer Application Programming Interface) — это архитектурный стиль, определяющий взаимодействие между клиентом и сервером через HTTP-протокол.

REST API использует стандартные HTTP-методы (GET, POST, PUT, DELETE) для выполнения операций над ресурсами, представленными в виде URL.

## Основные принципы *REST API* включают:

- stateless (каждый запрос содержит всю необходимую информацию и не зависит от предыдущих запросов)
- кэширование (повышает производительность за счет использования кеша)
- использование стандартных HTTP-кодов состояния для обозначения результата операций.

# Основные концепции REST API

## URL как идентификатор ресурса

Каждый ресурс в REST API имеет уникальный URL. Например: URL <https://api.example.com/users> может использоваться для получения списка пользователей.

## Использование HTTP-методов

- GET для получения данных
- POST для создания новых ресурсов
- PUT для обновления существующих ресурсов
- DELETE для удаления ресурсов.

## Примеры использования:

- Получение списка пользователей (GET <https://api.example.com/users>)
- Создание нового пользователя (POST <https://api.example.com/users>)
- Удаление пользователя (DELETE <https://api.example.com/users/1>)



# Библиотека OkHttp

OkHttp — это мощная и гибкая библиотека для выполнения HTTP-запросов.

Она поддерживает синхронные и асинхронные запросы, обработку ответов, кэширование, управление соединениями и многое другое.

**Пример использования *OkHttp*:** создание HTTP-клиента, выполнение GET-запроса и обработка ответа.

OkHttp позволяет легко выполнять сетевые операции и обрабатывать ответы от сервера.

# Пример: OkHttp

NetworkClient.kt

Kotlin

```
val client = OkHttpClient()

val request = Request.Builder()
    .url("https://api.example.com/data")
    .build()

client.newCall(request).enqueue(object : Callback {
    override fun onFailure(call: Call, e: IOException) {
        e.printStackTrace()
    }

    override fun onResponse(call: Call, response: Response) {
        if (response.isSuccessful) {
            val responseData = response.body?.string()
            // обработка ответа
        }
    }
})
```



# Библиотека Retrofit

**Retrofit** — это библиотека для взаимодействия с REST API. Она упрощает создание HTTP-запросов, обработку ответов и сериализацию данных. Retrofit использует OkHttp под капотом, предоставляя удобный интерфейс для работы с API.

Пример использования Retrofit: создание интерфейса API, выполнение запроса и обработка ответа. Retrofit позволяет легко определять методы API и выполнять запросы к серверу



# Библиотека Retrofit

*Retrofit* — это библиотека для взаимодействия с REST API. Она упрощает создание HTTP-запросов, обработку ответов и сериализацию данных. Retrofit использует OkHttp под капотом, предоставляя удобный интерфейс для работы с API.

**Пример использования *Retrofit*:** создание интерфейса API, выполнение запроса и обработка ответа.

Retrofit позволяет легко определять методы API и выполнять запросы к серверу.

# Пример: использование библиотеки Retrofit

NetworkCommunication.kt

Kotlin

```
interface ApiService {
    @GET("users/{user}/repos")
    fun listRepos(@Path("user") user: String): Call<List<Repo>>
}

val retrofit = Retrofit.Builder()
    .baseUrl("https://api.example.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

val service = retrofit.create(ApiService::class.java)

val repos = service.listRepos("username")
repos.enqueue(object : Callback<List<Repo>> {
    override fun onResponse(call: Call<List<Repo>>, response: Response<List<Repo>>) {
        if (response.isSuccessful) {
            val reposList = response.body()
            // Обработка ответа
        }
    }

    override fun onFailure(call: Call<List<Repo>>, t: Throwable) {
        t.printStackTrace()
    }
})
```





# Interceptor в сетевых запросах

**Interceptors** в OkHttp и Retrofit используются для перехвата и изменения запросов и ответов. Они позволяют добавлять заголовки, логировать запросы, кэшировать данные и многое другое.

Существует два типа интерсепторов: **Application Interceptors** и **Network Interceptors**. **Application Interceptors** работают на уровне приложений и могут изменять запросы перед отправкой. **Network Interceptors** работают на уровне сети и могут изменять запросы и ответы на уровне протокола.

# Пример: Interceptor #1

LoggingNetworkClient.kt

Kotlin

```
val loggingInterceptor = HttpLoggingInterceptor().apply {  
    level = HttpLoggingInterceptor.Level.BODY  
}  
  
val client = OkHttpClient.Builder()  
    .addInterceptor(loggingInterceptor)  
    .build()
```

# Пример: Interceptor #2

HeadersInterceptor.kt

Kotlin

```
// Кастомный интерсептор для добавления заголовков ко всем запросам
class HeadersInterceptor : Interceptor {
    @Throws(IOException::class)
    override fun intercept(chain: Interceptor.Chain): Response {
        val originalRequest = chain.request()

        // Создание нового запроса с добавлением кастомных заголовков
        val newRequest: Request = originalRequest.newBuilder()
            .header("Authorization", "Bearer your_token")
            .header("User-Agent", "TE_App")
            .build()

        return chain.proceed(newRequest)
    }
}

// Создание клиента OkHttpClient с кастомным интерсептором
val client = OkHttpClient.Builder()
    .addInterceptor(HeadersInterceptor())
    .build()
```



# Настройка безопасности сети

**Network Security Config** позволяет настроить безопасность сетевых подключений в Android. Она предоставляет возможности для настройки доверенных сертификатов, шифрования данных и других параметров безопасности.

# Пример:

## Network Security Config

network\_security\_config.xml

XML

```
<network-security-config>
  <domain-config cleartextTrafficPermitted="false">
    <domain includeSubdomains="true">example.com</domain>
    <trust-anchors>
      <certificates src="system" />
      <certificates src="user" />
    </trust-anchors>
  </domain-config>
</network-security-config>
```

AndroidManifest.xml

XML

```
<application
  android:networkSecurityConfig="@xml/network_security_config"
  ... >
  ...
</application>
```

# Кеширование сетевых данных

Кеширование позволяет улучшить производительность приложения, снижая количество сетевых запросов и ускоряя доступ к данным.

OkHttp поддерживает кеширование ответов на уровне HTTP-заголовков.

CacheNetworkClient.kt

Kotlin

```
val cacheSize = 10 * 1024 * 1024 // 10 MB
val cache = Cache(context.cacheDir, cacheSize)

val client = OkHttpClient.Builder()
    .cache(cache)
    .build()
```



# Работа с WebSocket

**WebSocket** — это протокол, обеспечивающий двустороннее общение между клиентом и сервером в реальном времени. Он особенно полезен для чатов, онлайн-игр и других приложений, требующих постоянного обмена данными.

OkHttp поддерживает WebSocket, позволяя легко создавать и управлять WebSocket-соединениями.

# Пример: WebSocket

WebSocket.kt

Kotlin

```
val client = OkHttpClient()

val request = Request.Builder()
    .url("wss://echo.websocket.org")
    .build()

val websocketListener = object : WebSocketListener() {
    override fun onOpen(webSocket: WebSocket, response: Response) {
        webSocket.send("Hello, World!")
    }

    override fun onMessage(webSocket: WebSocket, text: String) {
        println("Received: $text")
    }

    override fun onFailure(webSocket: WebSocket, t: Throwable, response: Response?) {
        t.printStackTrace()
    }
}

val webSocket = client.newWebSocket(request, websocketListener)
client.dispatcher.executorService.shutdown()
```



# Настройка приложения

Для выполнения сетевых операций в вашем приложении AndroidManifest.xml должен содержать разрешение INTERNET.

Никакой код и никакие библиотеки не заставят приложение ходить в сеть пока не будет выполнена первоначальная настройка.

AndroidManifest.xml

Kotlin

```
<uses-permission android:name="android.permission.INTERNET" />
```



True Engineering

630128, г. Новосибирск,  
ул. Кутателадзе, 4г

(383) 363-33-51, 363-33-50  
info@trueengineering.ru  
trueengineering.ru

**Новосибирский  
Государственный  
Университет**