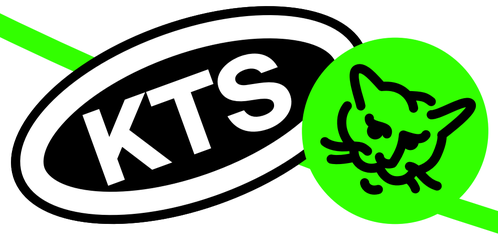
ОТ TELEGRAM ДО ОДНОКЛАССНИКОВ: СЕКРЕТЫ ВСТРАИВАНИЯ ПРИЛОЖЕНИЙ КУДА УГОДНО



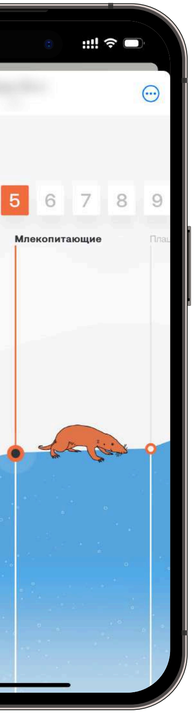
Спикер — Надежда Меркулова

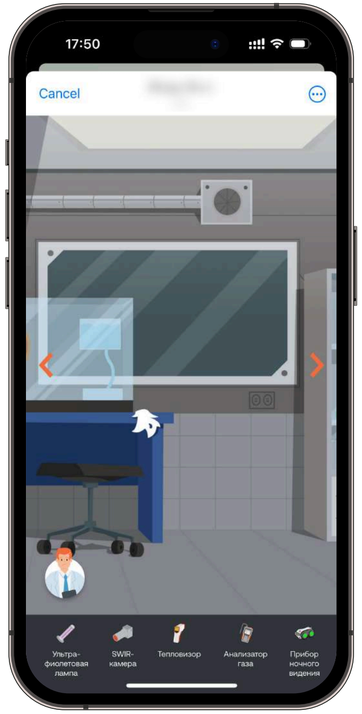
Эксперт — Федор Биличенко

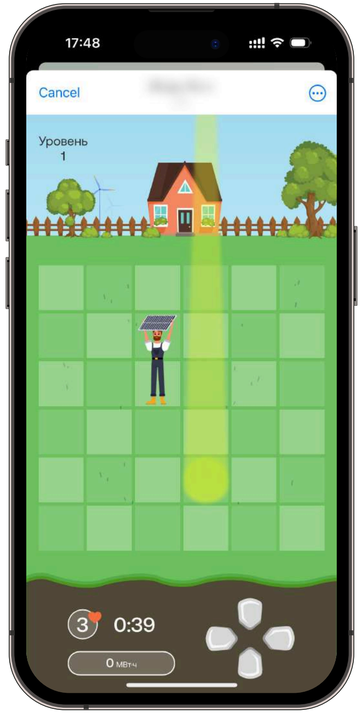
ОБО МНЕ

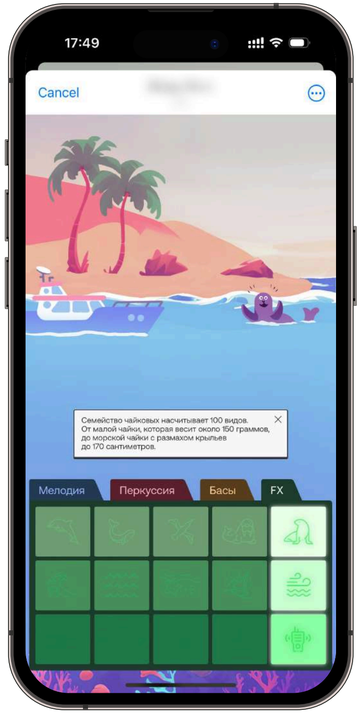
1. Опыт разработки 5 лет
2. Выпускница Технопарка VK в МГТУ им Баумана
3. Разработала своими руками 50+ проектов
4. Тимлид отдела рекламных спецпроектов в KTS

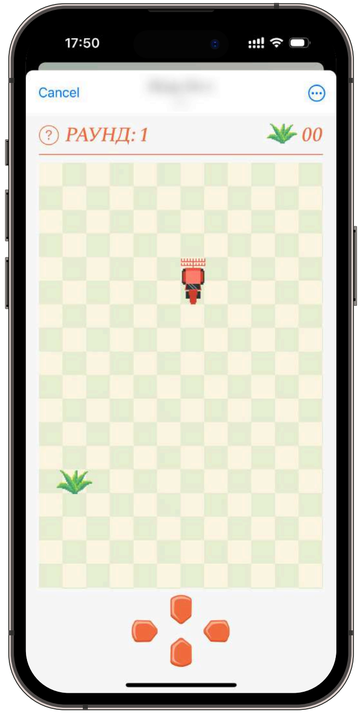
60 ИГР ЗА ПОЛГОДА

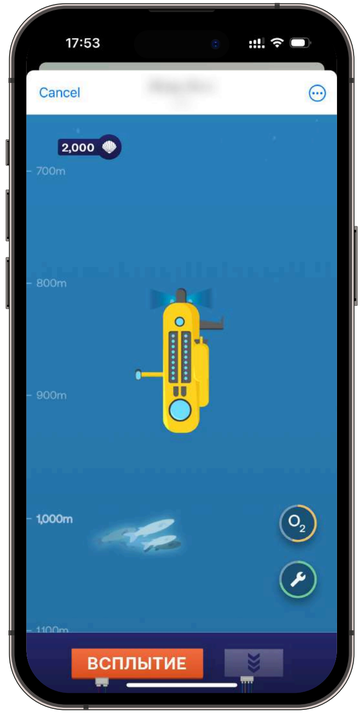


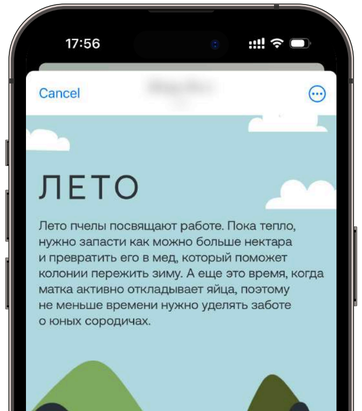


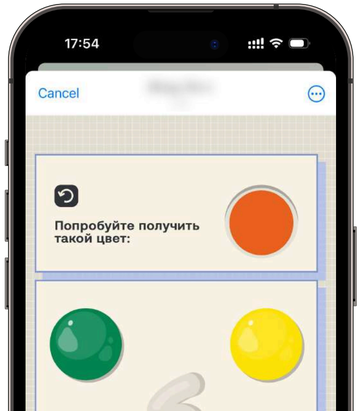


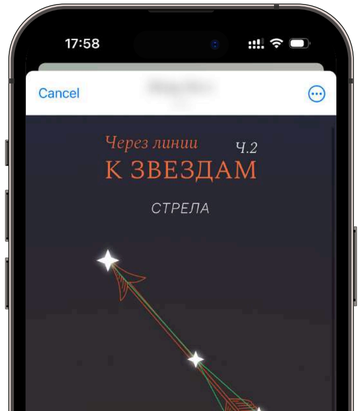


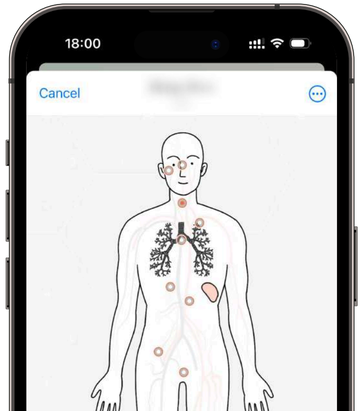


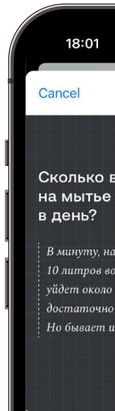










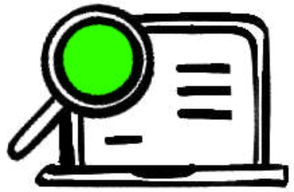


ЗАДАЧА

1. Разработать 60 игр
2. Встроить каждую в 4 окружения — статья на сайте, мини-аппы во ВКонтакте, Одноклассниках и Telegram
3. Настроить для каждой статистику и шеринги
4. Сделать все за полгода
5. Выпускать по 10 проектов в месяц

ЗАЧЕМ?

Разнообразить контент



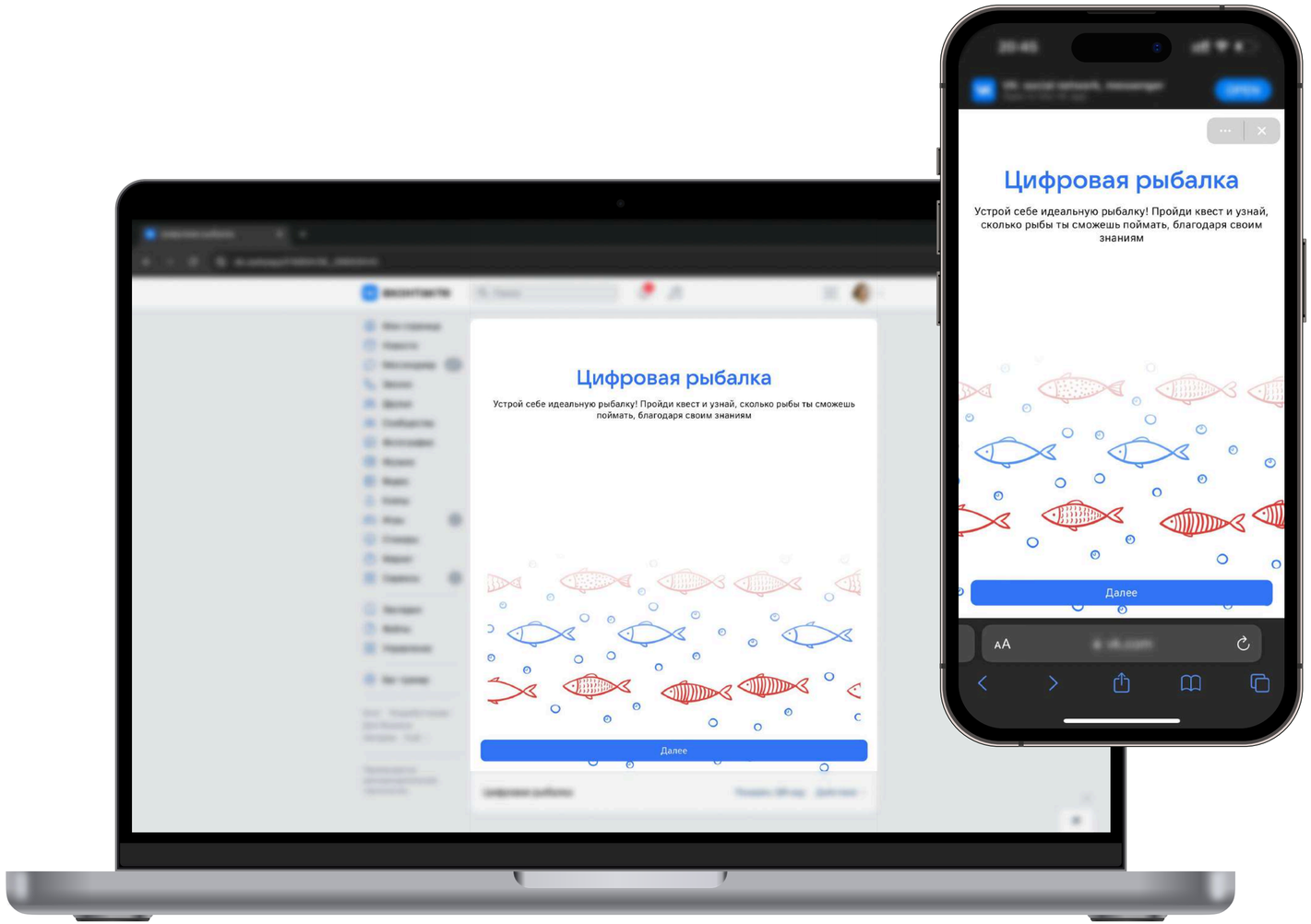
Увеличить охваты

Снизить затраты

Поддержать экосистемность

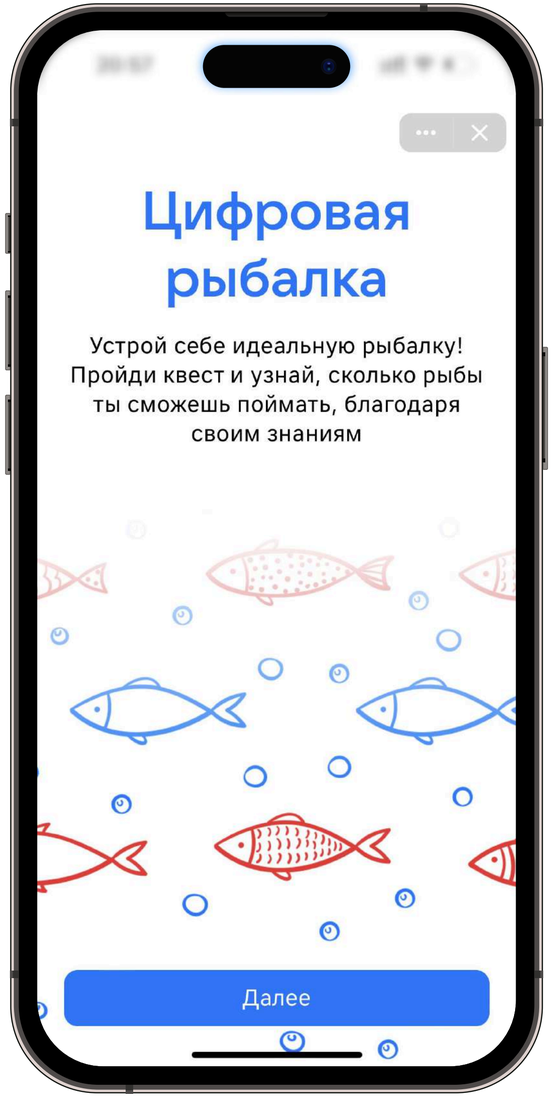
ЧТО ТАКОЕ ВСТРАИВАЕМЫЕ → ПРИЛОЖЕНИЯ?

ПРИЛОЖЕНИЯ, ВСТРАИВАЕМЫЕ ЧЕРЕЗ IFRAME

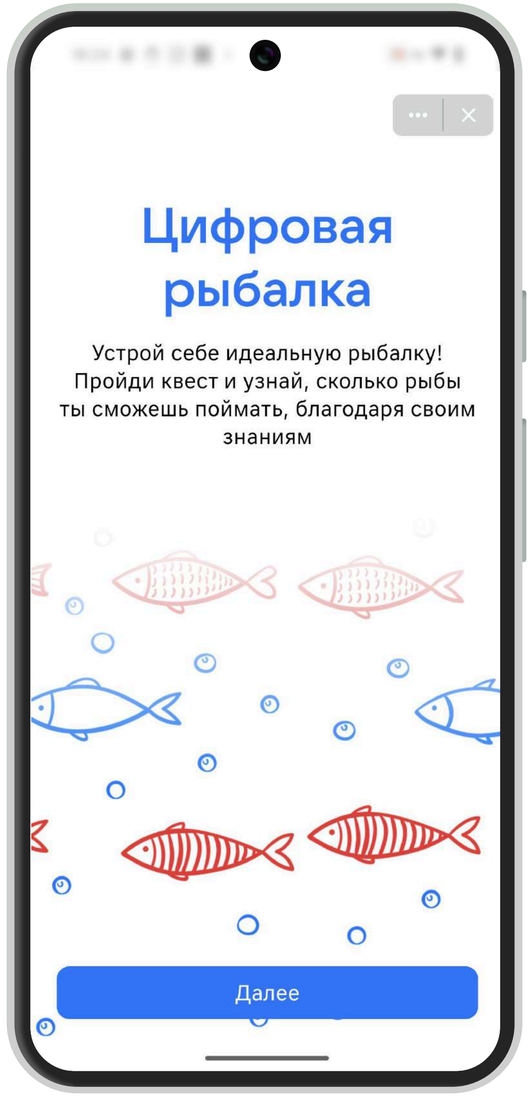


WEB DESKTOP / WEB MOBILE

ПРИЛОЖЕНИЯ, ВСТРАИВАЕМЫЕ ЧЕРЕЗ WEBVIEW



IOS



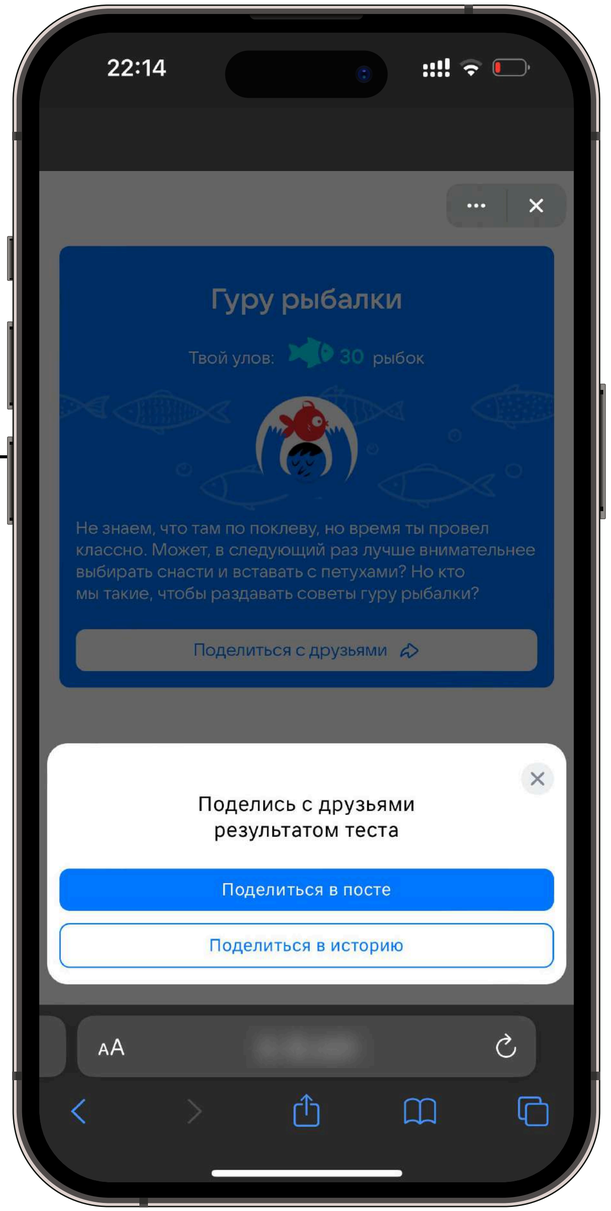
ANDROID

КАК РАБОТАЮТ ВСТРАИВАЕМЫЕ → ПРИЛОЖЕНИЯ?

ОТПРАВКА СОБЫТИЙ ИЗ IFRAME ИЛИ WEBVIEW

По клику на кнопку отправляем во внешнее окружение сообщение с запросом на открытие окна шеринга:

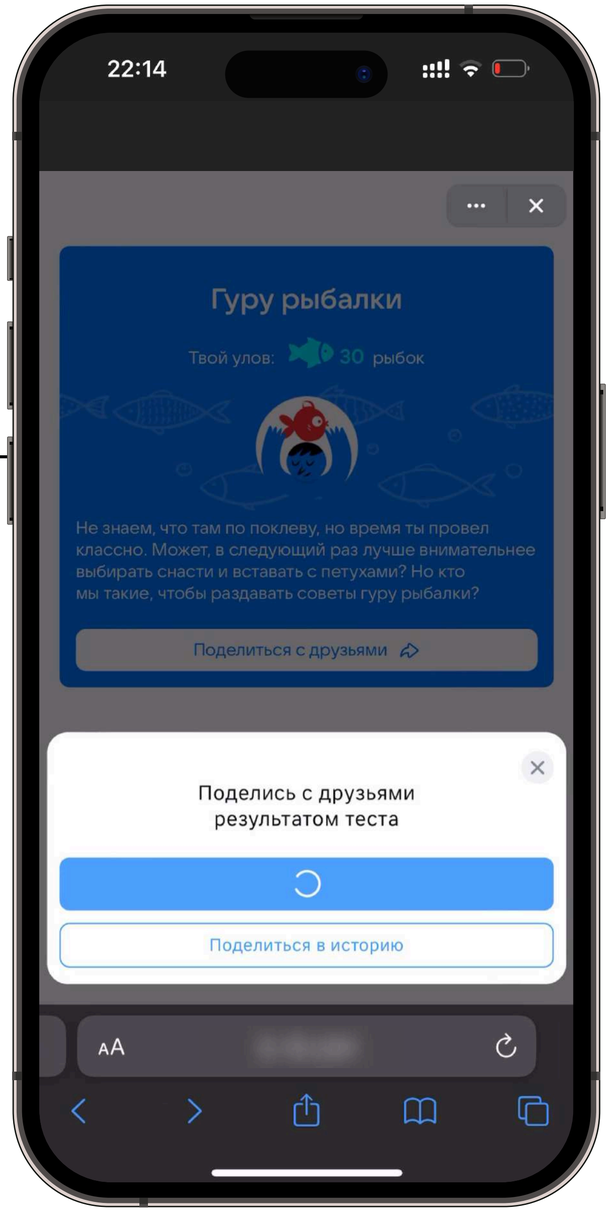
```
postMessageFromIframe(JSON.stringify({
  event: 'share',
  payload: {
    image: 'https://my-cdn/image-to-share.png',
  }
});
```



ОТПРАВКА СОБЫТИЙ ИЗ IFRAME ИЛИ WEBVIEW

В функции postMessage выбираем способ отправки сообщения в зависимости от текущей платформы:

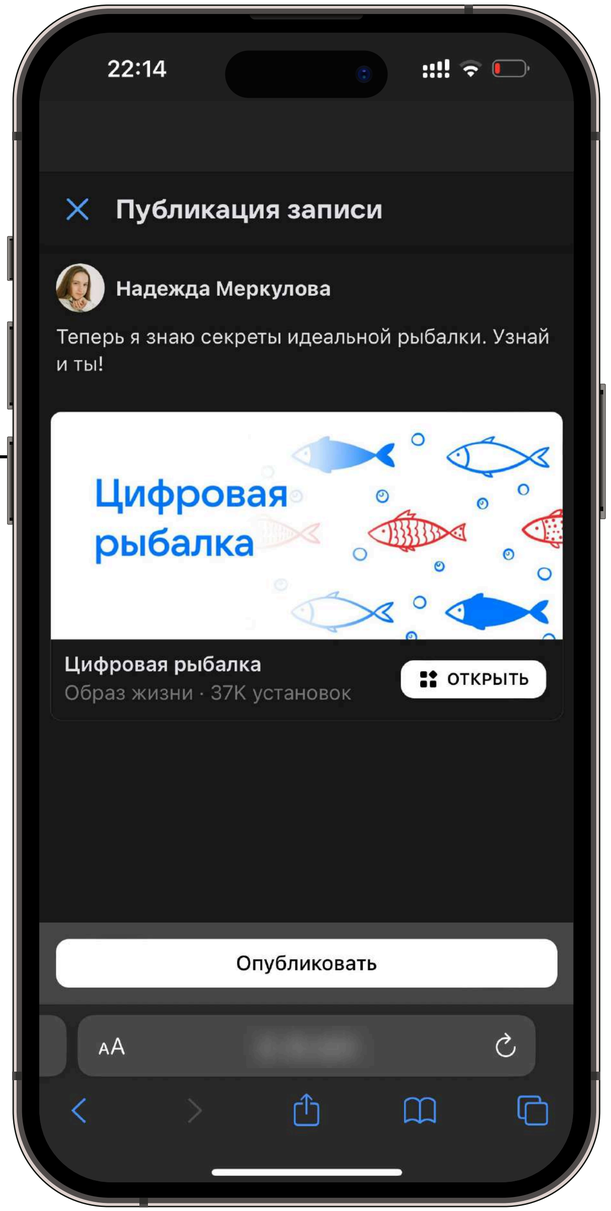
```
const postMessageFromIframe = (message: string) => {
  // IOS
  if (window.webkit?.messageHandlers?.eventsHandler?.postMessage) {
    window.webkit.messageHandlers.eventsHandler.postMessage(message);
    return;
  }

  // Android
  if (window.Android?.postMessage) {
    window.Android.postMessage(message);
    return;
  }

  // Web
  window.parent.postMessage(message, PARENT_ORIGIN);
}
```



ПОЛУЧЕНИЕ СОБЫТИЙ ИЗ IFRAME

Во внешнем окружении ловим сообщение от iframe, обрабатываем и в результате открываем окно с шерингом:

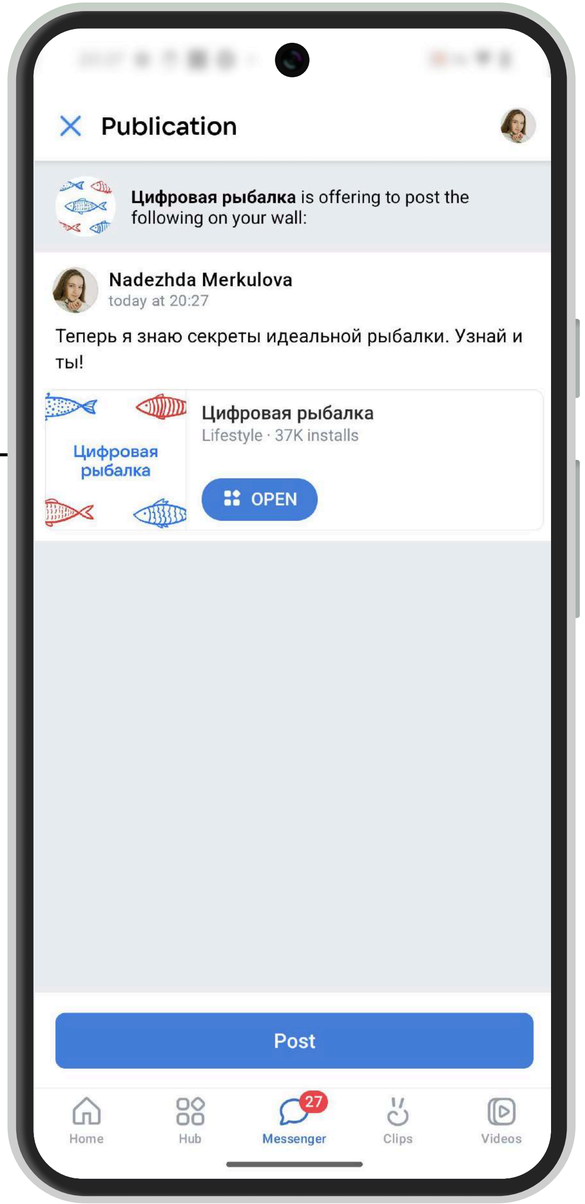
```
window.addEventListener('message', (event: MessageEvent) => {
  if (event.origin !== IFRAME_ORIGIN) {
    return;
  }

  handleMessageFromIframe(event.data);
});
```



ПОЛУЧЕНИЕ СОБЫТИЙ ИЗ WEBVIEW

Чтобы ловить сообщения от webview на Android, внутрь webview добавляем window.Android.postMessage

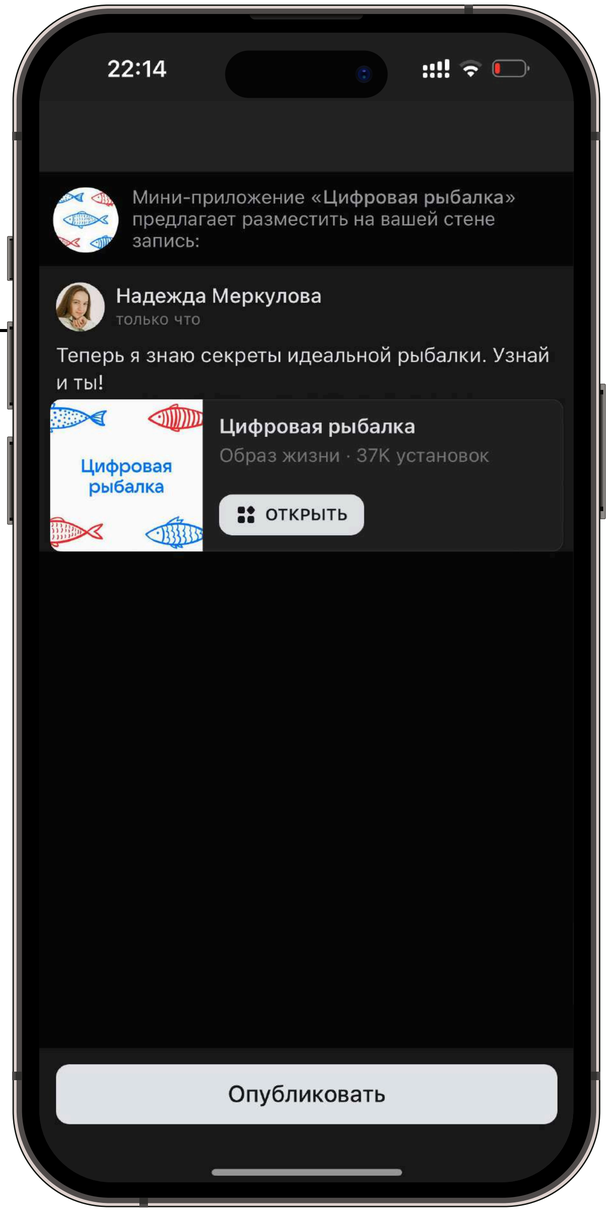
```
class WebAppInterface() {
    @JavascriptInterface
    fun postMessage(message: String) {
        handleMessageFromWebView(message)
    }
}

webView.addJavascriptInterface(WebAppInterface(), "Android")
```



ПОЛУЧЕНИЕ СОБЫТИЙ ИЗ WEBVIEW

Чтобы ловить сообщения от webview на IOS,
внутрь webview добавляем
window.webkit.messageHandlers.eventsHandler.postMessage

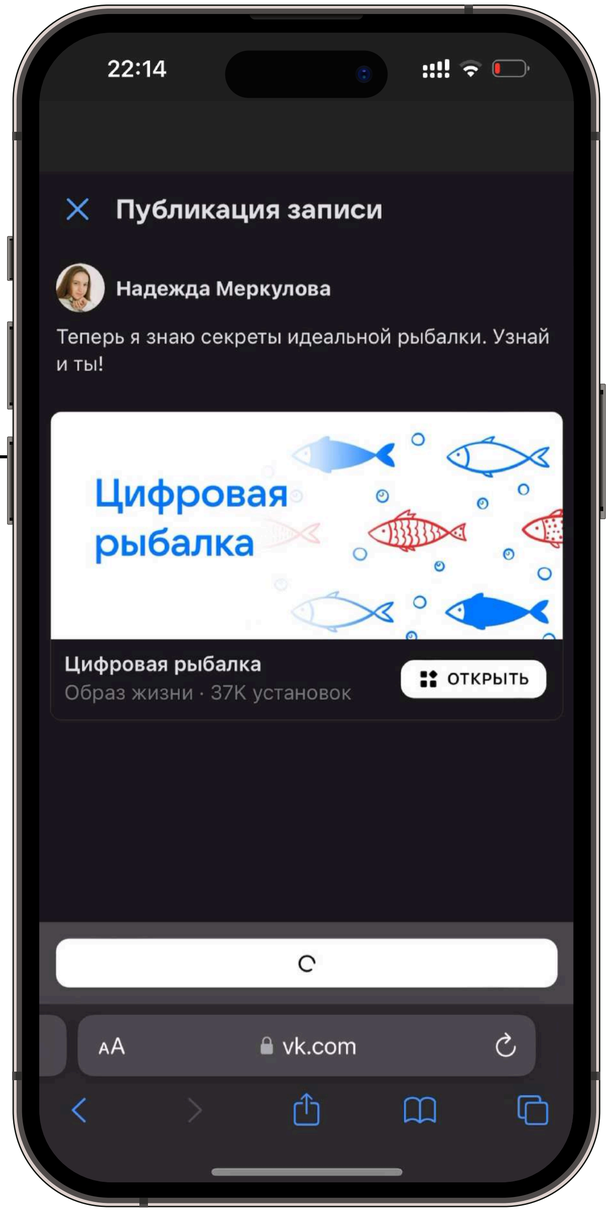
```
final class Example: NSObject, WKScriptMessageHandler {

    lazy var webView = WKWebView()

    override init() {
        super.init()
        setupWebView()
    }

    func setupWebView() {
        let userContentController = webView.configuration.userContentController
        userContentController.add(self, name: "eventsHandler")
    }

    func userContentController(
        _ userContentController: WKUserContentController,
        didReceive message: WKScriptMessage
    ) {
        if message.name == "eventsHandler" {
            handleMessageFromWebView(message.body)
        }
    }
}
```



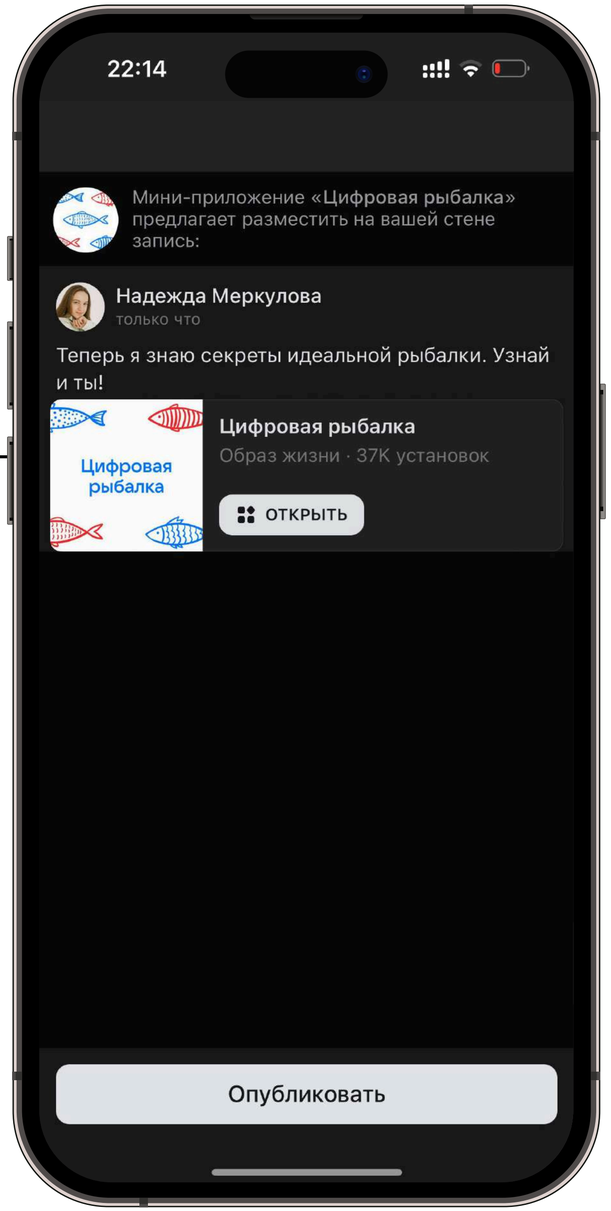
ОТПРАВКА СОБЫТИЙ В IFRAME

Когда пользователь пошерил пост во внешней среде, так же через postMessage отправляем событие с результатом и снова открываем iframe

```
postMessageToIframe(JSON.stringify({
  event: 'share-result',
  payload: {
    status: 'success',
  }
});

const postMessageToIframe = (message: string) => {
  iframe.contentWindow.postMessage(message, IFRAME_ORIGIN);
}
```



ОТПРАВКА СОБЫТИЙ В WEBVIEW



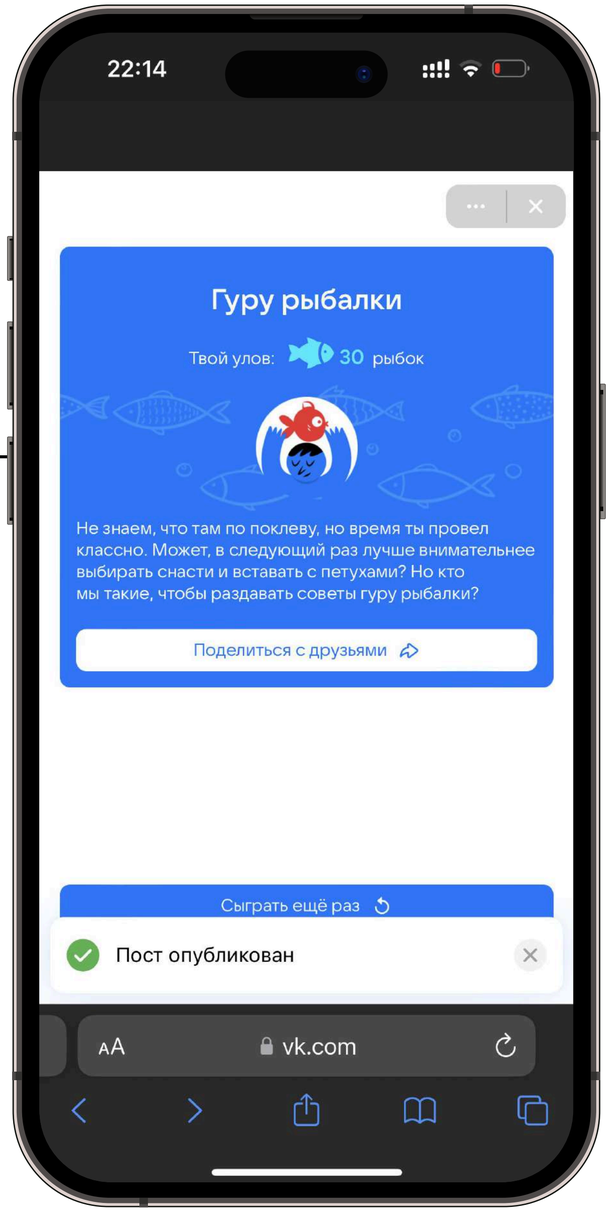
В мобильных приложениях через evaluateJavaScript вызываем функцию из webview, отвечающую за обработку сообщения, и передаем в нее сообщение о результате шеринга

```
func postMessageToWebView(message: String) {
    webView.evaluateJavaScript("window.handleMessageFromMobile(\(message));")
}
```

ПОЛУЧЕНИЕ СОБЫТИЙ В IFRAME ИЛИ WEBVIEW

Внутри iframe или webview ловим результат от внешнего окружения и показываем, что шеринг был успешным

```ts
const handleMessage = (message: string) => {
  const parsedMessage = JSON.parse(message);

  if (parsedMessage.event === 'share-result') {
    handleShareResult(parsedMessage.payload);
  }
}

// IOS / Android
window.handleMessageFromMobile = handleMessage;

// Web
const handleMessageFromParent = (event: MessageEvent<string>) => {
  if (event.origin !== PARENT_ORIGIN) {
    return;
  }

  handleMessage(event.data);
}

window.addEventListener('message', handleMessageFromParent);
```



ЧТО ПРЕДЛАГАЮТ СЕРВИСЫ?

ВКОНТАКТЕ

- VK Mini Apps

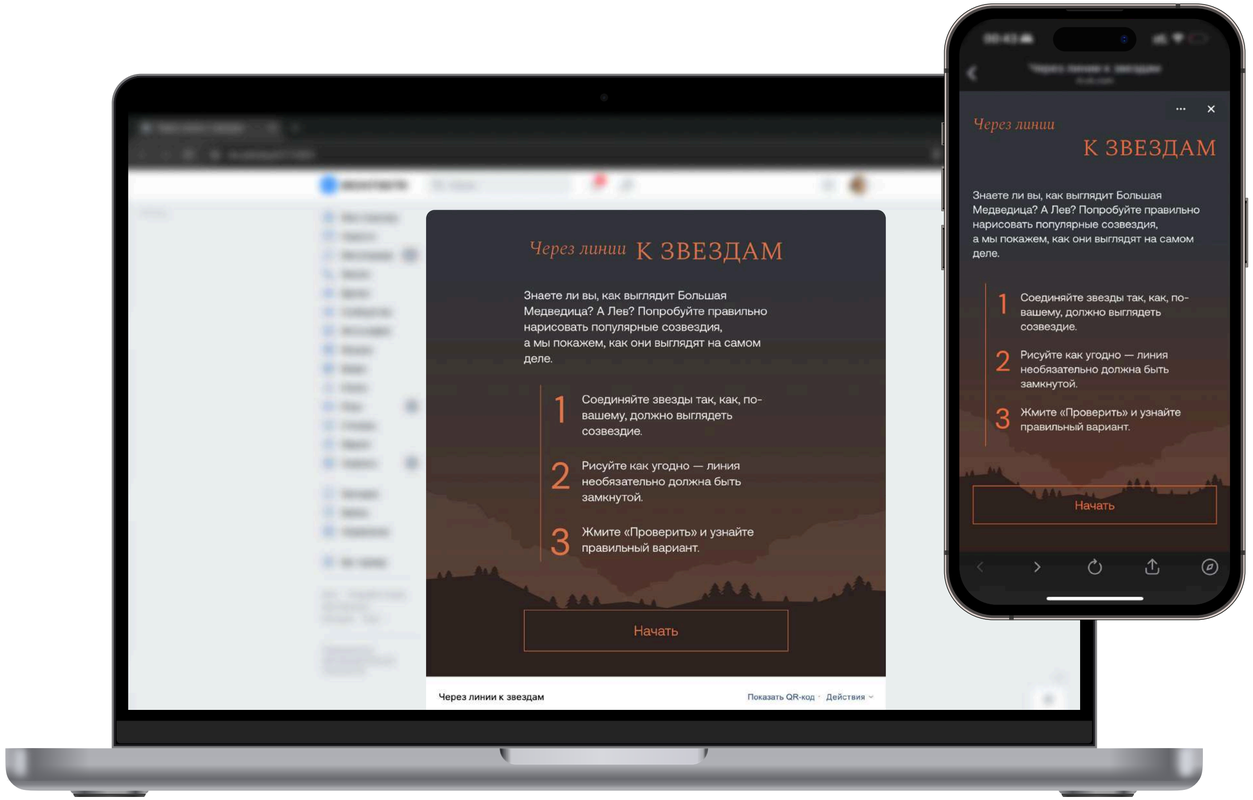
ОДНОКЛАССНИКИ

- Платформа приложений
- VK Mini Apps
  (с конца 2020 года)

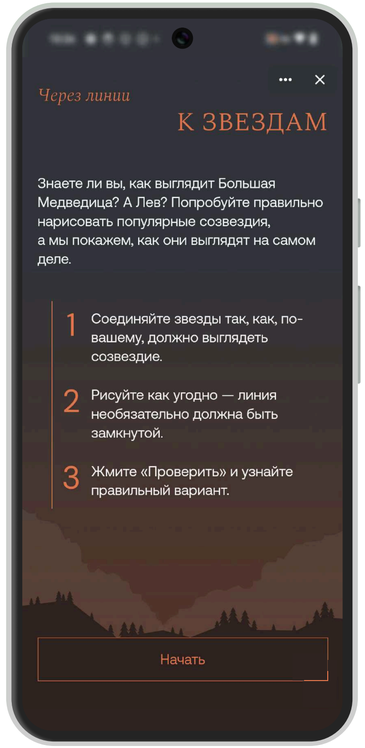
TELEGRAM

- Telegram ~~Web~~ Mini Apps

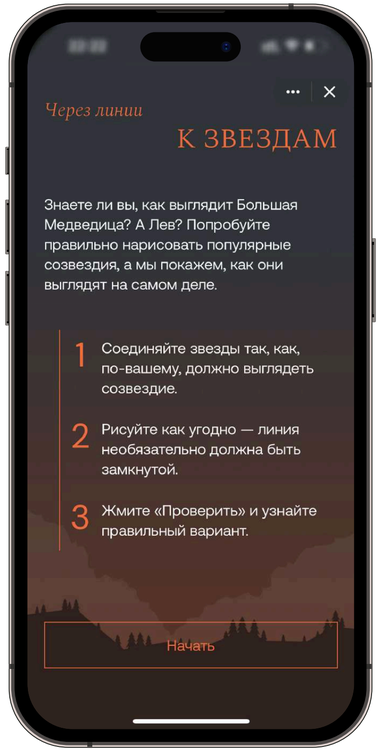
ВСТРАИВАНИЕ ВО ВКОНТАКТЕ



WEB DESKTOP / MOBILE



ANDROID APP

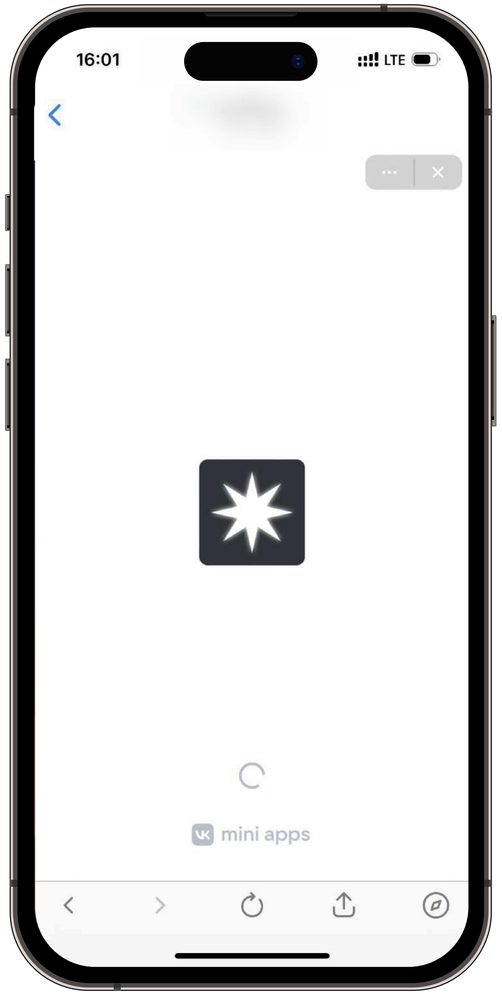


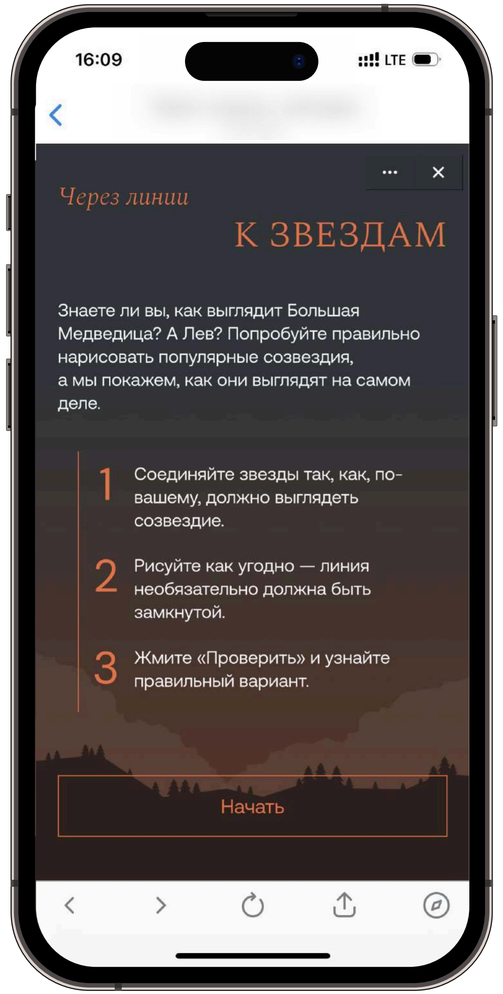
IOS APP

ИНИЦИАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

Подключить vk-bridge через npm-пакет или скрипт и отправить событие VKWebAppInit

```
vkBridge.send('VKWebAppInit', {})
  .then((data) => {
    if (data.result) {
        // Приложение инициализировано
    } else {
        // Ошибка
    }
  })
  .catch(() => {
    // Ошибка
  })
```



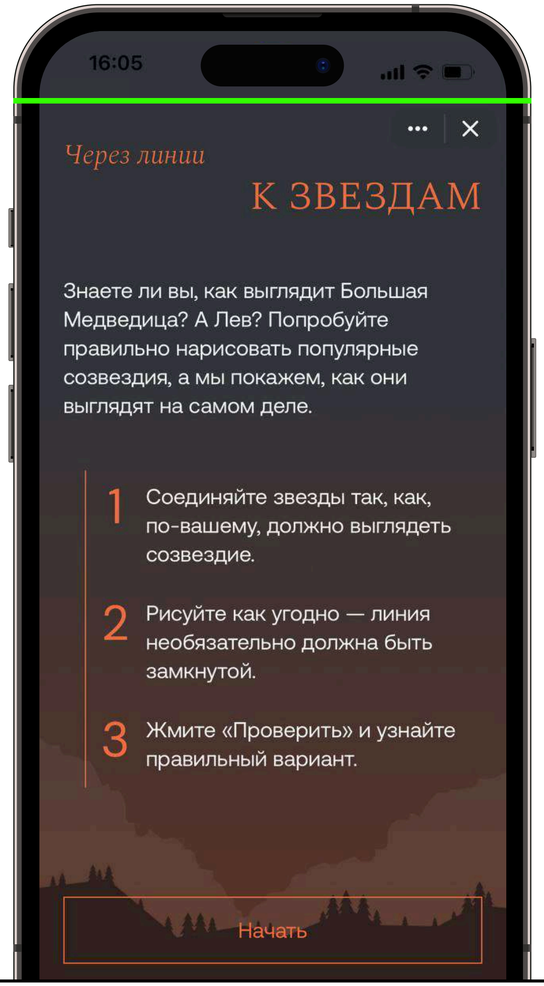


НАСТРОЙКА СТАТУС-БАРА

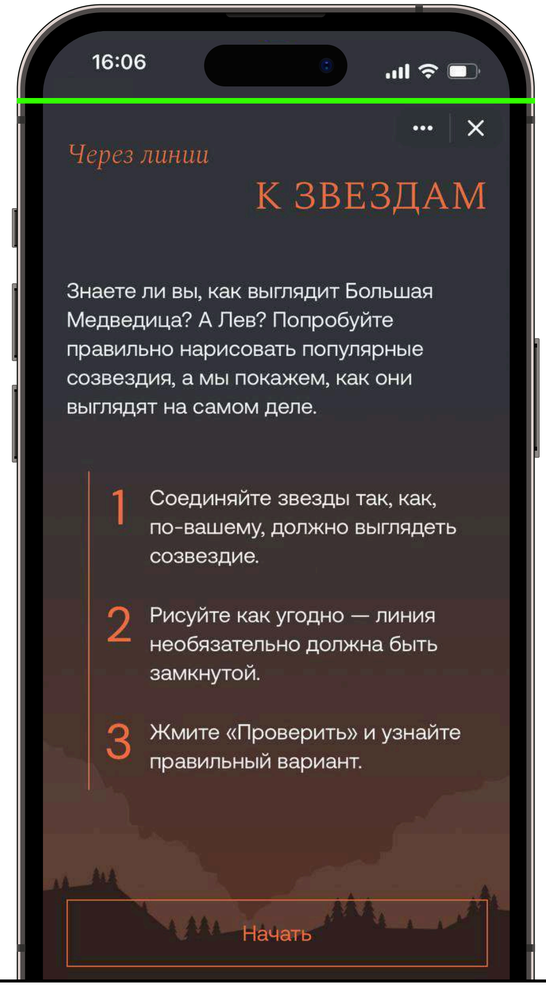
Настроить навигационный и статус-бары, отправив событие VKWebAppSetViewSettings

```
if (!vkBridge.supports('VKWebAppSetViewSettings')) {
  return;
}

vkBridge.send('VKWebAppSetViewSettings', {
  status_bar_style: 'dark',
  action_bar_color: '#FFF',
  navigation_bar_color: '#FFF',
});
```

ДО ВЫЗОВА



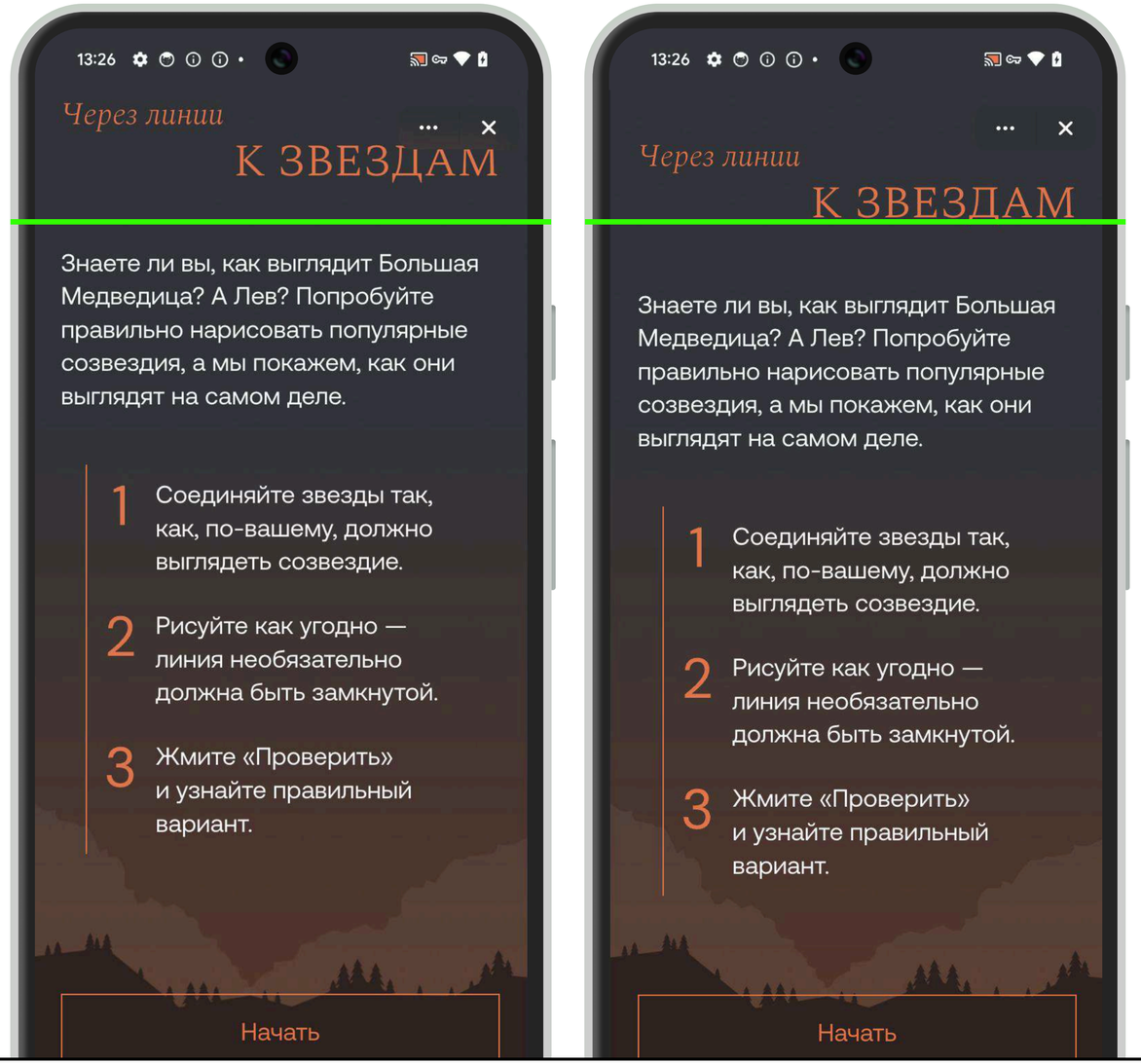
ПОСЛЕ



ПРОВЕРКА СИСТЕМНЫХ ОТСТУПОВ

1. Проверить, что отступы сверху и снизу выглядят нормально
2. Учесть позицию кнопки закрытия приложения, чтобы они не накладывались на контент

```
.container {
    // Отступ устройства + Отступ кнопок + Половина высоты кнопок
    padding-top: calc(var(--vkui_internal--safe_area_inset_top) +
10px + 32px * 0.5);

    // Отступ устройства
    padding-bottom: var(--vkui_internal--safe_area_inset_bottom);
}
```



VK-MINI-APPS НА IOS ИЛИ ЧТО ТАКОЕ ODR-АРХИВ

По умолчанию на IOS мини-приложение открывается в мобильном браузере, а не в WKWebView.

Чтобы этого не возникало, необходимо соблюсти требование Apple по использованию технологии ODR.

СБОРКА ODR-АРХИВА

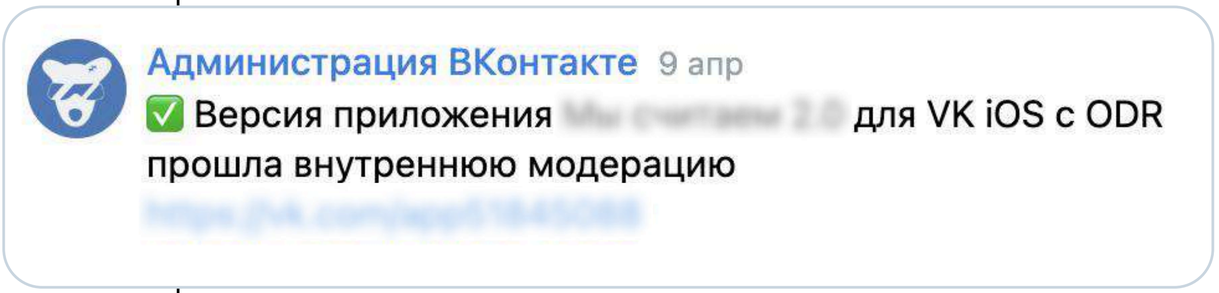
Адаптировать код приложения под требования ODR-архива, а именно:

1. Вынести на верхний уровень бандла index.html
2. Использовать относительные или полные пути к статике
3. Настроить CORS (звездочка или протокол *vkcors*)
4. Убедиться, что не превышен лимит на размер — 40 МБ
5. Использовать браузерный роутинг с осторожностью или заменить его
6. Для внесения изменений в код чисто под ODR ориентироваться на параметр odr_enabled

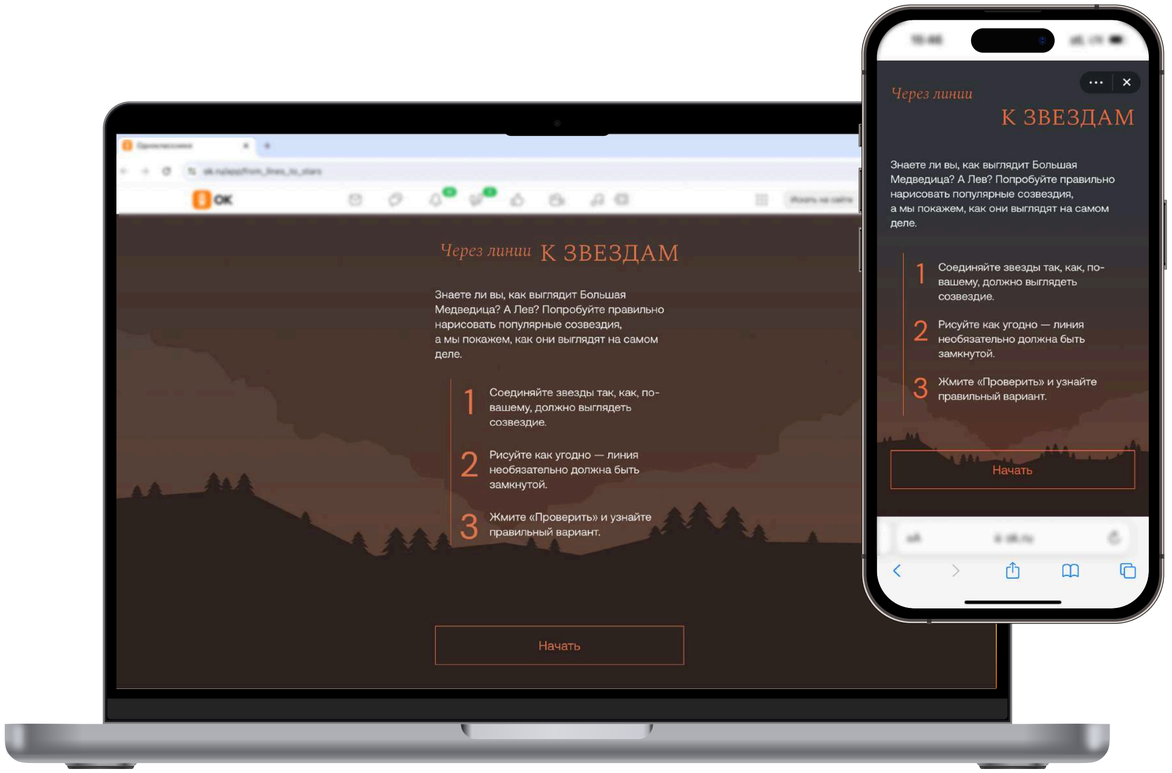
ОТПРАВКА ODR-АРХИВА

Чтобы архив попал в сборку IOS-приложения ВКонтакте, загрузить отправить архив на модерацию:

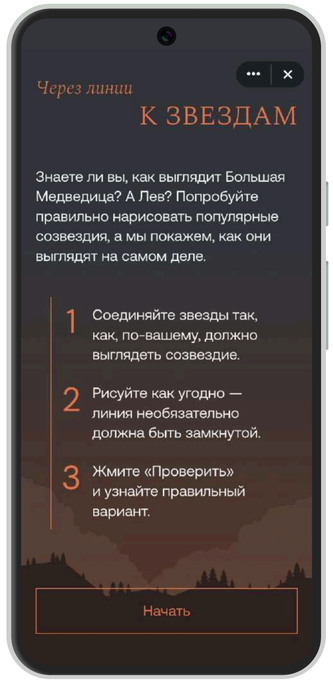
1. Заполнить данные разработчика Apple в настройках
2. Собрать и загрузить zip-архив
3. Потестировать архив (через режим “Тестирование ODR” в настройках)
4. Отправить архив на модерацию
5. Получить ответ от модерации через некоторое время и внести правки при необходимости



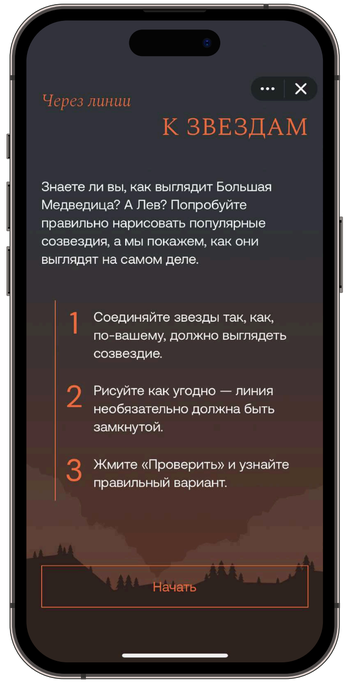
ВСТРАИВАНИЕ В ОДНОКЛАССНИКИ



WEB DESKTOP / MOBILE



ANDROID APP



IOS APP

ЭКСПОРТ ИЗ ВКОНТАКТЕ В ОДНОКЛАССНИКИ

Решить, подойдет ли встраивание экспортом из VK Mini Apps. Если да, то:

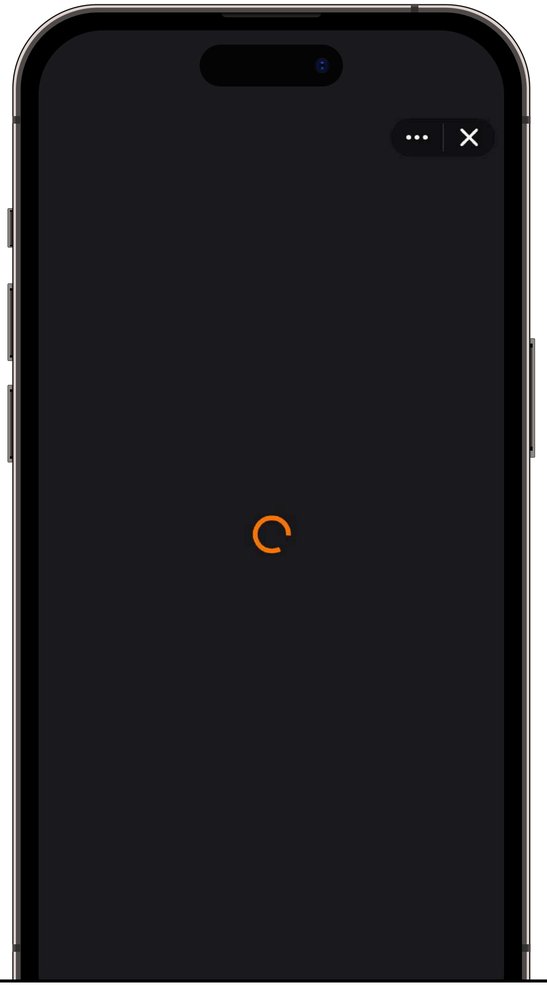
1. Учесть, что экспортировать приложения может только создатель
2. Проверить поддерживаемые методы vk-bridge
3. Немного подождать, пока произойдет экспорт
4. Поменять настройки экспортированного приложения при необходимости
5. Для внесения изменений в коде под Одноклассники ориентироваться на параметр vk_client=ok

ИНИЦИАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

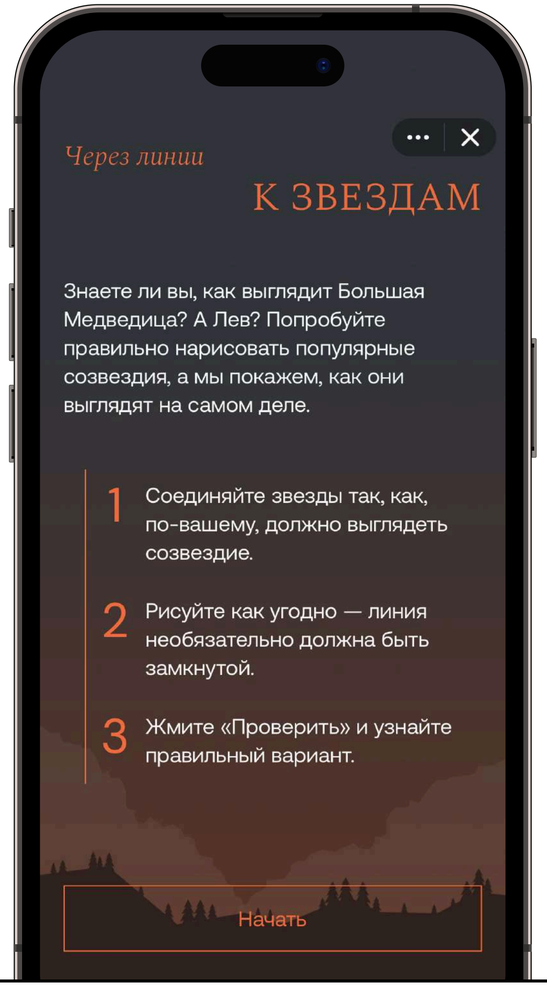
Если не подойдет встраивание экспортом не подходит, то подключить SDK, получить параметры запуска из метода FAPI.Util.getRequestParameters и передать в метод FAPI.init

```
const initializeOkApp = async (): Promise<void> => {
  const requestParameters = FAPI.Util.getRequestParameters();

  return new Promise((resolve, reject) => {
    FAPI.init(
      requestParameters.api_server,
      requestParameters.apiconnection,
      resolve,
      reject
    );
  });
};
```

ДО ВЫЗОВА



ПОСЛЕ

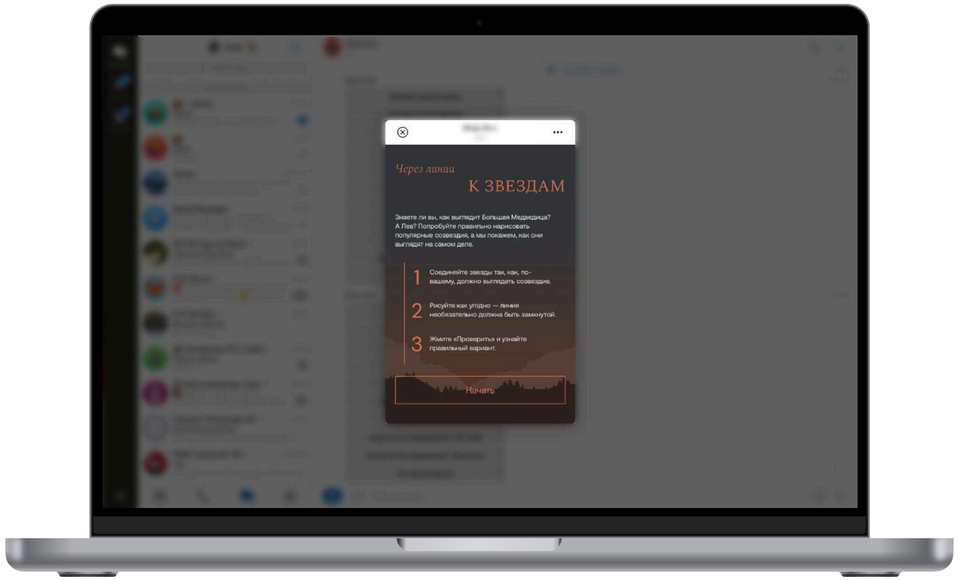


РАБОТА С FAPI

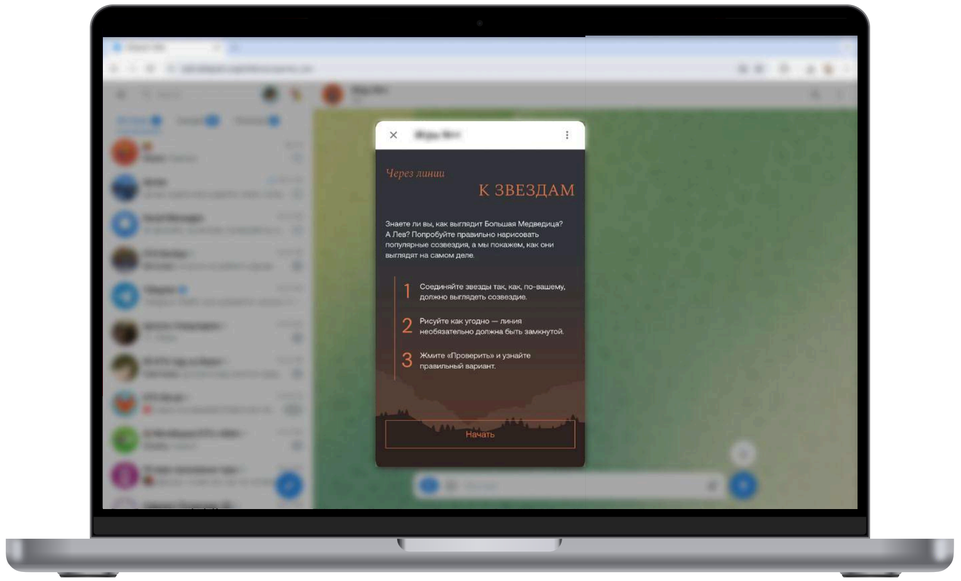
1. Ознакомиться со списком доступных методов и в случае, если необходим метод из группы FAPI.UI, создать глобально функцию API_callback
2. Написать типы для основных методов на основе документации

```
import { OKMiniApp } from './types';

declare global {
  const FAPI: {
    init: OKMiniApp.FAPIInit;
    Client: OKMiniApp.FAPIClient;
    Util: OKMiniApp.FAPIUtil;
  };

  interface Window {
    API_callback: OKMiniApp.FAPIApiCallback;
  }
}
```

ВСТРАИВАНИЕ В ТЕЛЕГРАМ



DESKTOP APP



DESKTOP WEB

MOBILE APP

ИНИЦИАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

1. Подключить SDK и вызвать Telegram.WebApp.ready
2. Подключить глобальную типизацию

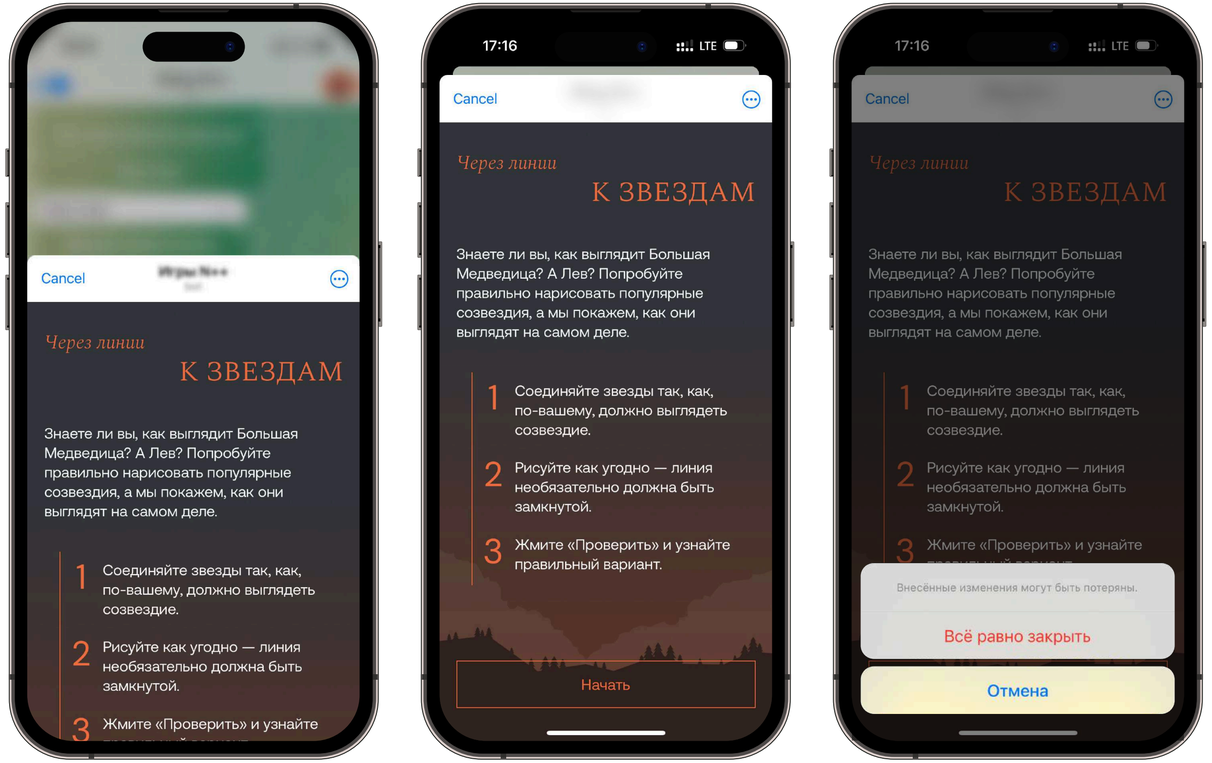
```ts
declare global {
  const Telegram: {
    WebApp: {
      ready(): void;
      // ...
    }
  }
}

Telegram.WebApp.ready();
```

НАСТРОЙКА ПОВЕДЕНИЯ ПРИЛОЖЕНИЯ

Для разворачивания приложения на всю высоту экрана использовать метод Telegram.WebApp.expand

Для запроса подтверждения закрытия приложения использовать метод Telegram.WebApp.enableClosingConfirmation

```
// Включаем подтверждение закрытия
Telegram.WebApp.enableClosingConfirmation();
// Раскрываем на всю высоту
Telegram.WebApp.expand();

Telegram.WebApp.ready();
```

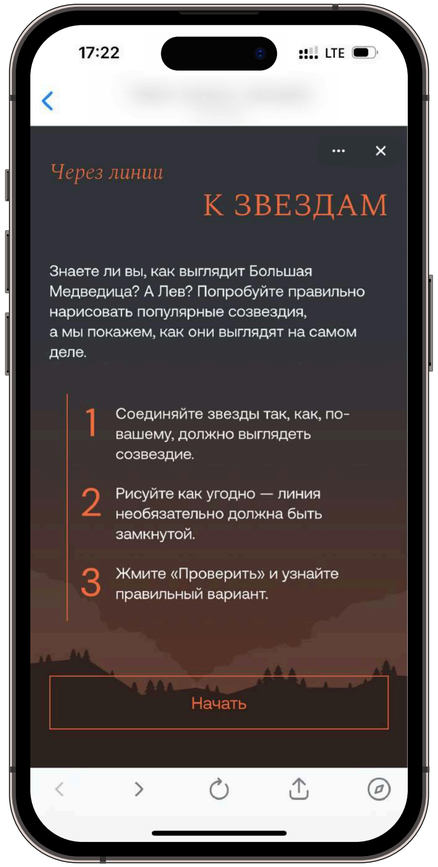


ОСТОРОЖНАЯ РАБОТА СО СВАЙПАМИ

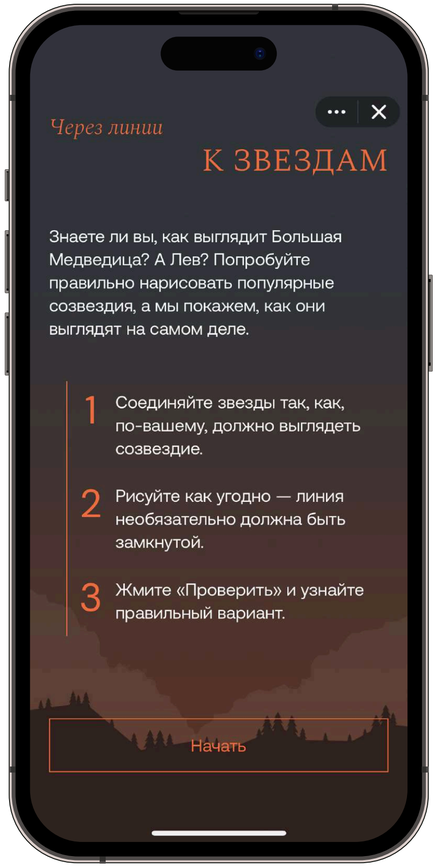
Поскольку приложение открывается в модальном окне, которое закрывается по свайпу, для корректной работы внутри приложения необходимо отлавливать свайпы на их целевом контейнере и не пускать дальше

```
swipeableContainer.addEventListener('touchmove', (event) => {
    event.preventDefault();
    event.stopPropagation();
  },
  {
    capture: true,
    passive: false
  }
});
```

ИТОГ



ВКОНТАКТЕ



ОДНОКЛАССНИКИ

TELEGRAM

IFRAME

ВСТРАИВАНИЕ НА ВСЕ ПЛОЩАДКИ СРАЗУ

GAMES.RU/VK

HTML со скриптом VK Bridge

GAMES.RU/OK

HTML со скриптом FAPI

GAMES.RU/TG

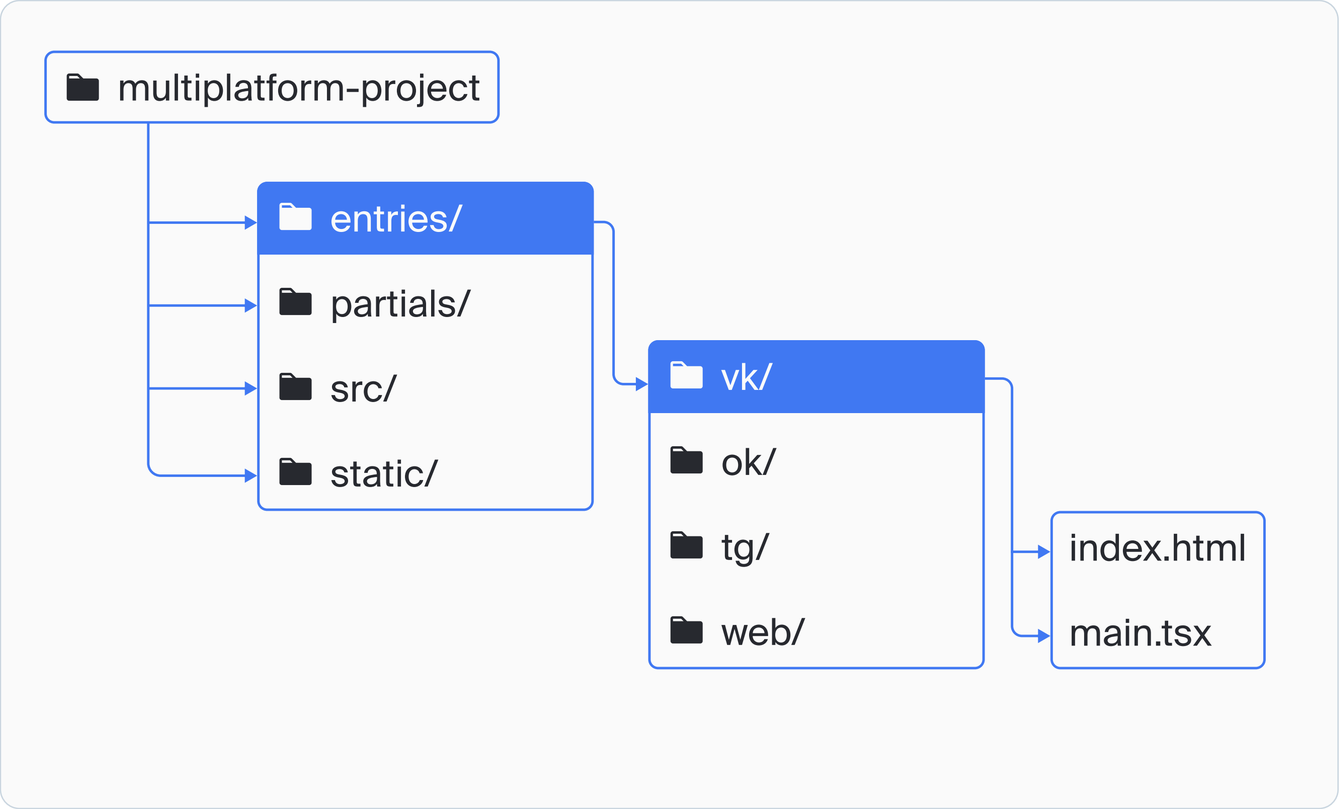
HTML со скриптом Telegram SDK

GAMES.RU

HTML со скриптом iframeResizer

ПОДГОТАВЛИВАЕМ СТРУКТУРУ ПРОЕКТА

Создаем директорию entries, внутри заводим директорию под каждую платформу и внутри кладем ее index.html и main.tsx



ЗАПОЛНЯЕМ HTML-ФАЙЛЫ

В каждый HTML-файл подключаем соответствующие внешний скрипт SDK и свой скрипт

```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <title>Мультиплатформенный проект</title>
    <link rel="icon" type="image/png" sizes="32x32"
href="/meta/favicon.png" />
    <meta property="og:description" content="Проект для VK, OK, TG и
iframe" />
    <meta property="og:image" content="/meta/share.png" />
    <script src="https://unpkg.com/@vkontakte/vk-
bridge/dist/browser.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/entries/vk/main.tsx"></script>
  </body>
</html>
```

ПИШЕМ ЛОГИКУ ИНИЦИАЛИЗАЦИИ

Под каждую платформу в main.tsx прописываем свою логику

```
import { startApp } from '../../src/main';

import { VKBridge } from './types';
import { configureViewSettings } from './utils';

declare global {
  const vkBridge: VKBridge;
}

const startVkApp = () => {
  vkBridge.send('VKWebAppInit', {}).then(() => {
    configureViewSettings(vkBridge);
    startApp('vk');
  });
};

startVkApp();
```

НАСТРАИВАЕМ VITE

Настраиваем несколько входных точек в конфиге vite

```
export default defineConfig(() => {
  return {
    // ...
    build: {
      outDir: 'public',
      // ...
      rollupOptions: {
        input: {
          vk: '/entries/vk/index.html',
          ok: '/entries/ok/index.html',
          tg: '/entries/tg/index.html',
          web: '/entries/web/index.html',
        },
      },
    },
    // ..
  };
});
```

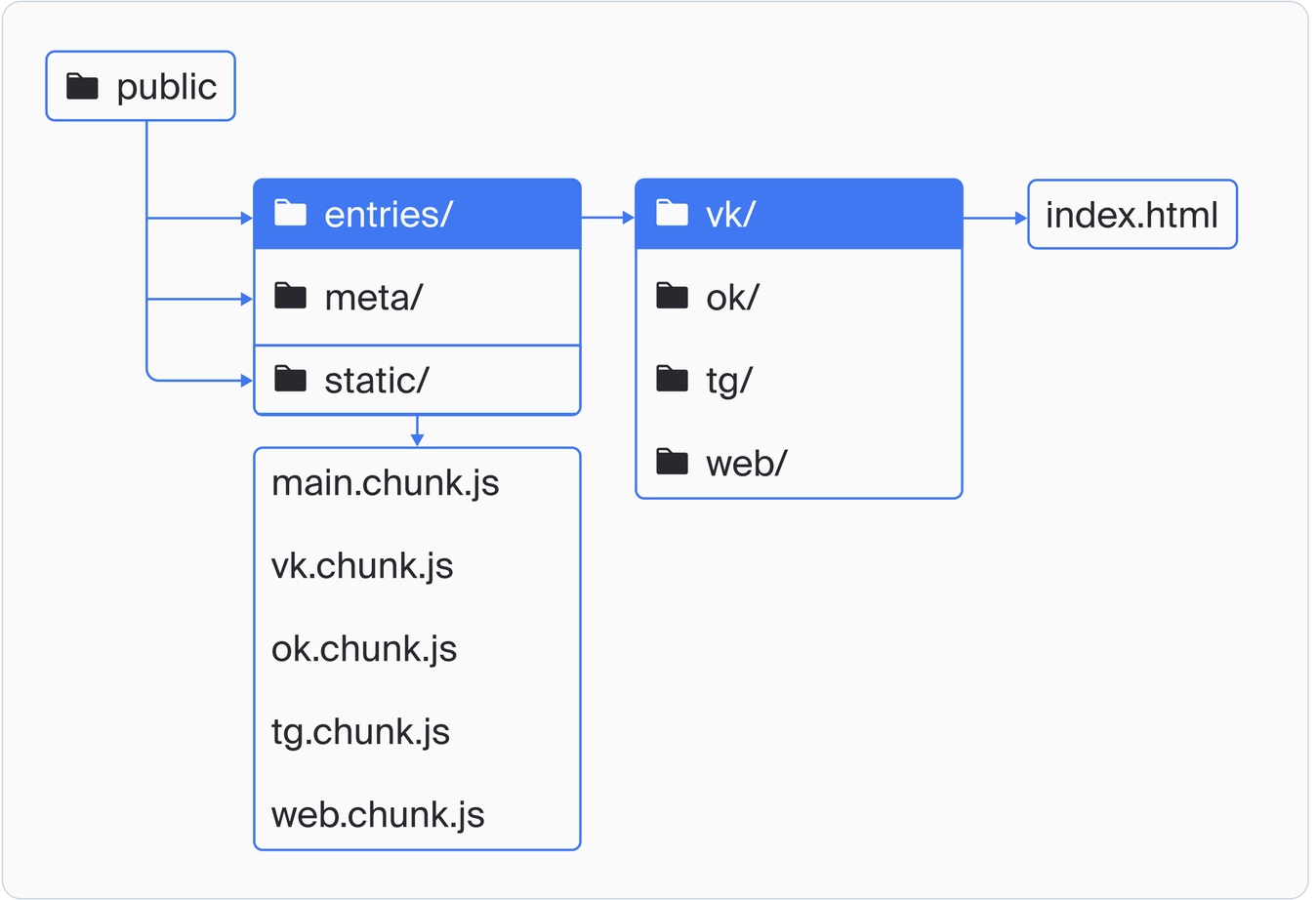
ПОДКЛЮЧАЕМ ШАБЛОНИЗАТОР

Подключаем шаблонизатор к vite через плагин и вставляем переиспользуемые куски в каждый HTML-файл

```
<!DOCTYPE html>
<html lang="ru">
<head>
  {{> head }}
  <script src="https://unpkg.com/@vkontakte/vk-
bridge/dist/browser.min.js"></script>
</head>
<body>
  {{> body }}
  <script type="module" src="/entries/vk/main.tsx"></script>
</body>
</html>
```

СМОТРИМ СБОРКУ

В бандле создаются отдельные HTML-файлы под каждую платформу, а статика остается общая



НАСТРАИВАЕМ NGINX

Под каждую платформу создаем свой location, который отдает нужный HTML-файл

```
server {
    #...

    location / {
        rewrite ^ /entries/web/index.html break;
        # ...
    }

    location /vk {
        rewrite ^/vk(/)?.*$ /entries/vk/index.html break;
        # ...
    }

    # ...

    location /static {
        # ...
    }

    location /meta {
        # ...
    }
}
```

НАСТРАИВАЕМ РОУТИНГ

Для каждой платформы добавляем соответствующий basename

```tsx
type Platform = 'vk' | 'ok' | 'tg' | 'web';

export const PLATFORMS_ROUTER_BASENAMES: Record<Platform, string> = {
  iframe: '/',
  vk: '/vk',
  ok: '/ok',
  tg: '/tg',
};

const startApp = (platform: Platform) => {
  createRoot(document.getElementById('root')!).render(
    <React.StrictMode>
      <Router basename={PLATFORMS_ROUTER_BASENAMES[platform]}>
        <App platform={platform} />
      </Router>
    </React.StrictMode>
  );
};
```

ДОРАБАТЫВАЕМ DEV-SERVER

Пишем плагин для vite, который
перехватывает запросы
и направляет на нужный путь
к платформе

```
const PLATFORM_ROUTES = [
  { platform: 'web', route: '/' },
  { platform: 'ok', route: '/ok' },
  { platform: 'tg', route: '/tg' },
  { platform: 'vk', route: '/vk' },
];

const getPlatformPath = (platform: string) => `/entries/${platform}/index.html`;

const customHistoryApiFallback = () => {
  return {
    name: 'middleware',
    apply: 'serve',
    configureServer(viteDevServer) {
      return () => {
        viteDevServer.middlewares.use(async (req, res, next) => {
          PLATFORM_ROUTES.forEach(({ route, platform }) => {
            if (req.originalUrl.startsWith(route)) {
              req.url = getPlatformPath(platform);
            }
          });

          next();
        });
      };
    },
  };
};
```

ССЫЛКА НА ПРОЕКТ

GITHUB.COM/N-MERKULOVA/MULTIPLATFORM-PROJECT

СПАСИБО, КОМАНДА!

НАТАША

АЛИСА

МАРГО

СЕРЕЖА

ГРАНТ

ПОЛИНА

КРИСТИНА

ЕГОР

СПАСИБО ЗА ВНИМАНИЕ! ВОПРОСЫ?

Почта

n.merkulova@kts.tech

Telegram

t.me/nmerk