










Документаиция "Расписание для Хекслет Колледж"

Документация представляет собой подробное руководство по мобильному приложению для просмотра расписания занятий Хекслет Колледжа.

Заголовки

 Документация проекта "Расписание для Хекслет"	2
 Общая информация	9
 Архитектура приложения	15
 Управление состоянием	24
 Стилизация	32
 Модели данных	40
 Сетевой слой	50
 Оптимизация производительности	59
Руководство разработчика	67
 Компоненты	74
API Документация	80

Документация проекта "Расписание для Хекслет"

Содержание

Общая информация

- Описание проекта
 - Цели и задачи
 - Целевая аудитория
 - Ключевые особенности
- Основные возможности
 - Просмотр расписания
 - Работа с избранным
 - Поиск и фильтрация
 - Настройка отображения
- Архитектура приложения
 - Общая структура
 - Компонентный подход
 - Управление состоянием
 - Работа с данными

Архитектура

- Общая структура

- Навигация
- Управление состоянием
- Потоки данных
- Компонентная архитектура
- Хранение данных
- Оптимизация
- Безопасность
- Масштабируемость

Модели данных

- Основные типы
 - Группа
 - Преподаватель
 - Расписание
 - Занятия
- Избранное
- Настройки
 - Общие настройки
 - Настройки темы
 - Настройки расписания
 - Настройки уведомлений
- Состояния
 - Состояния загрузки
 - Состояние расписания

- Фильтры
- Ответы API
- Преобразования данных
- Валидация
- Хранение
- Типы событий

Сетевой слой

- Конфигурация API
- API Сервисы
- Обработка ошибок
- Кэширование
- Retry механизм
- Мониторинг
- Оптимизация
- Безопасность

Производительность

- Метрики производительности
- Оптимизация рендеринга
 - Мемоизация
 - Виртуализация
 - Ленивая загрузка
- Оптимизация изображений

- Оптимизация состояния
- Оптимизация сети
- Профилирование
- Оптимизация памяти

Управление состоянием




- Контексты
 - FavoritesContext
 - ThemeContext
 - ScheduleSettingsContext
- Локальное состояние
- Персистентность данных
- Оптимизация
- Обработка ошибок

Руководство разработчика




- Настройка окружения
- Установка и запуск
- Структура проекта
- Разработка
- Тестирование
- Сборка и деплой
- Рекомендации
- Решение проблем

Быстрые ссылки





Начало работы

- Установка проекта ([Руководство разработчика](#))
- Структура проекта ( [Общая информация](#))
- Модели данных ( [Модели данных](#))
- Работа с API ( [Сетевой слой](#))

Разработка

- Создание компонентов ([Руководство разработчика](#))
- Управление состоянием ( [Управление состоянием](#))
- Оптимизация ( [Оптимизация производительности](#))
- Работа с данными ( [Модели данных](#))

Архитектура

- Общая структура ( [Архитектура приложения](#))
- Навигация ( [Архитектура приложения](#))
- Поток данных ( [Архитектура приложения](#))
- Безопасность ( [Архитектура приложения](#))

Обновления и поддержка

Версионирование

- Semantic Versioning
- Changelog

- Breaking Changes
- Миграции



Поддержка

- Обновления
- Исправление ошибок
- Улучшение производительности
- Обновление документации




? FAQ

Разработка




1. Как добавить новый компонент?

- Создание компонентов ([Руководство разработчика](#))
- Типизация ( [Модели данных](#))
- Оптимизация ( [Оптимизация производительности](#))




2. Как работать с данными?

- Модели данных ( [Модели данных](#))
- API сервисы ( [Сетевой слой](#))
- Кэширование ( [Сетевой слой](#))

3. Как оптимизировать производительность?

- Оптимизация рендеринга ( [Оптимизация производительности](#))
- Оптимизация состояния ( [Управление состоянием](#))
- Оптимизация сети ( [Сетевой слой](#))

Архитектура

- Структура проекта ( [Архитектура приложения](#))
- Управление состоянием ( [Управление состоянием](#))
- Работа с данными ( [Модели данных](#))

Общая информация

Описание проекта

Мобильное приложение "Расписание для Хекслет" предназначено для просмотра расписания занятий студентами и преподавателями Хекслет Колледжа.

Приложение разработано с использованием современных технологий React Native и Expo.

Основные возможности

- Просмотр расписания групп и преподавателей
- Переключение между текущей и следующей неделей
- Добавление в избранное
- Умный поиск
- Темная и светлая тема
- Настройка отображения информации

Архитектура

Структура проекта

```
app/  
├─ _layout.tsx          # Корневой layout  
├─ (tabs)/              # Основные экраны  
│   ├─ _layout.tsx      # Навигация по вкладкам  
│   ├─ index.tsx        # Список групп  
│   ├─ teacher.tsx      # Список преподавателей  
│   ├─ favorites.tsx     # Избранное  
│   └─ settings.tsx     # Настройки  
├─ schedule/            # Экраны расписания  
│   └─ [id].tsx         # Расписание группы  
└─ teacher/             # Экраны преподавателей
```

```
└─ [id].tsx          # Расписание преподавателя

components/          # Переиспользуемые компоненты
├─ GroupList.tsx     # Список групп
├─ TeacherList.tsx   # Список преподавателей
├─ ScheduleView.tsx  # Отображение расписания
├─ LessonCard.tsx    # Карточка занятия
└─ WeekSelector.tsx  # Выбор недели

contexts/             # React контексты
├─ FavoritesContext.tsx # Управление избранным
├─ ThemeContext.tsx    # Управление темой
└─ ScheduleSettingsContext.tsx # Настройки расписания

constants/           # Константы приложения
├─ groups.ts          # Информация о группах
├─ teachers.ts        # Информация о преподавателях
└─ theme.ts           # Настройки тем
```

Жизненный цикл приложения

1. Инициализация

- Загрузка приложения
- Инициализация контекстов
- Загрузка сохраненных настроек
- Проверка темы устройства

2. Основной цикл

- Рендеринг интерфейса
- Обработка пользовательских действий
- Управление состоянием

- Кэширование данных

3. Обновление данных

- Загрузка расписания
- Обновление избранного
- Сохранение настроек
- Обработка ошибок

Управление состоянием

FavoritesContext

Управление избранными группами и преподавателями.

```
interface FavoritesContextType {
  favorites: FavoriteItem[];
  addToFavorites: (item: GroupInfo | TeacherInfo, type: 'group' | 'teacher') => Promise<void>;
  removeFromFavorites: (id: number, type: 'group' | 'teacher') => Promise<void>;
  isFavorite: (id: number, type: 'group' | 'teacher') => boolean;
}
```

ThemeContext

Управление темой оформления.

```
interface ThemeContextType {
  isDarkTheme: boolean;
  useSystemTheme: boolean;
  toggleTheme: () => Promise<void>;
  toggleSystemTheme: () => Promise<void>;
}
```

ScheduleSettingsContext

Настройки отображения расписания.

```
interface ScheduleSettings {  
  showCabinetNumbers: boolean;  
  showTeacherNames: boolean;  
  compactMode: boolean;  
  showLessonNumbers: boolean;  
}
```

Навигация

Структура навигации

- Вкладки (Tabs)
 - Расписание (index)
 - Преподаватели (teacher)
 - Избранное (favorites)
 - Настройки (settings)
- Стек навигации
 - Расписание группы (schedule/[id])
 - Расписание преподавателя (teacher/[id])
 - Информация (info)

Параметры навигации

```
type RootStackParamList = {  
  '(tabs)': undefined;  
  'info': undefined;  
  'schedule/[id]': { id: string };  
  'teacher/[id]': { id: string };  
};
```

Обработка данных

Локальное хранение

- AsyncStorage для настроек и избранного
- Кэширование расписания
- Управление состоянием приложения

Сетевые запросы

- Fetch API для получения данных
- Обработка ошибок сети
- Retry механизмы
- Кэширование ответов

Стилизация

Темы

- Светлая тема
- Темная тема
- Системная тема

Компоненты

- Material Design
- Адаптивный дизайн
- Анимации
- Кастомные компоненты

Поддерживаемые платформы

iOS

- iOS 13.0 и выше
- Оптимизация для iPhone
- Поддержка жестов iOS

Android

- Android 6.0 и выше
- Material Design
- Адаптация под разные устройства



Архитектура приложения

Общая структура

Файловая структура

```
app/
├─ _layout.tsx           # Корневой layout с провайдерами
├─ (tabs)/               # Основные экраны в табх
│   ├─ _layout.tsx      # Настройка навигации по табам
│   ├─ index.tsx        # Список групп
│   ├─ teacher.tsx      # Список преподавателей
│   ├─ favorites.tsx     # Избранное
│   └─ settings.tsx     # Настройки
├─ schedule/            # Экраны расписания
│   └─ [id].tsx         # Динамический роут расписания группы
└─ teacher/             # Экраны преподавателей
    └─ [id].tsx         # Динамический роут расписания
                          преподавателя

components/              # Переиспользуемые компоненты
├─ GroupList.tsx        # Список групп
├─ TeacherList.tsx      # Список преподавателей
├─ ScheduleView.tsx     # Отображение расписания
├─ LessonCard.tsx       # Карточка занятия
└─ WeekSelector.tsx     # Выбор недели

contexts/                # React контексты
├─ FavoritesContext.tsx # Управление избранным
├─ ThemeContext.tsx     # Управление темой
└─ ScheduleSettingsContext.tsx # Настройки расписания
```

Навигация

Структура роутинга


```
type RootStackParamList = {
  '(tabs)': undefined;
  'info': undefined;
  'schedule/[id]': { id: string };
  'teacher/[id]': { id: string };
};
```

Схема навигации

```
Root
├─ (tabs)
│   ├─ index (Группы)
│   ├─ teacher (Преподаватели)
│   ├─ favorites (Избранное)
│   └─ settings (Настройки)
├─ schedule/[id] (Расписание группы)
├─ teacher/[id] (Расписание преподавателя)
└─ info (Информация)
```

Управление состоянием

Контексты

FavoritesContext

Управление избранными элементами:

```
interface FavoritesContextType {
  favorites: FavoriteItem[];
  addToFavorites: (item: GroupInfo | TeacherInfo, type: 'group' | 'teacher') => Promise<void>;
  removeFromFavorites: (id: number, type: 'group' | 'teacher') => Promise<void>;
  isFavorite: (id: number, type: 'group' | 'teacher') => boolean;
}
```

ThemeContext

Управление темой оформления:

```
interface ThemeContextType {  
    isDarkTheme: boolean;  
    useSystemTheme: boolean;  
    toggleTheme: () => Promise<void>;  
    toggleSystemTheme: () => Promise<void>;  
}
```

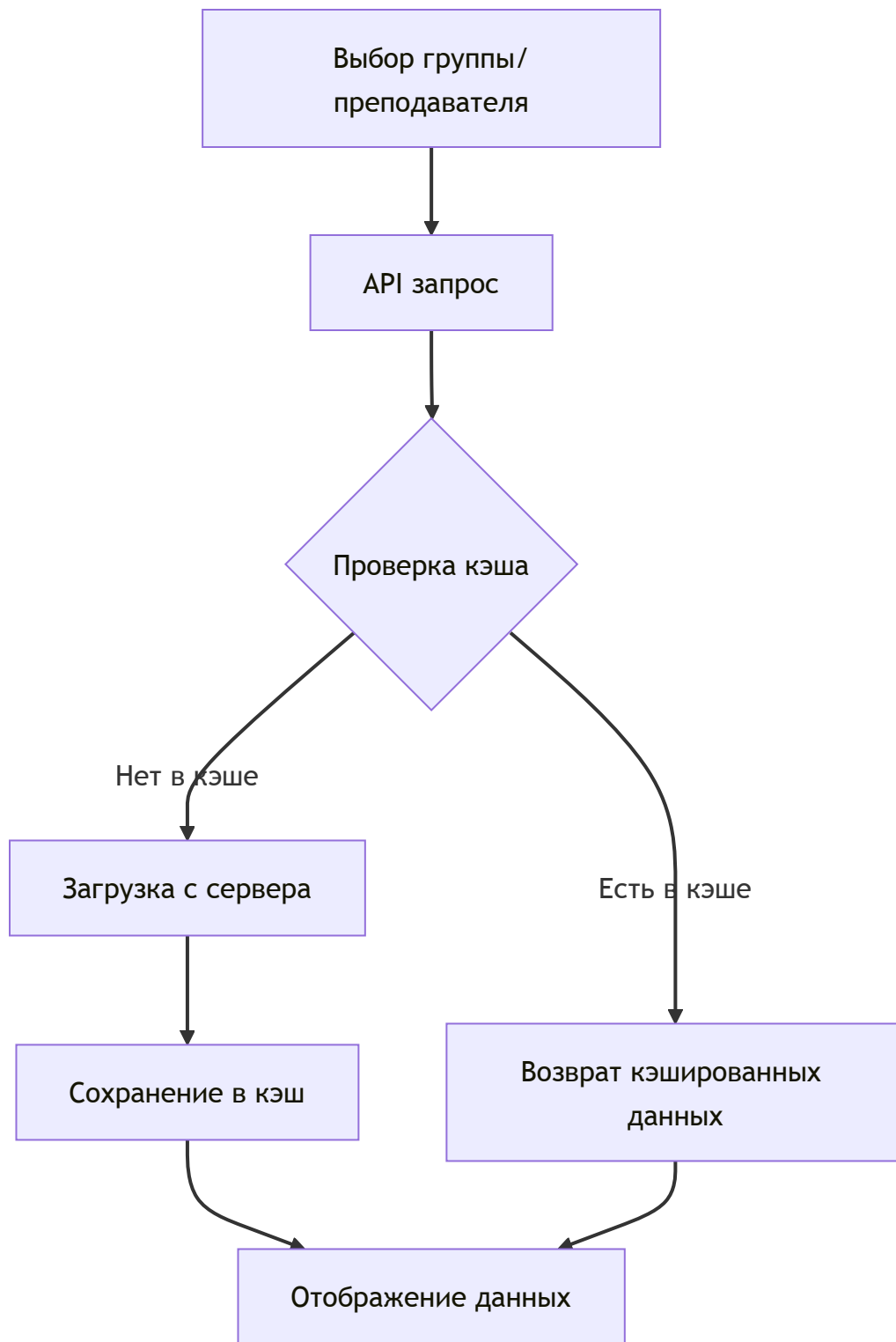
ScheduleSettingsContext

Настройки отображения расписания:

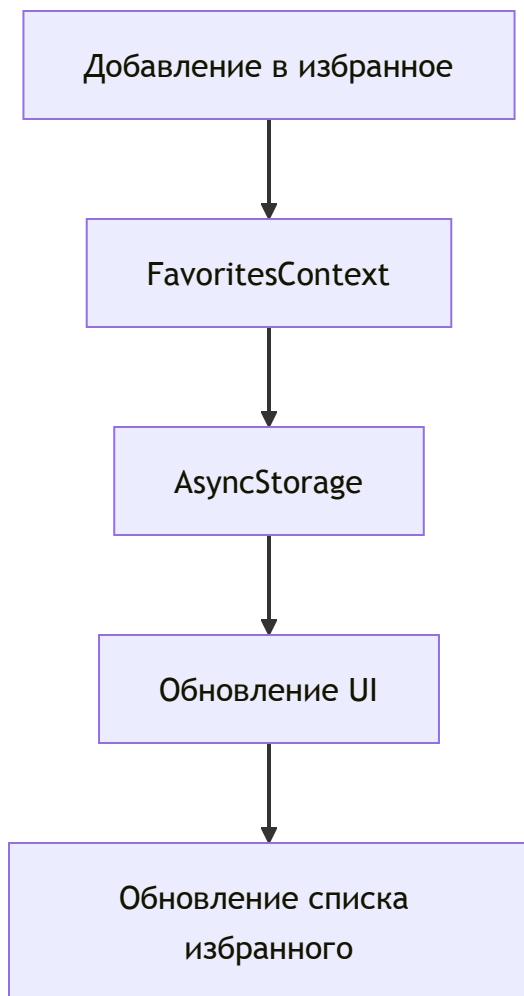
```
interface ScheduleSettings {  
    showCabinetNumbers: boolean;  
    showTeacherNames: boolean;  
    compactMode: boolean;  
    showLessonNumbers: boolean;  
}
```

Потоки данных

Загрузка расписания



Управление избранным



Компонентная архитектура

Основные компоненты

ScheduleView

- Отображение расписания
- Обработка состояний загрузки
- Группировка по дням
- Интеграция с WeekSelector

GroupList/TeacherList

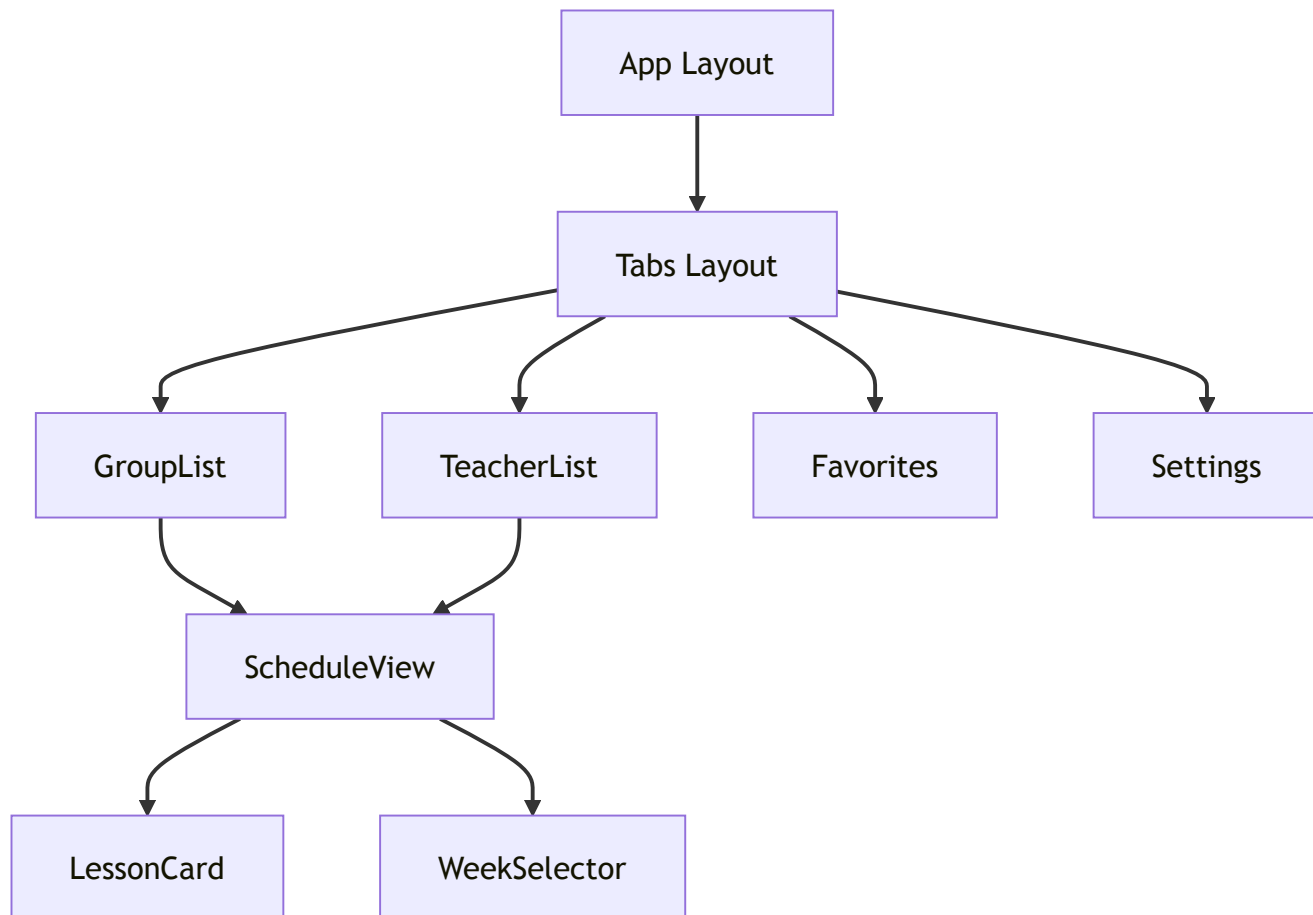
- Отображение списков

- Поиск и фильтрация
- Оптимизация производительности
- Интеграция с FavoritesContext

LessonCard

- Отображение информации о занятии
- Адаптивный дизайн
- Интеграция с ScheduleSettingsContext

Взаимодействие компонентов



Хранение данных

AsyncStorage

- Избранное
- Настройки темы
- Настройки отображения
- Кэш расписания

Кэширование

```
interface CacheItem<T> {  
  data: T;  
  timestamp: number;  
  expiry: number;  
}
```

Оптимизация

Стратегии

1. Мемоизация компонентов
2. Виртуализация списков
3. Кэширование данных
4. Ленивая загрузка изображений

Производительность

```
// Пример оптимизации списка  
const MemoizedItem = React.memo(({ item }) => (  
  <View>  
    <Text>{item.name}</Text>  
  </View>  
));  
  
const VirtualizedList = () => (  
  <FlatList  
    data={items}
```

```
renderItem={({ item }) => <MemoizedItem item={item} />}  
getItemLayout={(data, index) => ({  
  length: ITEM_HEIGHT,  
  offset: ITEM_HEIGHT * index,  
  index,  
})}  
/>  
);
```

Безопасность

Защита данных

- Валидация входных данных
- Санитизация данных
- Безопасное хранение

Обработка ошибок

```
try {  
  const data = await api.fetchSchedule();  
} catch (error) {  
  if (error instanceof NetworkError) {  
    // Обработка сетевых ошибок  
  } else if (error instanceof ValidationError) {  
    // Обработка ошибок валидации  
  }  
}
```

Масштабируемость

Добавление новых функций

1. Создание нового компонента
2. Интеграция с существующими контекстами

3. Добавление новых роутов

4. Обновление типов

Модификация существующего функционала

1. Изменение интерфейсов

2. Обновление компонентов

3. Миграция данных

4. Обновление документации



Управление состоянием

Обзор

В приложении используется комбинация React Context API и локального состояния компонентов для эффективного управления данными.

Контексты

FavoritesContext

Управление избранными группами и преподавателями.

```
// contexts/FavoritesContext.tsx

interface FavoritesContextType {
  favorites: FavoriteItem[];
  addToFavorites: (item: GroupInfo | TeacherInfo, type: 'group' | 'teacher') => Promise<void>;
  removeFromFavorites: (id: number, type: 'group' | 'teacher') => Promise<void>;
  isFavorite: (id: number, type: 'group' | 'teacher') => boolean;
}

const FavoritesContext = createContext<FavoritesContextType | undefined>(undefined);

export const FavoritesProvider: React.FC = ({ children }) => {
  const [favorites, setFavorites] = useState<FavoriteItem[]>([]);

  // Загрузка избранного при монтировании
  useEffect(() => {
    loadFavorites();
  }, []);

  // Методы управления избранным
  const addToFavorites = async (item, type) => {
```

```

    // Реализация
  };

  const removeFromFavorites = async (id, type) => {
    // Реализация
  };

  return (
    <FavoritesContext.Provider value={{
      favorites,
      addToFavorites,
      removeFromFavorites,
      isFavorite
    }}>
      {children}
    </FavoritesContext.Provider>
  );
};

```

ThemeContext

Управление темой оформления приложения.

```

// contexts/ThemeContext.tsx

interface ThemeContextType {
  isDarkTheme: boolean;
  useSystemTheme: boolean;
  toggleTheme: () => Promise<void>;
  toggleSystemTheme: () => Promise<void>;
}

export const ThemeProvider: React.FC = ({ children }) => {
  const [isDarkTheme, setIsDarkTheme] = useState(false);
  const [useSystemTheme, setUseSystemTheme] = useState(true);

  // Инициализация темы
  useEffect(() => {

```

```

    loadThemeSettings();
  }, []);

  // Методы управления темой
  const toggleTheme = async () => {
    // Реализация
  };

  const toggleSystemTheme = async () => {
    // Реализация
  };

  return (
    <ThemeContext.Provider value={{
      isDarkTheme,
      useSystemTheme,
      toggleTheme,
      toggleSystemTheme
    }}>
      {children}
    </ThemeContext.Provider>
  );
};

```

ScheduleSettingsContext

Настройки отображения расписания.

```

// contexts/ScheduleSettingsContext.tsx

interface ScheduleSettings {
  showCabinetNumbers: boolean;
  showTeacherNames: boolean;
  compactMode: boolean;
  showLessonNumbers: boolean;
}

export const ScheduleSettingsProvider: React.FC = ({ children })

```

```

=> {
  const [settings, setSettings] = useState<ScheduleSettings>
(defaultSettings);

  // Загрузка настроек
  useEffect(() => {
    loadSettings();
  }, []);

  // Методы управления настройками
  const updateSettings = async (newSettings:
Partial<ScheduleSettings>) => {
    // Реализация
  };

  return (
    <ScheduleSettingsContext.Provider value={{
      settings,
      updateSettings
    }}>
      {children}
    </ScheduleSettingsContext.Provider>
  );
};

```

Иерархия состояний

```

App (_layout.tsx)
├─ ThemeProvider
│   └─ FavoritesProvider
│       └─ ScheduleSettingsProvider
│           └─ RootLayoutNav
│               └─ TabsNavigator
│                   └─ GroupList (использует FavoritesContext)
│                   └─ TeacherList (использует FavoritesContext)
│                   └─ Settings (использует все контексты)

```

| | — ScheduleView (использует
ScheduleSettingsContext)

Локальное состояние

Компоненты с локальным состоянием

GroupList

```
const GroupList: React.FC<GroupListProps> = () => {  
  const [searchQuery, setSearchQuery] = useState('');  
  const [filteredGroups, setFilteredGroups] =  
    useState<GroupInfo[]>([]);  
  
  // Логика фильтрации  
};
```

WeekSelector

```
const WeekSelector: React.FC<WeekSelectorProps> = () => {  
  const [isLoading, setIsLoading] = useState(false);  
  
  // Логика переключения недель  
};
```

ScheduleView

```
const ScheduleView: React.FC<ScheduleViewProps> = () => {  
  const [refreshing, setRefreshing] = useState(false);  
  const [error, setError] = useState<string | null>(null);  
  
  // Логика обновления данных  
};
```

Персистентность данных

AsyncStorage

Сохранение состояния в локальное хранилище:

```
// Ключи для хранения
const STORAGE_KEYS = {
  FAVORITES: '@favorites',
  THEME: '@theme',
  SETTINGS: '@settings'
};

// Сохранение данных
const saveData = async (key: string, data: any) => {
  try {
    await AsyncStorage.setItem(key, JSON.stringify(data));
  } catch (error) {
    console.error('Error saving data:', error);
  }
};

// Загрузка данных
const loadData = async (key: string) => {
  try {
    const data = await AsyncStorage.getItem(key);
    return data ? JSON.parse(data) : null;
  } catch (error) {
    console.error('Error loading data:', error);
    return null;
  }
};
```

Оптимизация состояния

Мемоизация

```
// Мемоизация вычисляемых значений
const filteredGroups = useMemo(() => {
  return groups.filter(group =>
    group.name.toLowerCase().includes(searchQuery.toLowerCase())
```

```

    );
  }, [groups, searchQuery]));

  // Мемоизация колбэков
  const handleGroupSelect = useCallback((group: GroupInfo) => {
    // Обработка выбора группы
  }, []);

```

Дебаунсинг

```

// Дебаунс поискового запроса
const debouncedSearch = useCallback(
  debounce((query: string) => {
    setFilteredGroups(
      groups.filter(group =>
        group.name.toLowerCase().includes(query.toLowerCase())
      )
    );
  }, 300),
  [groups]
);

```

Обработка ошибок

```

const ErrorBoundary: React.FC = ({ children }) => {
  const [error, setError] = useState<Error | null>(null);

  if (error) {
    return (
      <View style={styles.errorContainer}>
        <Text style={styles.errorText}>
          Произошла ошибка: {error.message}
        </Text>
      </View>
    );
  }
}

```

```
    return children;  
};
```

Рекомендации

1. Использовать контексты для глобального состояния
2. Держать локальное состояние максимально близко к компонентам
3. Применять мемоизацию для оптимизации
4. Использовать TypeScript для типизации состояния
5. Обращивать ошибки на всех уровнях



Стилизация

Темы

Светлая тема

```
const lightTheme = {
  colors: {
    primary: '#7f61dd',
    background: '#ffffff',
    surface: '#ffffff',
    text: '#000000',
    onSurface: '#000000',
    onSurfaceVariant: '#666666',
    outline: '#e0e0e0',
    primaryContainer: '#f4f0ff',
    secondary: '#9e9e9e',
    error: '#b00020',
    errorContainer: '#fde7e9'
  }
};
```

Темная тема

```
const darkTheme = {
  colors: {
    primary: '#9f81fd',
    background: '#121212',
    surface: '#1e1e1e',
    text: '#ffffff',
    onSurface: '#ffffff',
    onSurfaceVariant: '#cccccc',
    outline: '#2c2c2c',
    primaryContainer: '#2c2c2c',
    secondary: '#757575',
    error: '#cf6679',
  }
};
```

```
    errorContainer: '#340e0e'
  }
};
```

Компоненты UI

Карточки

```
const cardStyles = StyleSheet.create({
  container: {
    borderRadius: 16,
    borderWidth: 1,
    overflow: 'hidden',
    ...Platform.select({
      ios: {
        shadowColor: '#000',
        shadowOffset: { width: 0, height: 2 },
        shadowOpacity: 0.1,
        shadowRadius: 4,
      },
      android: {
        elevation: 4,
      },
    }),
  },
  content: {
    padding: 16,
  },
  header: {
    flexDirection: 'row',
    alignItems: 'center',
    marginBottom: 12,
  },
  title: {
    fontSize: 16,
    fontWeight: '600',
    letterSpacing: -0.3,
```

```
}  
});
```

Кнопки

```
const buttonStyles = StyleSheet.create({  
  primary: {  
    borderRadius: 12,  
    paddingVertical: 12,  
    paddingHorizontal: 16,  
    backgroundColor: theme.colors.primary,  
  },  
  secondary: {  
    borderRadius: 12,  
    paddingVertical: 12,  
    paddingHorizontal: 16,  
    backgroundColor: theme.colors.primaryContainer,  
  },  
  text: {  
    fontSize: 16,  
    fontWeight: '600',  
    textAlign: 'center',  
  }  
});
```

Поля ввода

```
const inputStyles = StyleSheet.create({  
  container: {  
    borderRadius: 12,  
    borderWidth: 1,  
    paddingHorizontal: 16,  
    height: 44,  
    flexDirection: 'row',  
    alignItems: 'center',  
  },  
  input: {
```

```
    flex: 1,  
    fontSize: 16,  
    fontWeight: '500',  
  },  
  icon: {  
    marginRight: 12,  
  }  
});
```

Типографика

Заголовки

```
const typography = StyleSheet.create({  
  h1: {  
    fontSize: 24,  
    fontWeight: '700',  
    letterSpacing: -0.5,  
  },  
  h2: {  
    fontSize: 20,  
    fontWeight: '600',  
    letterSpacing: -0.3,  
  },  
  h3: {  
    fontSize: 18,  
    fontWeight: '600',  
  },  
  body: {  
    fontSize: 16,  
    lineHeight: 24,  
  },  
  caption: {  
    fontSize: 14,  
    lineHeight: 20,  
  }  
});
```

Анимации

Переходы

```
const fadeIn = {
  from: {
    opacity: 0,
  },
  to: {
    opacity: 1,
  },
};

const slideIn = {
  from: {
    transform: [{ translateY: 20 }],
    opacity: 0,
  },
  to: {
    transform: [{ translateY: 0 }],
    opacity: 1,
  },
};
```

Интерактивные элементы

```
const pressableStyles = (pressed: boolean) => ({
  transform: [{ scale: pressed ? 0.98 : 1 }],
  opacity: pressed ? 0.9 : 1,
});
```

Адаптивный дизайн

Размеры экрана

```
const { width, height } = Dimensions.get('window');
```

```
const metrics = {
  screenWidth: width,
  screenHeight: height,
  isSmallDevice: width < 375,
};
```

Отступы

```
const spacing = {
  xs: 4,
  sm: 8,
  md: 16,
  lg: 24,
  xl: 32,
};

const layout = StyleSheet.create({
  container: {
    padding: spacing.md,
  },
  row: {
    flexDirection: 'row',
    gap: spacing.sm,
  },
  column: {
    flexDirection: 'column',
    gap: spacing.md,
  }
});
```

Использование тем

Хук `useTheme`

```
const Component = () => {
  const theme = useTheme();

  return (
```

```

<View style={
  styles.container,
  { backgroundColor: theme.colors.background }
}>
  <Text style={
    styles.text,
    { color: theme.colors.onSurface }
  }>
    Контент
  </Text>
</View>
);
};

```

Переключение тем

```

const { isDarkTheme, toggleTheme } = useThemeContext();

// Применение стилей в зависимости от темы
const dynamicStyles = {
  backgroundColor: isDarkTheme ? '#121212' : '#ffffff',
  color: isDarkTheme ? '#ffffff' : '#000000',
};

```

Рекомендации по стилизации

Лучшие практики

1. Использовать `StyleSheet.create` для оптимизации
2. Группировать стили по компонентам
3. Использовать константы для повторяющихся значений
4. Применять `Platform.select` для платформо-зависимых стилей

Производительность

1. Избегать инлайн стилей
2. Мемоизировать динамические стили
3. Использовать `StyleSheet.flatten` при необходимости
4. Оптимизировать анимации

Доступность

1. Использовать достаточный контраст
2. Обеспечивать читаемые размеры текста
3. Поддерживать масштабирование
4. Учитывать VoiceOver и TalkBack



Модели данных

Основные типы

Группа

```
// Базовая информация о группе
interface GroupInfo {
  id: number;           // Уникальный идентификатор группы
  name: string;         // Название группы (например, "П-1")
  category: string;     // Категория группы (например,
  "Программирование")
  course?: number;      // Номер курса (опционально)
  studentsCount?: number; // Количество студентов (опционально)
}

// Расписание группы
interface GroupSchedule {
  startDate: string;    // Дата начала недели в формате ISO
  group: GroupInfo;     // Информация о группе
  lessons: Lesson[];    // Массив занятий
  weekType?: 'current' | 'next'; // Тип недели
  lastUpdated: string; // Время последнего обновления
}

// Группировка групп
interface GroupCategory {
  id: string;           // Идентификатор категории
  name: string;         // Название категории
  groups: GroupInfo[];  // Группы в категории
  description?: string; // Описание категории
}
```

Преподаватель

```
// Базовая информация о преподавателе
interface TeacherInfo {
    id: number;           // Уникальный идентификатор
    fio: string;          // ФИО преподавателя
    position: string;     // Должность
    department?: string;  // Кафедра
    email?: string;       // Email (опционально)
    photoUrl?: string;    // URL фотографии (опционально)
    subjects?: Subject[]; // Преподаваемые предметы
}

// Расписание преподавателя
interface TeacherSchedule {
    startDate: string;    // Дата начала недели
    teacher: TeacherInfo; // Информация о преподавателе
    lessons: Lesson[];    // Массив занятий
    weekType?: 'current' | 'next';
    lastUpdated: string;
    totalHours?: number;  // Общее количество часов
}
```

Расписание

```
// Занятие
interface Lesson {
    id: string;           // Уникальный идентификатор
    weekday: number;      // День недели (1-7)
    lesson: number;       // Номер пары
    startTime: string;    // Время начала (HH:mm)
    endTime: string;      // Время окончания (HH:mm)
    teachers: Teacher[];  // Преподаватели
    subject: Subject;     // Предмет
    cabinet: Cabinet;     // Аудитория
    unionGroups: UnionGroup[]; // Объединенные группы
    type?: LessonType;    // Тип занятия
    isOnline?: boolean;   // Онлайн занятие
    meetingUrl?: string;  // Ссылка на онлайн встречу
}
```

```

    materials?: Material[]; // Учебные материалы
}

// Предмет
interface Subject {
    id: number;           // Идентификатор
    name: string;         // Полное название
    shortName?: string;   // Сокращенное название
    description?: string; // Описание
    color?: string;       // Цвет для отображения
    department?: string;  // Кафедра
}

// Аудитория
interface Cabinet {
    id: number;           // Идентификатор
    number: string;       // Номер аудитории
    building?: string;    // Здание
    floor?: number;       // Этаж
    capacity?: number;    // Вместимость
    type?: CabinetType;   // Тип аудитории
    equipment?: string[]; // Оборудование
}

// Объединенная группа
interface UnionGroup {
    id: number;           // Идентификатор
    name: string;         // Название
    groups: GroupInfo[];  // Входящие группы
    type?: string;        // Тип объединения
}

// Тип занятия
enum LessonType {
    LECTURE = 'lecture',
    PRACTICE = 'practice',
    LAB = 'laboratory',
    SEMINAR = 'seminar',
}

```

```

    CONSULTATION = 'consultation'
}

// Тип аудитории
enum CabinetType {
    CLASSROOM = 'classroom',
    LABORATORY = 'laboratory',
    LECTURE_HALL = 'lecture_hall',
    COMPUTER_CLASS = 'computer_class'
}

// Учебные материалы
interface Material {
    id: string;
    title: string;
    type: 'document' | 'presentation' | 'video' | 'link';
    url: string;
    uploadedAt: string;
}

```

Избранное

Модели избранного

```

// Элемент избранного
interface FavoriteItem {
    id: number;           // Уникальный идентификатор
    type: 'group' | 'teacher'; // Тип элемента
    data: GroupInfo | TeacherInfo; // Данные элемента
    addedAt: string;      // Дата добавления
    order?: number;       // Порядок отображения
    notes?: string;       // Заметки пользователя
    customColor?: string; // Пользовательский цвет
}

// Состояние избранного
interface FavoritesState {

```

```

items: FavoriteItem[]; // Элементы
lastUpdated: string;    // Последнее обновление
maxItems?: number;      // Максимальное количество
categories?: {           // Категории избранного
  groups: FavoriteItem[];
  teachers: FavoriteItem[];
};
}

```

Настройки

Настройки приложения

```

// Общие настройки
interface AppSettings {
  theme: ThemeSettings;
  schedule: ScheduleSettings;
  notifications: NotificationSettings;
  display: DisplaySettings;
  sync: SyncSettings;
}

// Настройки темы
interface ThemeSettings {
  isDarkTheme: boolean;
  useSystemTheme: boolean;
  primaryColor?: string;
  accentColor?: string;
  fontSize?: 'small' | 'medium' | 'large';
}

// Настройки расписания
interface ScheduleSettings {
  showCabinetNumbers: boolean;
  showTeacherNames: boolean;
  compactMode: boolean;
  showLessonNumbers: boolean;
}

```

```

    defaultView: 'day' | 'week';
    showWeekends: boolean;
    highlightCurrentDay: boolean;
    autoScrollToCurrent: boolean;
}

// Настройки уведомлений
interface NotificationSettings {
    enabled: boolean;
    beforeLesson: number; // минуты
    soundEnabled: boolean;
    vibrationEnabled: boolean;
    scheduleUpdates: boolean;
}

// Настройки отображения
interface DisplaySettings {
    language: string;
    timeFormat: '12h' | '24h';
    dateFormat: string;
    listViewDensity: 'compact' | 'normal' | 'comfortable';
}

// Настройки синхронизации
interface SyncSettings {
    autoSync: boolean;
    syncInterval: number;
    syncOnWifiOnly: boolean;
    lastSyncTime: string;
}

```

Состояния

Состояния загрузки

```

// Базовое состояние загрузки
interface LoadingState {

```

```

loading: boolean;    // Флаг загрузки
error: string | null; // Ошибка
lastUpdated?: string; // Последнее обновление
retryCount?: number; // Количество попыток
status?: 'idle' | 'loading' | 'success' | 'error';
}

// Состояние расписания
interface ScheduleState extends LoadingState {
  data: GroupSchedule | TeacherSchedule | null;
  isNextWeek: boolean;
  cache?: {
    [key: string]: CacheItem<GroupSchedule | TeacherSchedule>;
  };
  filters?: ScheduleFilters;
}

// Фильтры расписания
interface ScheduleFilters {
  days?: number[];
  teachers?: number[];
  subjects?: number[];
  types?: LessonType[];
}

```

[Продолжение в следующем сообщении из-за ограничения длины]

Ответы API

Форматы ответов

```

interface ApiResponse<T> {
  success: boolean;
  data: T;
  error?: ApiError;
}

interface ApiError {

```

```
code: string;
message: string;
details?: Record<string, any>;
}
```

Преобразования данных

Маппинг данных

```
// Преобразование сырых данных в модели
const mapGroupSchedule = (raw: any): GroupSchedule => ({
  startDate: raw.start_date,
  group: {
    id: raw.group.id,
    name: raw.group.name,
    category: raw.group.category
  },
  lessons: raw.lessons.map(mapLesson)
});

const mapLesson = (raw: any): Lesson => ({
  id: raw.id,
  weekday: raw.weekday,
  lesson: raw.lesson,
  startTime: raw.start_time,
  endTime: raw.end_time,
  teachers: raw.teachers.map(mapTeacher),
  subject: mapSubject(raw.subject),
  cabinet: mapCabinet(raw.cabinet),
  unionGroups: raw.union_groups.map(mapUnionGroup)
});
```

Валидация

Схемы валидации


```

const groupSchema = {
  id: { type: 'number', required: true },
  name: { type: 'string', required: true },
  category: { type: 'string', required: true }
};

const lessonSchema = {
  id: { type: 'string', required: true },
  weekday: { type: 'number', min: 1, max: 7, required: true },
  lesson: { type: 'number', min: 1, required: true },
  startTime: { type: 'string', pattern: 'time', required: true },
  endTime: { type: 'string', pattern: 'time', required: true }
};

```

Хранение

Структуры хранения

```

// Ключи для AsyncStorage
const STORAGE_KEYS = {
  FAVORITES: '@favorites',
  SETTINGS: '@settings',
  SCHEDULE_CACHE: '@schedule_cache'
};

// Формат кэша
interface CacheItem<T> {
  data: T;
  timestamp: number;
  expiry: number;
}

```

Типы событий

События приложения

```
type AppEvent =  
  | { type: 'SCHEDULE_LOADED'; payload: ScheduleData }  
  | { type: 'FAVORITE_ADDED'; payload: FavoriteItem }  
  | { type: 'FAVORITE_REMOVED'; payload: { id: number; type:  
string } }  
  | { type: 'SETTINGS_UPDATED'; payload: Partial<AppSettings> };
```

Рекомендации

Лучшие практики

2. Валидировать данные при получении
3. Нормализовать данные перед сохранением
4. Использовать иммутабельные обновления
5. Документировать все интерфейсы

Обработка данных

1. Проверять наличие обязательных полей
2. Предоставлять значения по умолчанию
3. Обрабатывать краевые случаи
4. Логировать ошибки валидации
5. Поддерживать обратную совместимость

Сетевой слой

Конфигурация

Базовая настройка

```
// services/api/config.ts
export const API_CONFIG = {
  BASE_URL: process.env.EXPO_PUBLIC_API_BASE_URL,
  PUBLICATION_ID: process.env.EXPO_PUBLIC_PUBLICATION_ID,
  TIMEOUT: 10000, // 10 секунд
  RETRY_ATTEMPTS: 3,
};
```

Инстанс Axios

```
// services/api/instance.ts
import axios from 'axios';

export const apiInstance = axios.create({
  baseURL: API_CONFIG.BASE_URL,
  timeout: API_CONFIG.TIMEOUT,
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
  },
});

// Интерцептор запросов
apiInstance.interceptors.request.use(
  (config) => {
    // Добавление publication_id
    config.params = {
      ...config.params,
      publication_id: API_CONFIG.PUBLICATION_ID,
    };
  },
```

```

    return config;
  },
  (error) => Promise.reject(error)
);

// Интерцептор ответов
apiInstance.interceptors.response.use(
  (response) => response,
  (error) => handleApiError(error)
);

```

API Сервисы

Расписание

```

// services/api/scheduleService.ts
export class ScheduleService {
  static async getGroupSchedule(groupId: number, nextWeek:
boolean): Promise<GroupSchedule> {
    try {
      const response = await
apiInstance.get(`/schedule/group/${groupId}`, {
        params: { next_week: nextWeek }
      });
      return response.data;
    } catch (error) {
      throw handleScheduleError(error);
    }
  }

  static async getTeacherSchedule(teacherId: number, nextWeek:
boolean): Promise<TeacherSchedule> {
    try {
      const response = await
apiInstance.get(`/schedule/teacher/${teacherId}`, {
        params: { next_week: nextWeek }
      });
    }
  }

```

```

        return response.data;
    } catch (error) {
        throw handleScheduleError(error);
    }
}
}

```

Обработка ошибок

Типы ошибок

```

// types/api.ts
export enum ApiErrorType {
    NETWORK = 'NETWORK_ERROR',
    TIMEOUT = 'TIMEOUT_ERROR',
    SERVER = 'SERVER_ERROR',
    NOT_FOUND = 'NOT_FOUND',
    VALIDATION = 'VALIDATION_ERROR',
}

export class ApiError extends Error {
    constructor(
        public type: ApiErrorType,
        public message: string,
        public statusCode?: number,
        public originalError?: any
    ) {
        super(message);
        this.name = 'ApiError';
    }
}

```

Обработчик ошибок

```

// services/api/errorHandler.ts
export const handleApiError = (error: any): never => {
    if (axios.isAxiosError(error)) {

```

```

    if (!error.response) {
      throw new ApiError(
        ApiErrorType.NETWORK,
        'Ошибка сети. Проверьте подключение к интернету.'
      );
    }

    const { response } = error;
    switch (response.status) {
      case 404:
        throw new ApiError(
          ApiErrorType.NOT_FOUND,
          'Запрашиваемые данные не найдены.',
          404
        );
      case 422:
        throw new ApiError(
          ApiErrorType.VALIDATION,
          'Ошибка валидации данных.',
          422,
          response.data.errors
        );
      default:
        throw new ApiError(
          ApiErrorType.SERVER,
          'Ошибка сервера.',
          response.status
        );
    }
  }

  throw new ApiError(
    ApiErrorType.NETWORK,
    'Неизвестная ошибка сети.'
  );
};

```

Кэширование

Стратегия кэширования

```
// services/cache/scheduleCache.ts
export class ScheduleCache {
  private static CACHE_TTL = 5 * 60 * 1000; // 5 минут
  private static cache = new Map<string, CacheItem>();

  static async get(key: string): Promise<any | null> {
    const item = this.cache.get(key);

    if (!item) return null;

    if (Date.now() - item.timestamp > this.CACHE_TTL) {
      this.cache.delete(key);
      return null;
    }

    return item.data;
  }

  static set(key: string, data: any): void {
    this.cache.set(key, {
      data,
      timestamp: Date.now()
    });
  }

  static clear(): void {
    this.cache.clear();
  }
}
```

Retry механизм

Реализация повторных попыток

```
// services/api/retry.ts
export const withRetry = async <T>(<
  operation: () => Promise<T>,
  retryCount: number = API_CONFIG.RETRY_ATTEMPTS
>): Promise<T> => {
  let lastError: Error;

  for (let i = 0; i < retryCount; i++) {
    try {
      return await operation();
    } catch (error) {
      lastError = error;

      if (!isRetryableError(error)) {
        throw error;
      }

      await delay(getRetryDelay(i));
    }
  }

  throw lastError;
};

const isRetryableError = (error: any): boolean => {
  if (error instanceof ApiError) {
    return error.type === ApiErrorType.NETWORK ||
      error.type === ApiErrorType.TIMEOUT;
  }
  return false;
};

const getRetryDelay = (attempt: number): number => {
  return Math.min(1000 * Math.pow(2, attempt), 10000);
};
```


Мониторинг

Логирование запросов

```
// services/api/logging.ts
export const logApiRequest = (config: any) => {
  if (__DEV__) {
    console.log(`API Request: ${config.method.toUpperCase()}
    ${config.url}`);
    console.log('Params:', config.params);
    console.log('Data:', config.data);
  }
};

export const logApiResponse = (response: any) => {
  if (__DEV__) {
    console.log(`API Response: ${response.status}`);
    console.log('Data:', response.data);
  }
};
```

Оптимизация

Параллельные запросы

```
// services/api/parallel.ts
export const fetchParallelData = async (groupIds: number[]):
Promise<GroupSchedule[]> => {
  const promises = groupIds.map(id =>
    ScheduleService.getGroupSchedule(id, false)
  );

  return Promise.all(promises);
};
```

Отмена запросов

```

// hooks/useSchedule.ts
export const useSchedule = (groupId: number) => {
  const abortController = useRef<AbortController>();

  useEffect(() => {
    abortController.current = new AbortController();

    const fetchSchedule = async () => {
      try {
        const response = await
apiInstance.get(`/schedule/group/${groupId}`, {
          signal: abortController.current.signal
        });
        // Обработка ответа
      } catch (error) {
        if (axios.isCancel(error)) {
          console.log('Request canceled');
        }
      }
    };

    fetchSchedule();

    return () => {
      abortController.current?.abort();
    };
  }, [groupId]);
};

```

Рекомендации

Лучшие практики

1. Использовать единый инстанс axios
2. Реализовать обработку ошибок

3. Внедрить кэширование
4. Добавить retry механизм
5. Логировать запросы в development

Безопасность

1. Валидация входных данных
2. Санитизация ответов
3. Защита от CSRF
4. Обработка чувствительных данных
5. Таймауты запросов

Оптимизация производительности

Общие принципы

Метрики производительности

- Время до интерактивности (TTI)
- Время первой отрисовки (FP)
- Время блокировки основного потока
- Частота кадров (FPS)
- Потребление памяти

Оптимизация рендеринга

Мемоизация компонентов

```
// Мемоизация компонента GroupList
const MemoizedGroupList = React.memo(GroupList, (prevProps,
nextProps) => {
  return prevProps.groups === nextProps.groups;
});

// Мемоизация хуков
const filteredGroups = useMemo(() => {
  return groups.filter(group =>
    group.name.toLowerCase().includes(searchQuery.toLowerCase())
  );
}, [groups, searchQuery]);
```

Виртуализация списков

```
// Оптимизация длинных списков
const VirtualizedGroupList: React.FC<GroupListProps> = ({ groups
}) => {
  return (
    <VirtualizedList
      data={groups}
      renderItem={({ item }) => <GroupItem group={item} />}
      getItemCount={({data}) => data.length}
      getItem={({data, index}) => data[index]}
      initialNumToRender={10}
      maxToRenderPerBatch={5}
      windowSize={5}
      updateCellsBatchingPeriod={50}
    />
  );
};
```

Ленивая загрузка

```
// Ленивая загрузка компонентов
const LazyScheduleView = React.lazy(() =>
import('./ScheduleView'));

// Использование
<Suspense fallback={<LoadingSpinner />}>
  <LazyScheduleView />
</Suspense>
```

Оптимизация изображений

Кэширование изображений

```
// Использование expo-image
import { Image } from 'expo-image';

const CachedImage: React.FC<ImageProps> = ({ uri, ...props }) => {
  return (
```

```

    <Image
      source={uri}
      cachePolicy="memory-disk"
      transition={200}
      {...props}
    />
  );
};

```

Предзагрузка ресурсов

```

import * as Font from 'expo-font';
import { Asset } from 'expo-asset';

const cacheResources = async () => {
  const imageAssets = Assets.map((asset) => {
    return Asset.fromModule(asset).downloadAsync();
  });

  const fontAssets = Font.loadAsync({
    'custom-font': require('./assets/fonts/custom-font.ttf'),
  });

  await Promise.all([...imageAssets, fontAssets]);
};

```

Оптимизация состояния

Управление обновлениями

```

// Предотвращение лишних обновлений
const GroupItem: React.FC<GroupItemProps> = React.memo(({ group })
=> {
  const { isFavorite } = useFavoritesContext();

  const favoriteStatus = useMemo(() =>
    isFavorite(group.id, 'group'),
    [group.id, isFavorite]
  );

```

```

);

return (
  <View>
    <Text>{group.name}</Text>
    <FavoriteIcon active={favoriteStatus} />
  </View>
);
});

```

Батчинг обновлений

```

// Группировка обновлений состояния
const handleMultipleUpdates = () => {
  batch(() => {
    setLoading(false);
    setData(newData);
    setError(null);
  });
};

```

Оптимизация сети

Кэширование запросов

```

// Кэширование ответов API
const scheduleCache = new Map<string, {
  data: ScheduleData;
  timestamp: number;
}>();

const getScheduleWithCache = async (groupId: number) => {
  const cacheKey = `schedule_${groupId}`;
  const cached = scheduleCache.get(cacheKey);

  if (cached && Date.now() - cached.timestamp < CACHE_TTL) {
    return cached.data;
  }

```

```

const data = await fetchSchedule(groupId);
scheduleCache.set(cacheKey, {
  data,
  timestamp: Date.now()
});

return data;
};

```

Предзагрузка данных

```

// Предварительная загрузка данных
const prefetchSchedule = async (groupId: number) => {
  try {
    const data = await fetchSchedule(groupId);
    scheduleCache.set(`schedule_${groupId}`, {
      data,
      timestamp: Date.now()
    });
  } catch (error) {
    console.error('Prefetch error:', error);
  }
};

```

Профилирование

React Native Profiler

```

// Использование Profiler
<Profiler id="GroupList" onRender={({id, phase, actualDuration}) => {
  console.log(`Component ${id} took ${actualDuration}ms to ${phase}`);
}}>
  <GroupList />
</Profiler>

```


Метрики производительности

```
import { PerformanceObserver, performance } from 'react-native-performance';

// Отслеживание производительности
const observer = new PerformanceObserver((list) => {
  const entries = list.getEntries();
  entries.forEach((entry) => {
    console.log(`${entry.name}: ${entry.duration}ms`);
  });
});

observer.observe({ entryTypes: ['measure'] });

// Измерение времени операции
performance.mark('startOperation');
// ... выполнение операции
performance.mark('endOperation');
performance.measure('operationDuration', 'startOperation', 'endOperation');
```

Оптимизация памяти

Очистка ресурсов

```
// Правильная очистка в useEffect
useEffect(() => {
  const subscription = EventEmitter.addListener('event', handler);

  return () => {
    subscription.remove();
  };
}, []);
```

Управление размером списков

```
// Пагинация для длинных списков
const PaginatedList: React.FC = () => {
  const [page, setPage] = useState(1);
  const [data, setData] = useState<Item[]>([]);

  const loadMore = useCallback(() => {
    setPage(prev => prev + 1);
  }, []);

  return (
    <FlatList
      data={data}
      onEndReached={loadMore}
      onEndReachedThreshold={0.5}
      removeClippedSubviews={true}
      maxToRenderPerBatch={10}
    />
  );
};
```

Рекомендации

Лучшие практики

1. Использовать мемоизацию для тяжелых вычислений
2. Применять виртуализацию для длинных списков
3. Кэшировать сетевые запросы
4. Оптимизировать изображения
5. Следить за утечками памяти

Инструменты разработчика

1. React Native Debugger

2. Chrome Performance Tools

3. Expo Performance Monitor

4. React Profiler

5. Memory Profiler

Метрики для отслеживания

1. Время запуска приложения

2. FPS в критичных компонентах

3. Потребление памяти

4. Время ответа сети

5. Размер бандла

Руководство разработчика

Настройка окружения

Установка зависимостей

```
# Установка Expo CLI глобально
npm install -g expo-cli

# Клонирование репозитория
git clone https://github.com/y9tr3ble/schedule-app.git

# Переход в директорию проекта
cd schedule-app

# Установка зависимостей проекта
npm install
```

Запуск проекта

Режим разработки

```
# Запуск с Expo Go
npx expo start

# Запуск для Android
npx expo start --android

# Запуск для iOS
npx expo start --ios
```

Горячая перезагрузка

- Включена по умолчанию
- Двойное нажатие 'r' для ручной перезагрузки

- Shake устройства для меню разработчика

Структура проекта

Основные директории

```
app/                # Основные экраны и навигация
├─ (tabs)/          # Вкладки приложения
├─ schedule/        # Экраны расписания
└─ teacher/         # Экраны преподавателей

components/         # Переиспользуемые компоненты
contexts/           # React контексты
constants/          # Константы приложения
```

Ключевые файлы

```
app.json            # Конфигурация Expo
babel.config.js     # Конфигурация Babel
tsconfig.json       # Конфигурация TypeScript
package.json        # Зависимости и скрипты
```

Разработка

Создание нового компонента

```
import React from 'react';
import { StyleSheet, View } from 'react-native';
import { useTheme } from 'react-native-paper';

interface Props {
  // Определение пропсов
}

export const NewComponent: React.FC<Props> = (props) => {
  const theme = useTheme();
```

```

    return (
      <View style={styles.container}>
        {/* Контент компонента */}
      </View>
    );
  };

  const styles = StyleSheet.create({
    container: {
      // Стили
    }
  });

```

Работа с контекстами

```

// Создание контекста
const MyContext = createContext<ContextType | undefined>
(undefined);

// Провайдер
export const MyProvider: React.FC = ({ children }) => {
  // Логика провайдера
  return (
    <MyContext.Provider value={value}>
      {children}
    </MyContext.Provider>
  );
};

// Хук для использования
export const useMyContext = () => {
  const context = useContext(MyContext);
  if (context === undefined) {
    throw new Error('useMyContext must be used within a
MyProvider');
  }
}

```

```
    return context;  
  };
```

Тестирование

Модульные тесты

```
# Запуск всех тестов  
npm test  
  
# Запуск с покрытием  
npm run test:coverage  
  
# Запуск конкретного теста  
npm test ComponentName.test.tsx
```

E2E тестирование

```
# Установка Detox  
npm install -g detox-cli  
  
# Сборка для тестирования  
detox build  
  
# Запуск тестов  
detox test
```

Сборка приложения

Android

```
# Создание релизной сборки  
eas build -p android --profile production  
  
# Локальная сборка APK  
npx expo run:android
```

iOS

```
# Создание релизной сборки  
eas build -p ios --profile production
```

```
# Локальная сборка  
npx expo run:ios
```

Деплой

Публикация в Ехро

```
# Публикация обновления  
expo publish  
  
# Публикация конкретного канала  
expo publish --release-channel production
```

App Store / Google Play

1. Создание релизной сборки
2. Тестирование сборки
3. Подготовка материалов
4. Публикация в магазинах

Рекомендации по разработке

Код стайл

- Использовать TypeScript
- Следовать принципам Clean Code
- Документировать код
- Использовать ESLint и Prettier

Производительность

- Мемоизация компонентов
- Оптимизация ререндеров
- Ленивая загрузка
- Кэширование данных

Безопасность

- Проверка входных данных
- Безопасное хранение данных
- Обработка ошибок
- Защита API ключей

Решение проблем

Общие проблемы

1. Очистка кэша

```
npx expo start -c
```

2. Пересборка проекта

```
npm run clean  
npm install
```

3. Обновление зависимостей

```
npm update
```

Отладка

- Использование React Native Debugger
- Console.log и отладочные точки
- Performance Monitor
- Network Inspector

Компоненты

Основные компоненты

GroupList

Компонент для отображения списка групп с поиском и фильтрацией.

```
interface GroupListProps {  
    groups: GroupInfo[];  
    onSelectGroup: (group: GroupInfo) => void;  
}
```

Особенности:

- Поиск по названию группы
- Сортировка по категориям
- Добавление в избранное
- Анимированные карточки
- Оптимизированная производительность

TeacherList

Компонент для отображения списка преподавателей.

```
interface TeacherListProps {  
    teachers: TeacherInfo[];  
    onSelectTeacher: (teacher: TeacherInfo) => void;  
}
```

Функционал:

- Поиск по ФИО и должности
- Алфавитная сортировка

- Управление избранным
- Анимации при взаимодействии
- Оптимизированный рендеринг

ScheduleView

Основной компонент для отображения расписания.

```
interface ScheduleViewProps {  
  data: GroupData | TeacherSchedule | null;  
  loading: boolean;  
  error: string | null;  
  isNextWeek: boolean;  
  isTeacherSchedule?: boolean;  
  onRefresh?: () => void;  
}
```

Возможности:

- Группировка по дням недели
- Pull-to-refresh обновление
- Индикация текущего занятия
- Анимированные переходы между состояниями
- Обработка ошибок и состояния загрузки

LessonCard

Компонент карточки отдельного занятия.

```
interface LessonCardProps {  
  lesson: Lesson;  
  isTeacherSchedule?: boolean;  
  isNextWeek?: boolean;  
}
```

Функционал:

- Отображение информации о занятии
 - Название предмета
 - Время проведения
 - Преподаватель
 - Аудитория
- Поддержка подгрупп
- Интерактивные элементы
- Индикация текущего занятия
- Адаптивный дизайн

WeekSelector

Компонент выбора недели расписания.

```
interface WeekSelectorProps {  
    isNextWeek: boolean;  
    onWeekChange: (isNext: boolean) => void;  
    loading?: boolean;  
}
```

Особенности:

- Переключение между неделями
- Анимированные переходы
- Индикация загрузки
- Отображение дат недели

Вспомогательные компоненты

MaterialSwitch

Кастомный компонент переключателя.

```
interface MaterialSwitchProps {  
  selected: boolean;  
  onPress: () => void;  
  disabled?: boolean;  
  fluid?: boolean;  
  switchOnIcon?: string;  
  switchOffIcon?: string;  
}
```

Функционал:

- Анимированное переключение
- Поддержка иконок
- Поддержка отключенного состояния
- Настраиваемые стили

MaterialSwitchListItem

Элемент списка с переключателем.

```
interface MaterialSwitchListItemProps {  
  title: string;  
  selected: boolean;  
  onPress: () => void;  
  disabled?: boolean;  
  fluid?: boolean;  
  switchOnIcon?: string;  
  switchOffIcon?: string;  
}
```

Особенности:

- Интеграция с MaterialSwitch
- Настраиваемый заголовок

- Поддержка иконок
- Адаптивный дизайн

Оптимизация производительности

Мемоизация

```
// Пример использования React.memo
const MemoizedComponent = React.memo(Component, (prevProps,
nextProps) => {
  // Логика сравнения props
});

// Пример использования useMemo
const memoizedValue = useMemo(() => computeExpensiveValue(a, b),
[a, b]);
```

Виртуализация списков

```
// Пример использования VirtualizedList
<VirtualizedList
  data={items}
  renderItem={renderItem}
  getItemCount={getItemCount}
  getItem={getItem}
  initialNumToRender={10}
  maxToRenderPerBatch={10}
  windowSize={5}
/>
```

Ленивая загрузка

```
// Пример ленивой загрузки компонента
const LazyComponent = React.lazy(() => import('./Component'));

// Использование
<Suspense fallback={<LoadingSpinner />}>
```

```
<LazyComponent />
</Suspense>
```

Стилизация компонентов

Использование тем

```
const Component = () => {
  const theme = useTheme();

  return (
    <View style={[styles.container, { backgroundColor:
theme.colors.background }]}>
      {/* Контент компонента */}
    </View>
  );
};
```

Адаптивные стили

```
const styles = StyleSheet.create({
  container: {
    ...Platform.select({
      ios: {
        shadowColor: '#000',
        shadowOffset: { width: 0, height: 2 },
        shadowOpacity: 0.1,
        shadowRadius: 4,
      },
      android: {
        elevation: 4,
      },
    }),
  },
});
```


API Документация

Конфигурация

```
const API_CONFIG = {  
  BASE_URL: process.env.EXPO_PUBLIC_API_BASE_URL,  
  PUBLICATION_ID: process.env.EXPO_PUBLIC_PUBLICATION_ID  
};
```

Основные методы

Расписание группы

```
getGroupSchedule(groupId: number, nextWeek: boolean):  
Promise<GroupData>
```

Параметры:

- `groupId`: ID группы
- `nextWeek`: флаг следующей недели

Ответ:

```
interface GroupData {  
  startDate: string;  
  group: {  
    id: number;  
    name: string;  
  };  
  lessons: Lesson[];  
}
```

Расписание преподавателя

```
getTeacherSchedule(teacherId: number, nextWeek: boolean):  
Promise<TeacherSchedule>
```

Параметры:

- `teacherId`: ID преподавателя
- `nextWeek`: флаг следующей недели

Ответ:

```
interface TeacherSchedule {  
  startDate: string;  
  teacher: TeacherInfo;  
  lessons: Lesson[];  
}
```

Модели данных

Lesson

```
interface Lesson {  
  id: string;  
  weekday: number;  
  lesson: number;  
  startTime: string;  
  endTime: string;  
  teachers: Teacher[];  
  subject: Subject;  
  cabinet: Cabinet;  
  unionGroups: UnionGroup[];  
}
```

Teacher

```
interface Teacher {  
  id: number;
```

```
fio: string;  
position: string;  
}
```

Обработка ошибок

Типы ошибок

- Сетевые ошибки (NetworkError)
- Ошибки сервера (ServerError)
- Ошибки валидации (ValidationError)

Обработка

```
try {  
  const data = await getGroupSchedule(groupId, false);  
} catch (error) {  
  if (error instanceof NetworkError) {  
    // Обработка сетевой ошибки  
  } else if (error instanceof ServerError) {  
    // Обработка серверной ошибки  
  }  
}
```

Кэширование

Стратегии кэширования

- In-memory кэш для быстрого доступа
- AsyncStorage для персистентного хранения
- Stale-while-revalidate для оптимизации загрузки

Пример реализации кэша

```

const CACHE_TTL = 5 * 60 * 1000; // 5 минут

class ScheduleCache {
  private cache: Map<string, {
    data: any;
    timestamp: number;
  }>;

  constructor() {
    this.cache = new Map();
  }

  set(key: string, data: any) {
    this.cache.set(key, {
      data,
      timestamp: Date.now()
    });
  }

  get(key: string) {
    const cached = this.cache.get(key);
    if (!cached) return null;

    if (Date.now() - cached.timestamp > CACHE_TTL) {
      this.cache.delete(key);
      return null;
    }

    return cached.data;
  }
}

```

Утилиты API

Форматирование данных

```
export const formatScheduleData = (data: RawScheduleData):  
FormattedSchedule => {  
  // Логика форматирования данных  
};
```

Валидация

```
export const validateScheduleData = (data: any): boolean => {  
  // Логика валидации  
};
```

Примеры использования

Получение расписания группы

```
const getGroupScheduleWithCache = async (groupId: number,  
nextWeek: boolean) => {  
  const cacheKey = `group_${groupId}_${nextWeek}`;  
  const cached = scheduleCache.get(cacheKey);  
  
  if (cached) {  
    return cached;  
  }  
  
  const data = await getGroupSchedule(groupId, nextWeek);  
  scheduleCache.set(cacheKey, data);  
  return data;  
};
```