






Документация "Расписание для Хекслет"

Table of Contents

Руководство разработчика	2
 Стилизация	9
 Документация проекта "Расписание для Хекслет"	17
 Компоненты	27
 Общая информация	33
API Документация	39
 Архитектура приложения	44

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



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

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
- Жизненный цикл
 - Инициализация
 - Основной цикл

- Обновление данных
- Обработка ошибок
- Поддерживаемые платформы
 - iOS требования
 - Android требования
 - Особенности платформ

Архитектура

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

Руководство разработчика

- Настройка окружения
 - Системные требования
 - Необходимое ПО
 - Конфигурация IDE
- Установка и запуск

- Клонирование репозитория
- Установка зависимостей
- Запуск проекта
- Отладка
- Разработка
 - Структура проекта
 - Создание компонентов
 - Работа с контекстами
 - Стилизация
- Тестирование
 - Модульные тесты
 - Интеграционные тесты
 - E2E тестирование
 - Покрытие кода
- Сборка и деплой
 - Сборка для Android
 - Сборка для iOS
 - Публикация в магазинах
 - CI/CD

Компоненты

- Основные компоненты
 - GroupList
 - Структура

- Пропсы
- Методы
- Примеры использования
- TeacherList
 - Конфигурация
 - События
 - Оптимизация
- ScheduleView
 - Отображение данных
 - Обработка ошибок
 - Обновление
- LessonCard
 - Структура карточки
 - Взаимодействие
 - Анимации
- WeekSelector
 - Выбор недели
 - Навигация
 - Состояния
- Вспомогательные компоненты
 - MaterialSwitch
 - Настройка
 - Темизация

- MaterialSwitchListItem
 - Интеграция
 - Кастомизация
- Оптимизация
 - Мемоизация
 - Виртуализация
 - Ленивая загрузка
 - Кэширование

API

- Конфигурация
 - Базовые настройки
 - Переменные окружения
 - Заголовки запросов
- Основные методы
 - Расписание группы
 - Параметры
 - Ответы
 - Ошибки
 - Расписание преподавателя
 - Структура запроса
 - Обработка ответа
 - Валидация
- Модели данных

- Группы
- Преподаватели
- Расписание
- Занятия
- Обработка ошибок
 - Типы ошибок
 - Обработчики
 - Retry стратегии
- Кэширование
 - Стратегии
 - Инвалидация
 - Персистентность
- Безопасность
 - Аутентификация
 - Авторизация
 - Защита данных


Стилизация

- Дизайн система
 - Цветовая палитра
 - Типографика
 - Отступы
 - Тени
- Темы


- Светлая тема
- Темная тема
- Системная тема
- Кастомные темы
- Компоненты UI
 - Карточки
 - Кнопки
 - Поля ввода
 - Списки
 - Модальные окна
- Анимации
 - Переходы
 - Интерактивные элементы
 - Загрузка
 - Обратная связь
- Адаптивный дизайн
 - Размеры экранов
 - Ориентация
 - Доступность
 - Масштабирование

Быстрые ссылки


Начало работы

- Установка проекта ([Руководство разработчика](#))
- Первые шаги ([Руководство разработчика](#))
- Структура проекта ( [Общая информация](#))
- Руководство по стилю ([Руководство разработчика](#))

Разработка

- Создание компонентов ([Руководство разработчика](#))
- Работа с API ([API Документация](#))
- Стилизация ( [Стилизация](#))
- Тестирование ([Руководство разработчика](#))

Продвинутые темы

- Оптимизация производительности ( [Компоненты](#))
- Безопасность ([API Документация](#))
- CI/CD ([Руководство разработчика](#))
- Мониторинг ([Руководство разработчика](#))

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?

- Создать файл в `/components`
- Добавить типы
- Написать тесты
- Обновить документацию

2. Как работать с API?

- Использовать сервисы из `/services/api`
- Обрабатывать ошибки
- Эшировать данные

Деплой







1. Как собрать приложение?

- Следовать инструкции в разделе сборки ([Руководство разработчика](#))

2. Как опубликовать обновление?

- Обновить версию
- Собрать релиз
- Отправить в магазины

Архитектура

- Общая структура ( [Архитектура приложения](#))
- Компонентная архитектура ( [Архитектура приложения](#))
- Потоки данных ( [Архитектура приложения](#))
- Масштабируемость ( [Архитектура приложения](#))
 - Жизненный цикл приложения ( [Общая информация](#))
 - Управление состоянием ( [Общая информация](#))

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,
      },
    }),
  },
});
```

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

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;  
};
```



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>;  
}
```

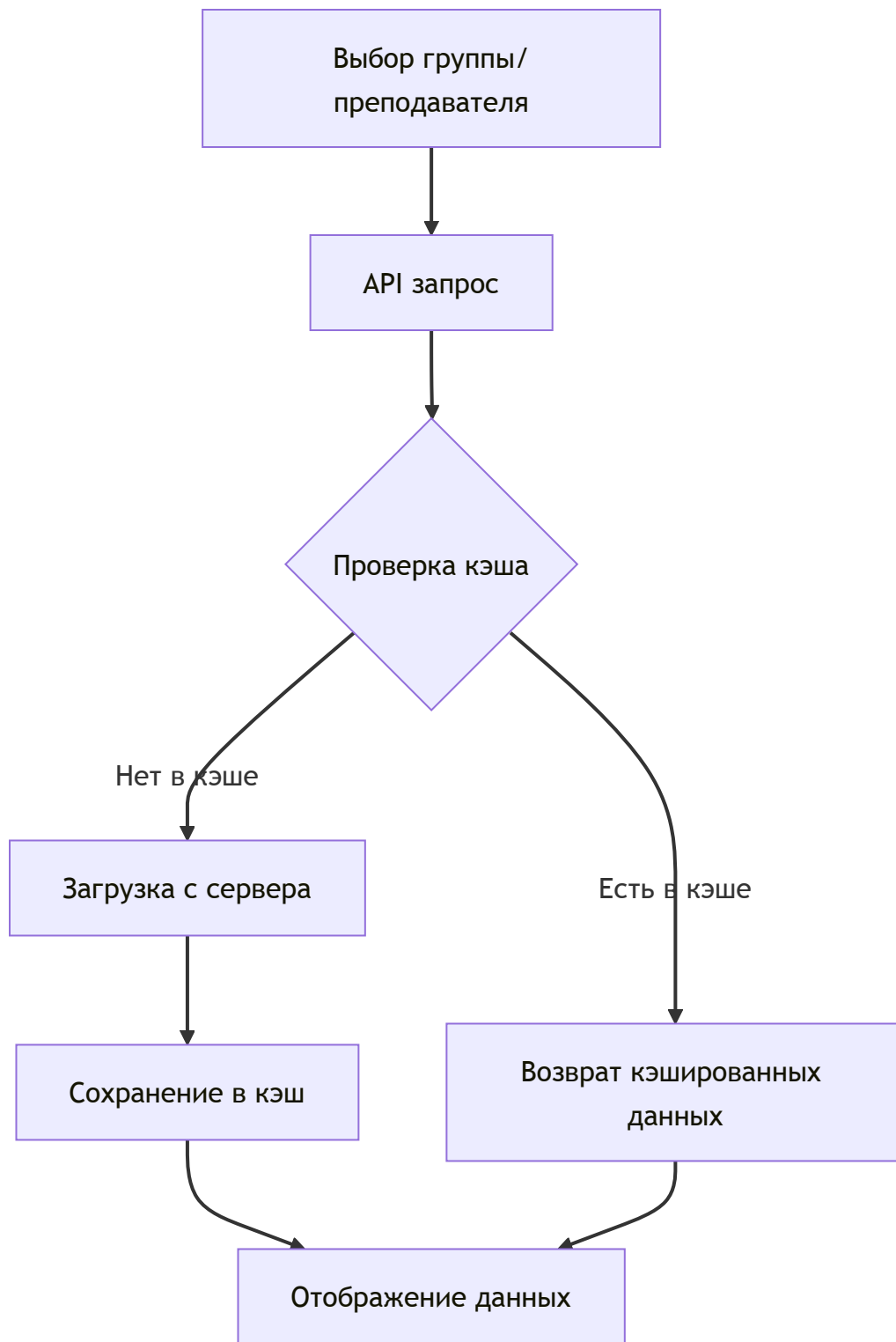
ScheduleSettingsContext

Настройки отображения расписания:

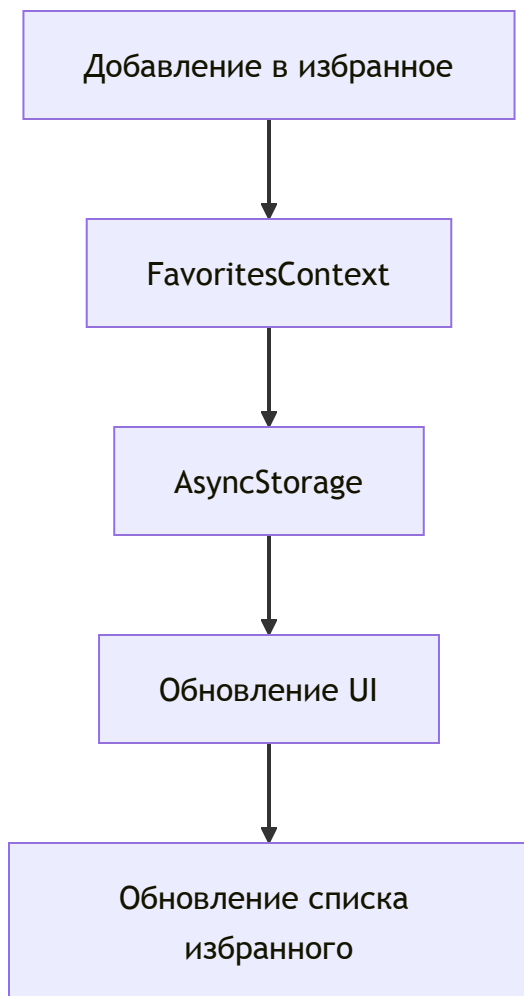
```
interface ScheduleSettings {  
    showCabinetNumbers: boolean;  
    showTeacherNames: boolean;  
    compactMode: boolean;  
    showLessonNumbers: boolean;  
}
```

Потоки данных

Загрузка расписания



Управление избранным



Компонентная архитектура

Основные компоненты

ScheduleView

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

GroupList/TeacherList

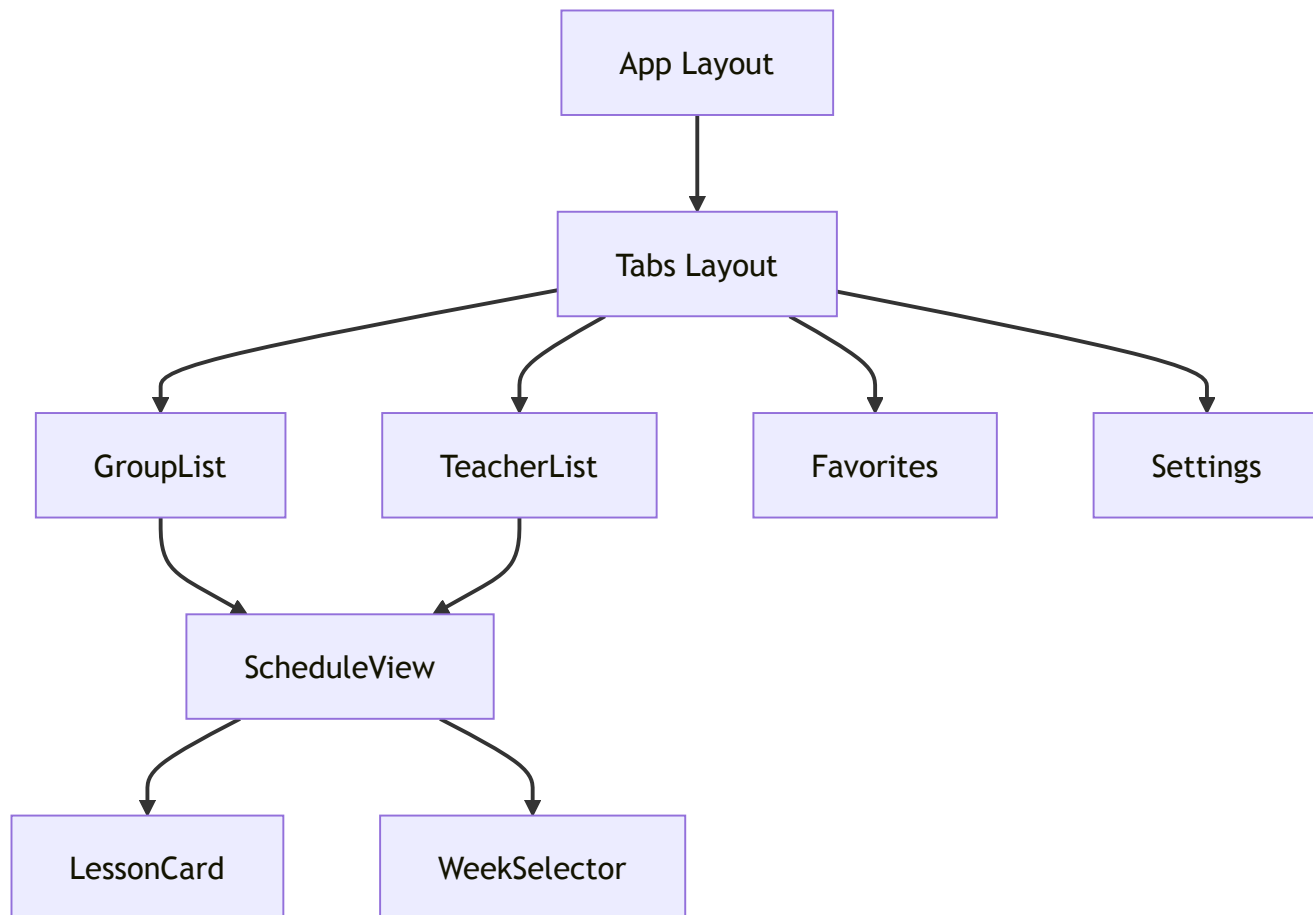
- Отображение списков

- 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