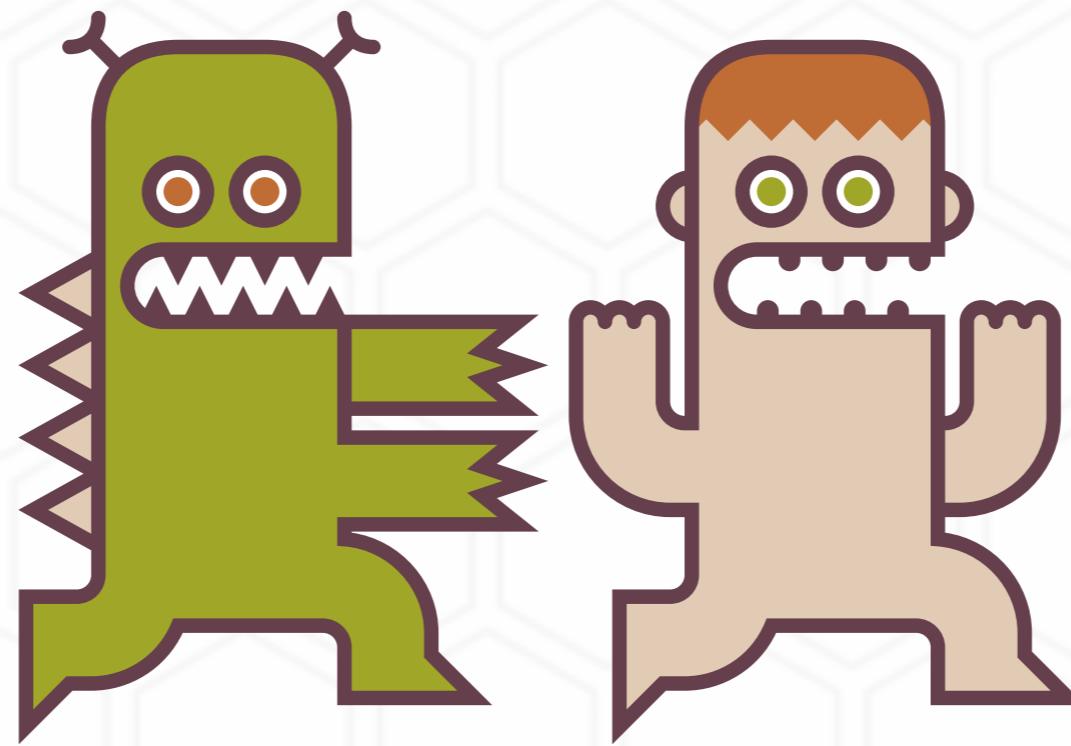




Как построить Гексагон: прагматичная архитектура для клиентских приложений

Александр Мадьянкин
Злые марсиане



EVIL MARTIANS

evilmartians.com



fountain

ENTY



ManageBac



GROUPON

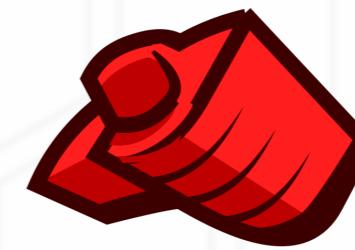
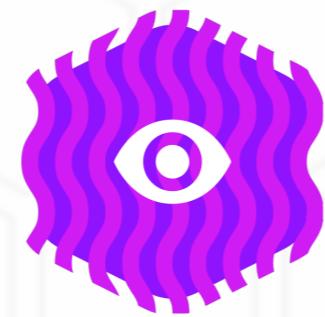
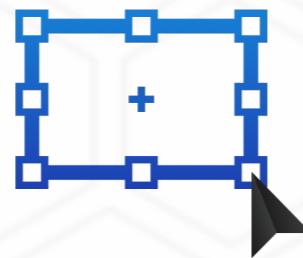


PODIUM



Rocketbank

evilmartians.com



Будет полезно

- Если вы тимлид/техлид
- Если вы еще не писали SPA
- Если вы писали SPA, но еще не выработали методику
- Если вам интересна архитектура приложений

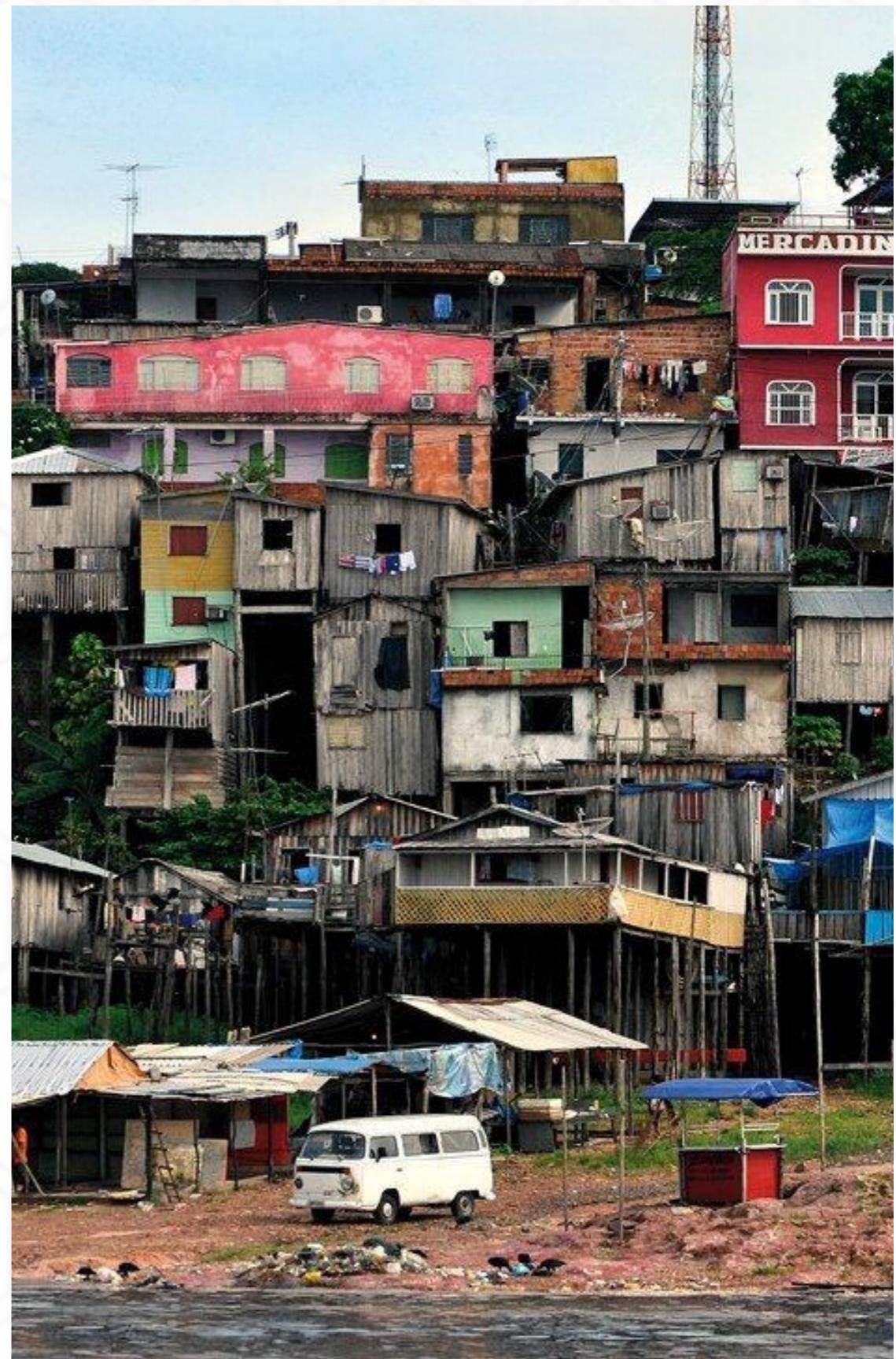
Зачем?

- Выработать четкие правила организации кода
- Узнать, как можно упростить внесение изменений
- Облегчить вхождение в проект
- Выделить переиспользуемый код
- Минимизировать технический долг и баги

План

- Частые ошибки
- Первое приложение и его архитектура
- Собранные грабли
- Второе приложение и гексагональная архитектура
- Практическое применение

Архитектура типового SPA



Tutorial Driven Development

- Не успеваем разобраться, как следует
- Копируем демо-код из документации
- Строим на этом коде приложение

Hype Driven Development

- Смотрим на звездочки
- Смотрим на твиты и посты
- Не смотрим на количество багов
- А как библиотека вообще решает задачу?

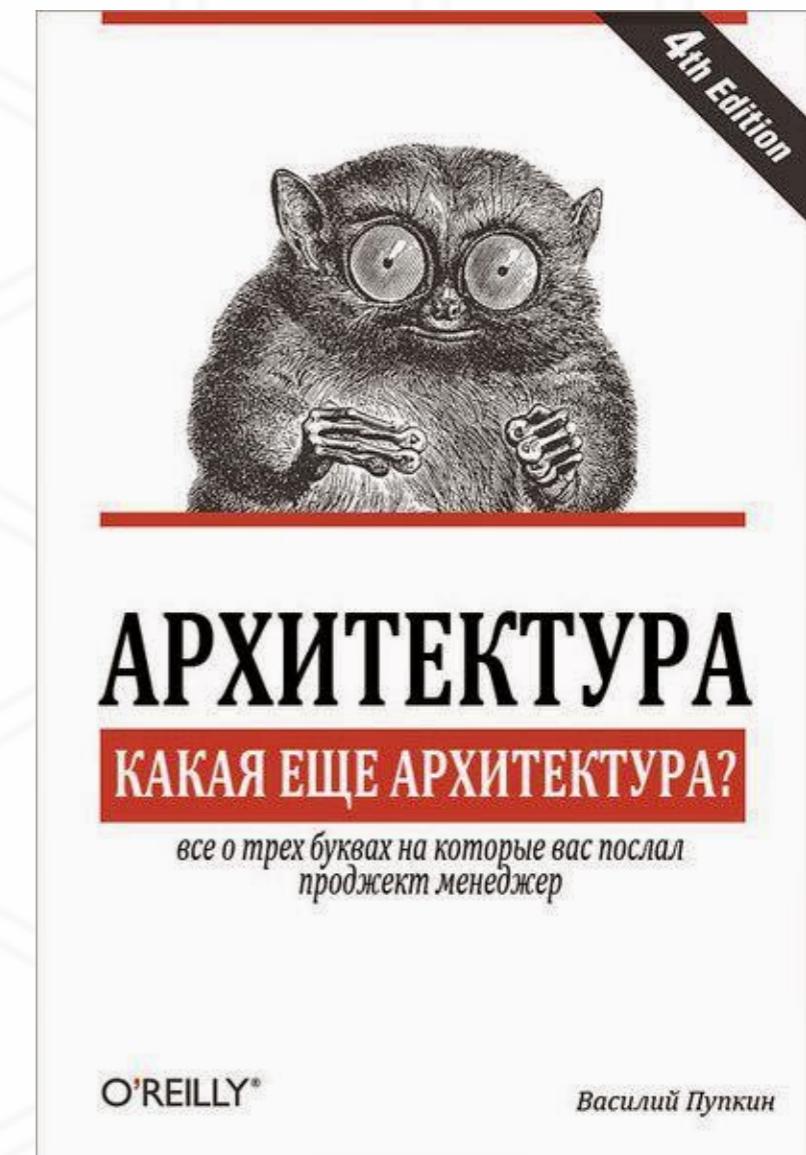
А-А-А! Надо быстрее релиз!

- Некогда точить пилу, надо пилиТЬ
- Пока пишем код, не видим общей картины
- Когда видим общую картину, уже поздно



Это выливается в проблемы

- Сильная зависимость от сторонних библиотек
- Код решает несколько задач
- Все зависит от всего



**Стыдиться нечего,
у всех такое бывает**

**Нужно делать выводы
и учиться**

Архитектура,
заявленная на
Apollo (или
любую другую
библиотеку)



at the MoMa

PM
Modern Art, 11 W 53rd St, New York, NY

Common's Member Experience
m is the creator and host of this
nt

April-June
2019

The Museum of Modern Art (MoMA) is one of
City's premiere modern art museums,
one of the largest and most influential
centers of modern art in the world.

Collection offers an overview of modern
contemporary art, including works of
sculpture and design, drawing, painting,
photography, prints, illustrated books
and books, film, and electronic media.
Includes approximately 300,000 books and
catalogs, over 1,000 periodical titles,
over 10,000 files of ephemera about
artists and groups.

Get and head over

Hey Moviepassers, who wants to check out
the new Marvel movie this weekend?

You 4:24PM 04/29/2019

Duane I'm in! 4:25PM 04/29/2019

Ben Sounds great, how's 7:30pm at Alamo?
4:27PM 04/29/2019

Duane Looks like they still have lots of good seats
available if we book it now
4:28PM 04/29/2019

Duane Let me ask my suitemate, he'll probably
want to join too
4:28PM 04/29/2019

+ The time works for me! We can grab a
quick dinner beforehand too. >

Perks

As a Common member, you get access to exclusive deals from local and national brands. Click on the logos below to learn more about each perk, or search by brand and category.

Name X

All types

Artspace
Shopping

Blink Fitness
Health & Wellness

Brooklinen
Shopping

Brooklyn Boulders
Health & Wellness


Melissa Smith Chauncey Member Edit

Interest tags:
Cooking MLB Running Soccer Travel
YouTube

[Manage my tags](#)

Chauncey
Member since February 2019

Hi everyone! I work as a consultant so I'm travel often, but I love meeting new people and chat when I'm around. Feel free to message me if you need any travel tips! I'm also an amateur coo

Приложение

- Приложение для съемщиков коммунального жилья
- Чтобы общаться с арендодателем
- Узнавать о мероприятиях и новостях
- Организовывать мероприятия
- Общаться с другими съемщиками

Условия задачи

- Приложение для съемщиков жилья с чатом
- Есть MVP в виде веб-приложения на React + Apollo
- Нужно сделать клон MVP на React Native
- Как можно быстрее
- Один разработчик в первую пару месяцев

Отличия от веб-приложения

- Выкинул apollo-link-state
- Оставил Apollo, чтобы копировать код из MVP
- Взял Redux
- Взял TypeScript
- Своя архитектура

Архитектура



Фичи

- Самостоятельный кусок функциональности
- Изолированы друг от друга
- Общаются через события
- Содержат запросы к серверу
- Содержат свое состояние

Работа с API

- К серверу ходим с помощью Apollo
- Контейнеры используют компоненты-обертки Apollo
- Данные от Apollo не хранятся в общем состоянии

Контейнеры

```
class PostsList extends Component {  
  public render() {  
    const { postsData } = this.props;  
  }  
}  
  
export default withPostsQuery(PostsList);
```

Получилось хорошо

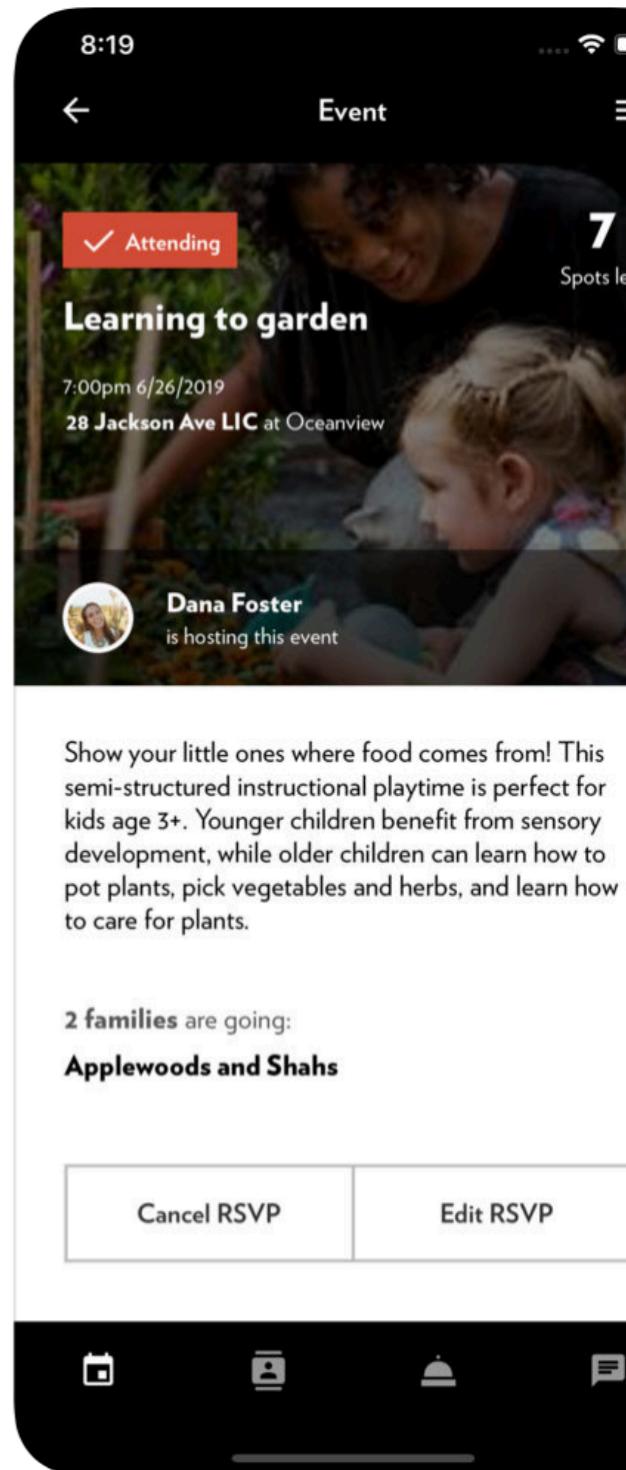
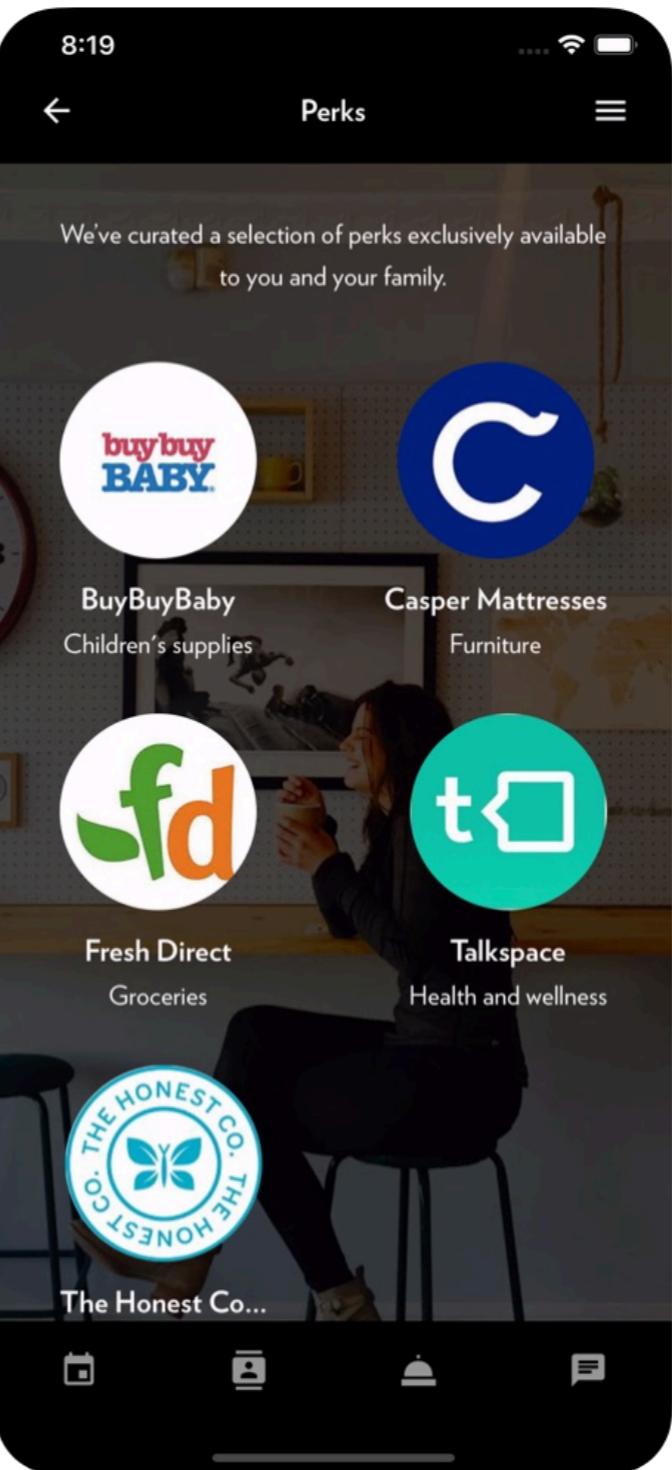
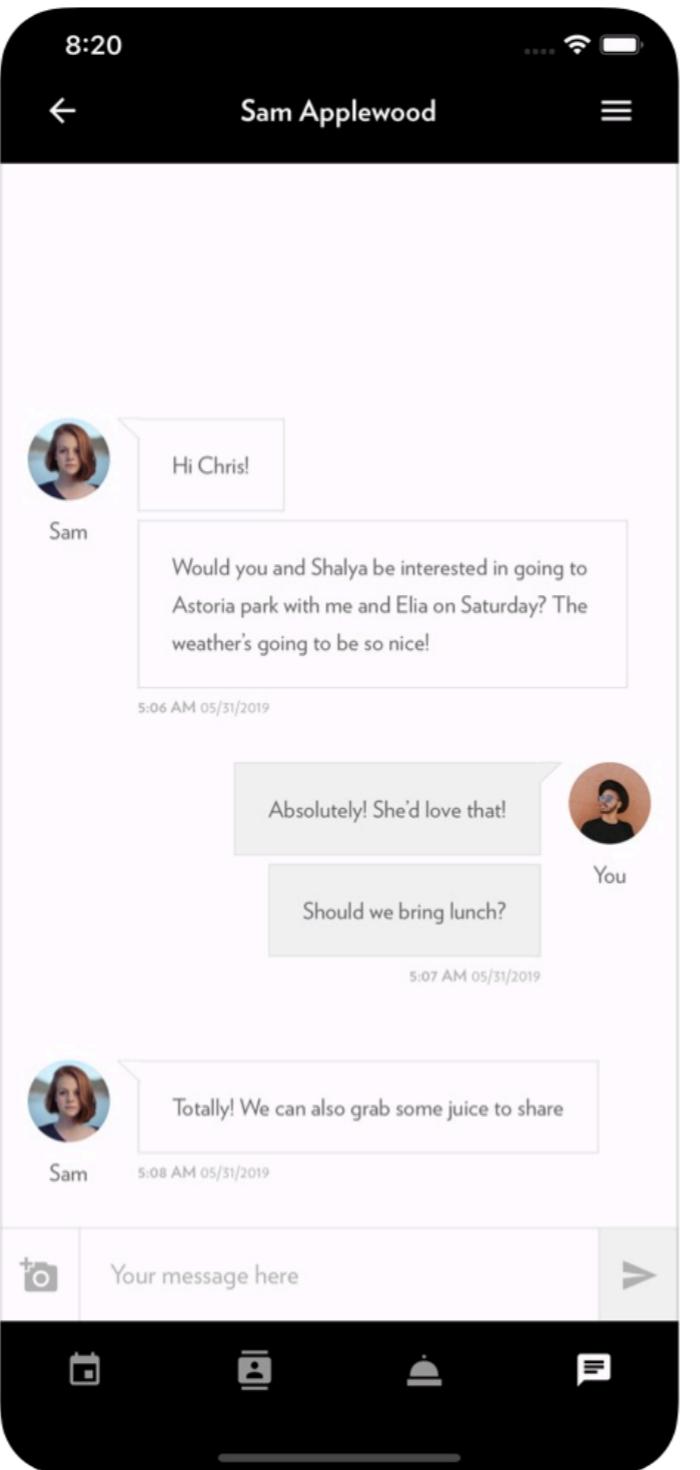
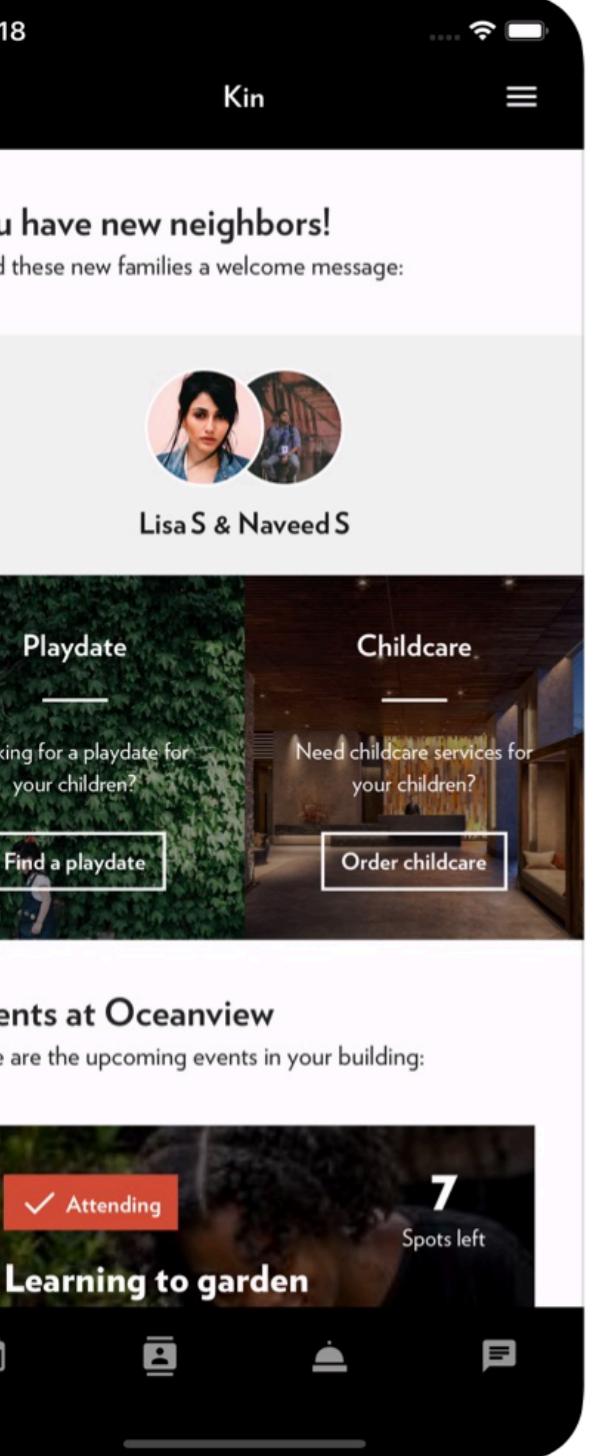
- Разделение кода по фичам
- Управление процессами
- Нет сложных абстракций

Грабли

- Дублирование и синхронизация данных
- Две платформы – два приложения
- Производительность
- Проблемы с Apollo, а выпилить уже нельзя
- Траты времени на отладку



**Вызов —
новое приложение за
полтора месяца**



Приложение

- То же самое, только для семей
- Общаться с арендодателем
- Узнавать о мероприятиях и новостях
- Организовывать дни рождения для детей
- Искать няньек
- Общаться с другими семьями

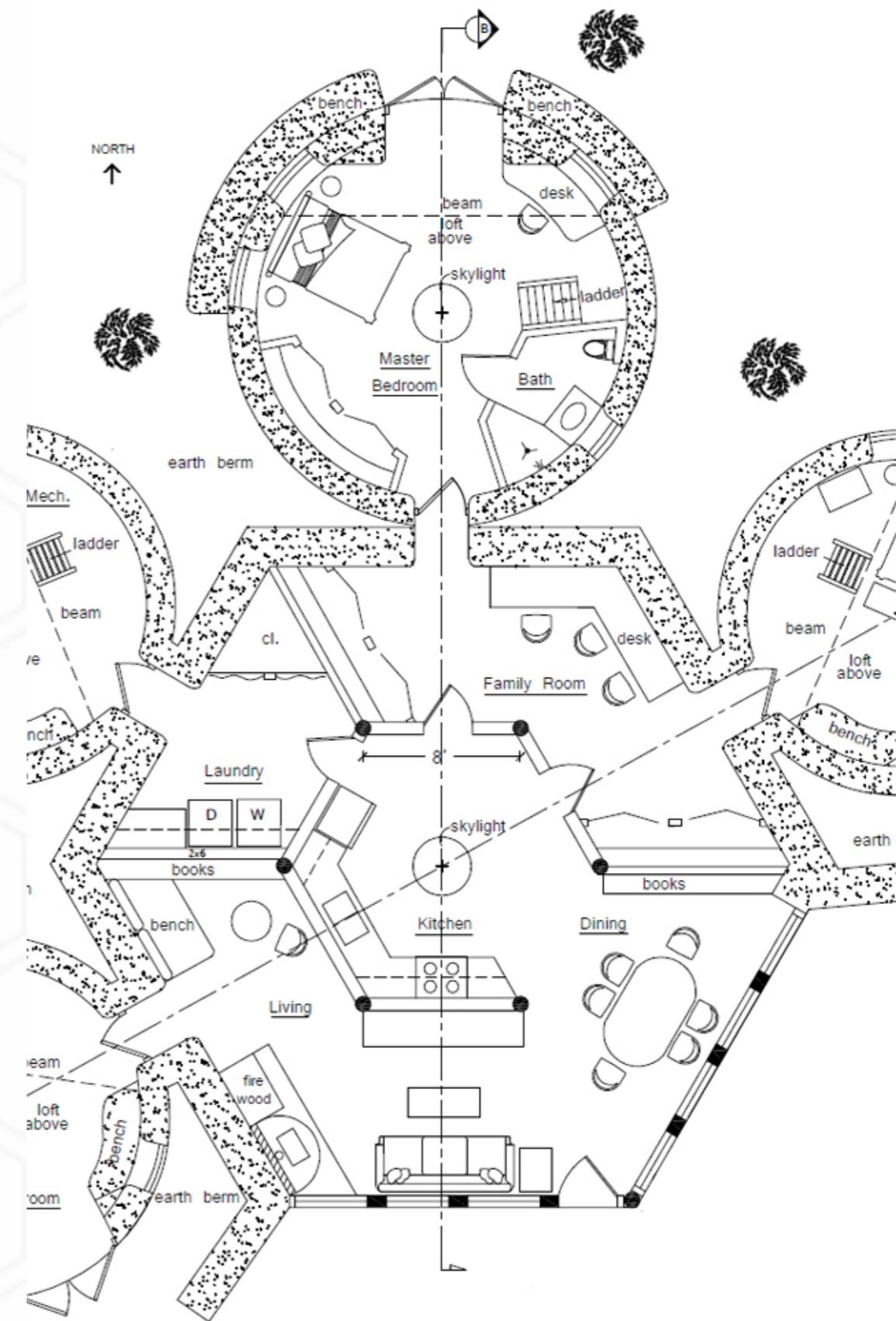
Условия задачи

- Пять разработчиков с разным опытом в RN
- Шесть недель на разработку
- Первые две недели – один разработчик
- Наделать как можно меньше багов и костылей
- Могут быть разные платформы

Архитектура ≠ Реализация



Архитектура



Что мы хотим?

- Уменьшить технический долг
- Уменьшить количество багов
- Кроссплатформенность
- Быстрый интерфейс
- Поддержка оффлайна без костылей

**Как написать
поддерживаемое
приложение?**

**Упростить внесение
изменений!**

**Упростить внесение
изменений!**



Kak?

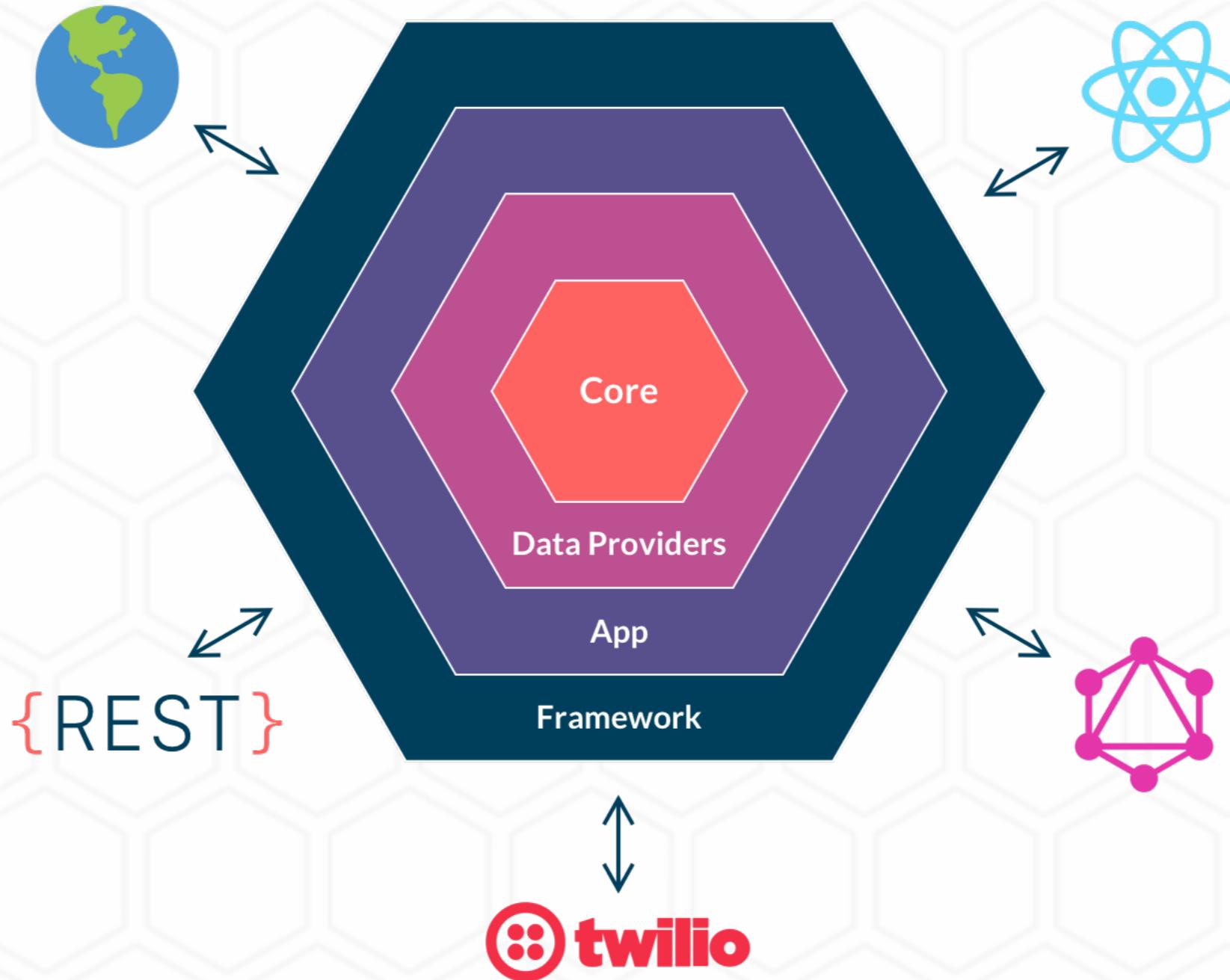
**Отделить неизменяемые
части от тех, что меняются**

**Отделить интерфейсы
от их реализаций**

**Отделить бизнес-логику
от UI и API**

Отделить логирование от основного кода

Гексагон – слоеный пирог



Гексагональная архитектура

- Алистер Кокберн
- Архитектура портов и адаптеров
- Число сторон не имеет значения
- Сторона — порт между приложением и миром
- Приложение делится на слои
- Более внешние слои знают больше о мире

Взаимодействие между слоями

- Между слоями есть границы
- Пересечение границ через порты (интерфейсы)
- Данные передаются через DTO (Data Transfer Objects)
- Зависимости передаются снаружи внутрь

Сколько слоев нужно?

- Зависит от приложения
- Минимум – два
- Больше слоев – больше границ
- Больше границ – больше возни с интерфейсами
- Опираемся на здравый смысл

ядро

Ядро

Провайдеры

Приложение

Фреймворк

Ядро



- Бизнес-логика и данные приложения
- Изолировано от API и графического интерфейса
- Команды для вызова действий
- Можно вынести в отдельную библиотеку

Бизнес-процессы



- Звоним в пиццерию
- Говорим, какую пиццу хотим
- Сообщаем, как будем оплачивать
- Сообщаем наш адрес
- Получаем время доставки заказа

Бизнес-процессы



- `getPizzaList()`
- `selectPizza(id)`
- `setPaymentMethod(PaymentMethod.Card)`
- `setAddress(addressData)`
- `getDeliveryTime()`

Ядро



Провайдеры данных

Ядро

Провайдеры

Приложение

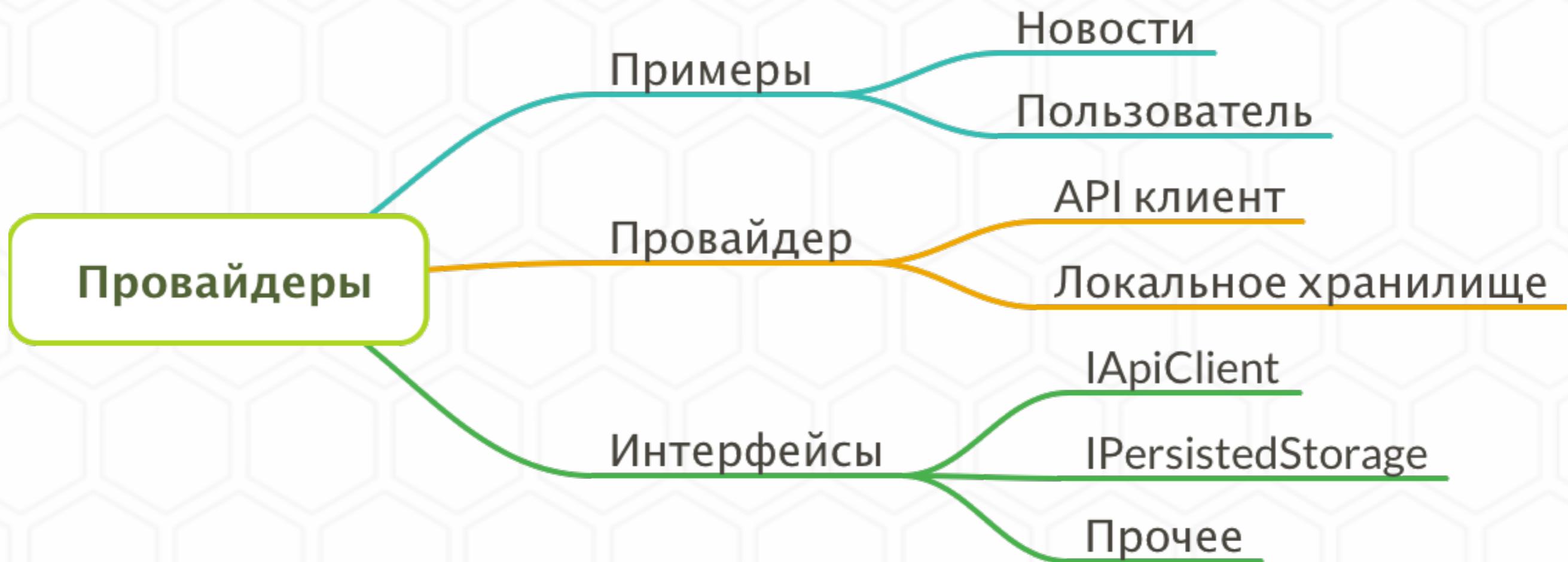
Фреймворк

Провайдеры данных



- Запросы к API и локальному хранилищу
- Перехват исключений при вызове API
- Валидация схемы данных
- Конвертация данных
- Можно вынести в отдельную библиотеку

Провайдеры данных



Приложение

Ядро

Провайдеры

Приложение

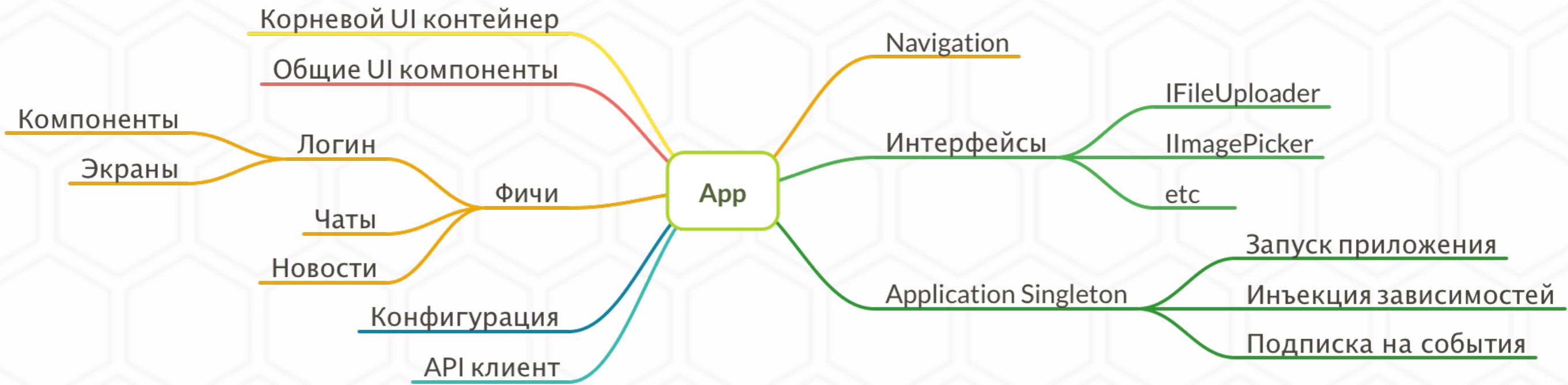
Фреймворк

Приложение



- Реализация для конкретной платформы
- Управление жизненным циклом приложения
- Пользовательский интерфейс
- Навигация
- Конфигурация
- Сложно перенести целиком на другую платформу

Приложение



Фреймворк

Ядро

Провайдеры

Приложение

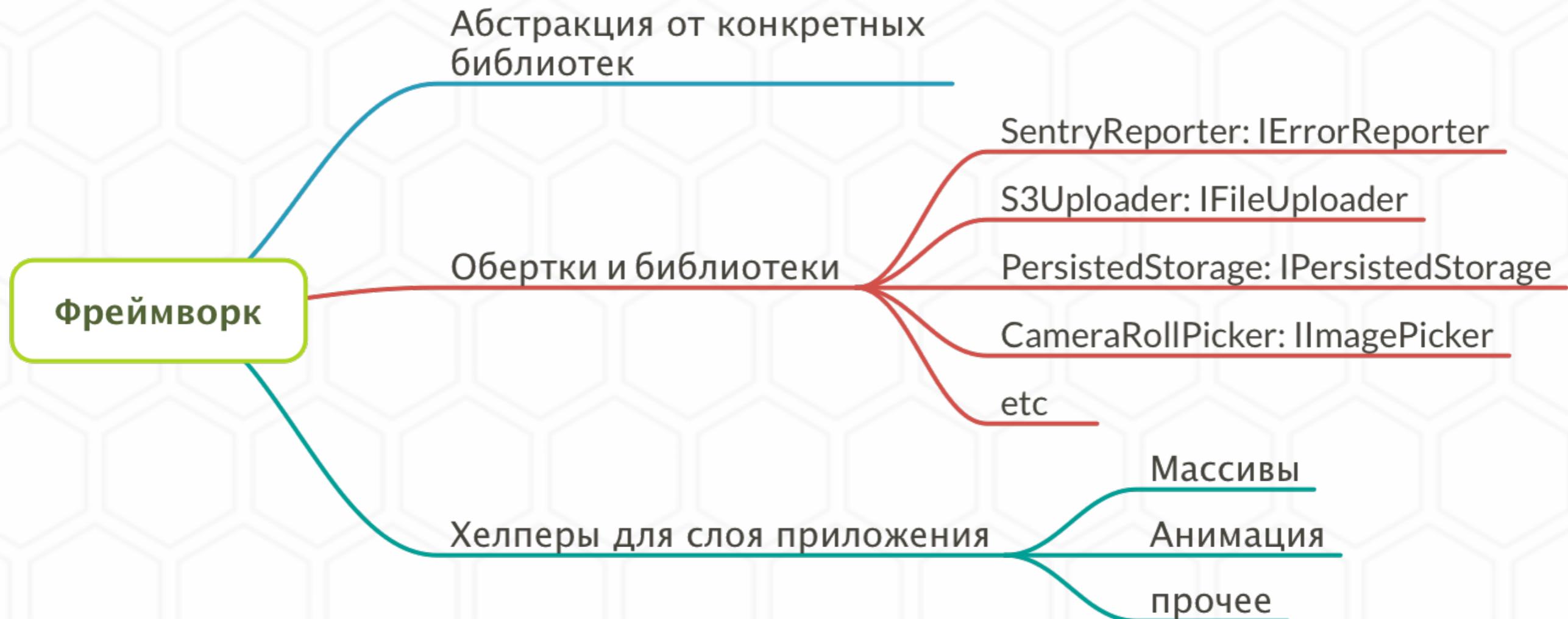
Фреймворк

Фреймворк

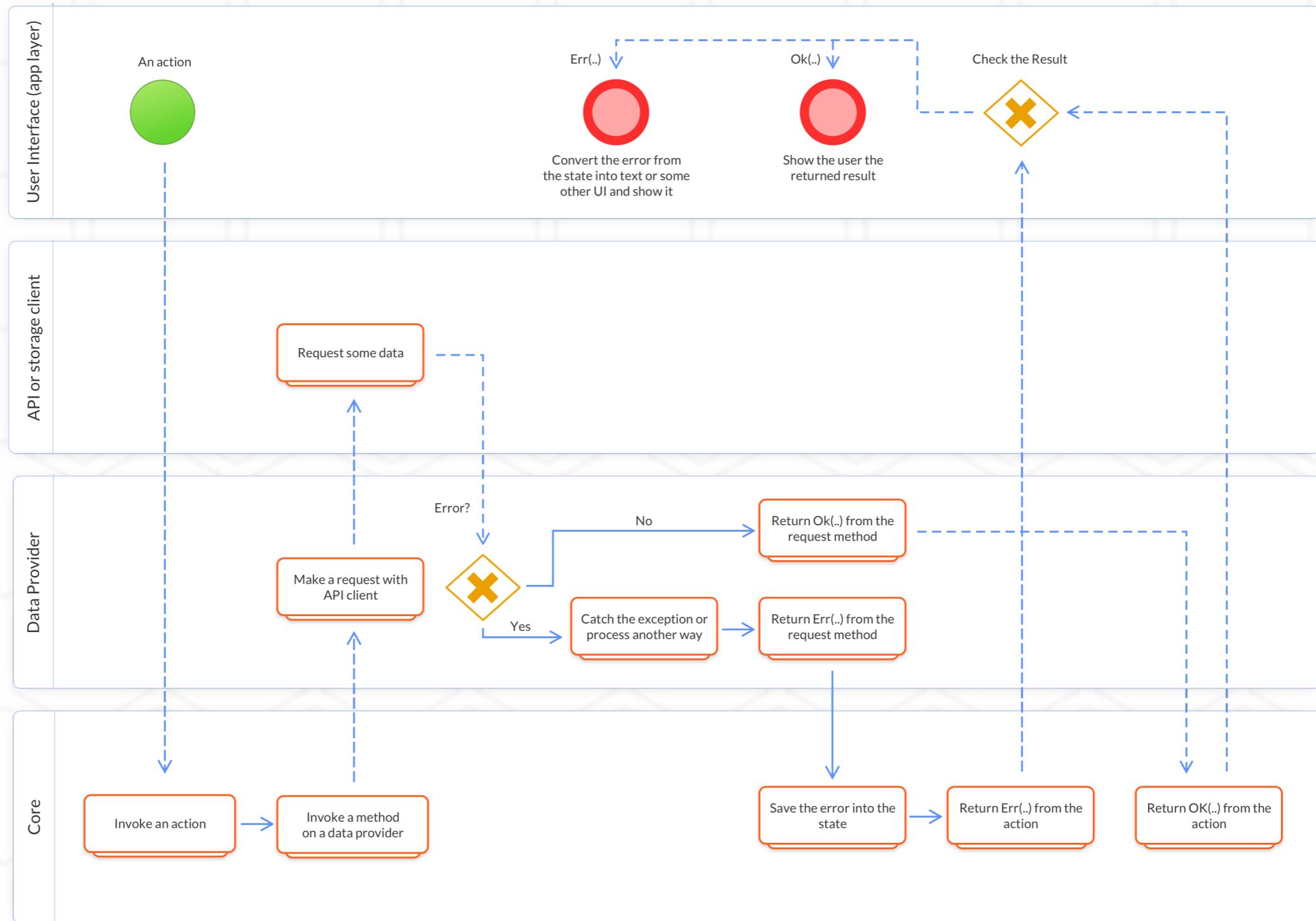


- Посредник между приложением и миром
- Реализации зависимостей для внутренних слоев
- Обертки над сторонними сервисами и библиотеками
- Хелперы для слоя приложения
- Можно выделить в отдельную библиотеку

Фреймворк



Обработка ошибок



Аналогия: MVC

- Ядро — модель
- Контейнер — контроллер
- Компонент — вид

Реализация



ядро

Ядро

Провайдеры

Приложение

Фреймворк

Ядро



- Redux
- Состояние разбито на изолированные подсостояния
- Хелперы типов для работы с Redux
- Публичные интерфейсы
- Объект Core для экспорта
- АдAPTERЫ для UI-библиотек

Объект Core



```
interface ICoreInitOptions {  
    config: IStoreConfig;  
    dataProvider: IDataProvider;  
}
```

```
export const Core = {  
    init(options: ICoreInitOptions) {},  
    get actions() {},  
    get state() {},  
    resetState() {}  
};
```

Объект Core



```
Core.actions.registration  
  .registerUser({ email, password })  
  
Core.actions.currentUser  
  .saveUserData({ name, birthdate })  
  
const { currentUser } = Core.state
```

Подсостояние



```
export interface IState {}  
export interface INamespace { currentUser: IState }  
export const initialState: IState = {};
```

Подсостояние



```
export interface IState {}  
export interface INamespace { currentUser: IState }  
export const initialState: IState = {};
```

```
export function update(state, action) {  
  switch (action.type) {  
    default:  
      return initialState;  
  }  
};
```

Подсостояние



```
export interface IState {}  
export interface INamespace { currentUser: IState }  
export const initialState: IState = {};
```

```
export function getStateWithDerivedData(  
  state: { currentUser }  
) {  
  return {  
    currentUser: {  
      ...currentUser,  
      // Вычисляемые данные  
    },  
  }  
}
```

Действия/команды



```
export const Actions = {}
```

```
export const mapDispatch = (dispatch) => ({  
  currentUser: bindActionCreators(  
    Actions,  
    dispatch,  
  ),  
});
```

Использование провайдеров данных



```
export const Actions = {  
  fetchCurrentUserData() {  
    return async (dispatch, _, { currentUser }) => {  
      //  
    };  
  };  
};
```

Использование провайдеров данных



```
export const Actions = {  
  fetchCurrentUserData() {  
    return async (dispatch, _, { currentUser }) => {  
      const result = await currentUser.fetchCurrentUser();  
  
    },  
};
```

Использование провайдеров данных

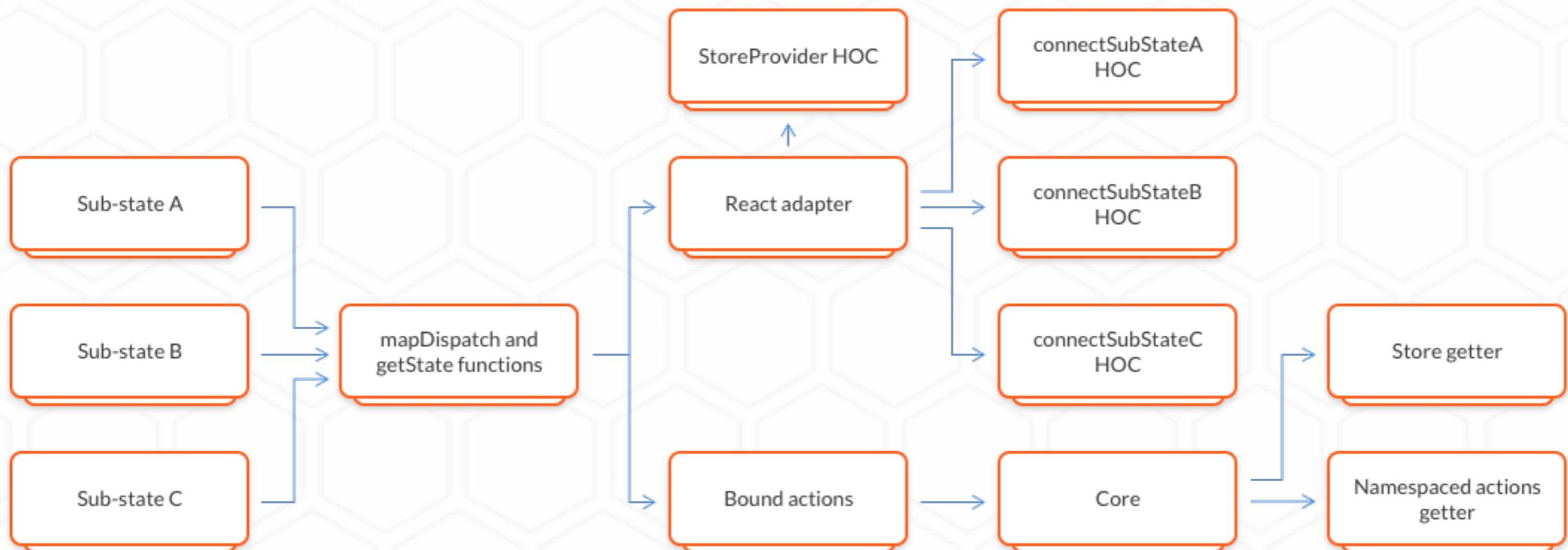


```
export const Actions = {
  fetchCurrentUserData() {
    return async (dispatch, _, { currentUser }) => {
      const result = await currentUser.fetchCurrentUser();

      if (result.is_ok()) {
        const data = result.ok();
        dispatch(new SetUserDataAction(data));
      }

      return result;
    },
  };
}
```

Адаптер для React



Адаптер для React



```
export const connectCurrentUserState = connect(  
  (state: IGlobalState) => ({  
    currentUser: currentUser  
      .getStateWithDerivedData(state)  
      .currentUser,  
  }) ,  
  
  currentUser.mapDispatch,  
);
```

Адаптер для React



```
export const connectCurrentUserState = connect(  
  (state: IGlobalState) => ({  
    currentUser: currentUser  
      .getStateWithDerivedData(state)  
      .currentUser,  
  }) ,  
  currentUser.mapDispatch,  
);
```

Провайдеры данных

Ядро

Провайдеры

Приложение

Фреймворк

Провайдеры данных



- Запросы к API и локальному хранилищу
- Перехват исключений при вызове API
- Валидация схемы данных (опционально)
- Конвертация данных (опционально)

Провайдеры данных



```
class NewsDataProvider implements INewsDataProvider {  
}  
}
```

Провайдеры данных



```
class NewsDataProvider implements INewsDataProvider {  
    public async fetchPosts() {  
        ...  
    }  
}
```

Провайдеры данных



```
class NewsDataProvider implements INewsDataProvider {  
    public constructor(client, storage) {}  
  
    public async fetchPosts() {  
        ...  
    }  
}
```

Провайдеры данных



```
class NewsDataProvider implements INewsDataProvider {  
    public constructor(client, storage) {}  
  
    public async fetchPosts() {  
        const response = await this.client  
            .query(fetchPostsQuery);  
  
    }  
}
```

Провайдеры данных



```
class NewsDataProvider implements INewsDataProvider {  
    public constructor(client, storage) {}  
  
    public async fetchPosts() {  
        const response = await this.client  
            .query(fetchPostsQuery);  
        if (result.is_ok()) {  
            return Ok(response.posts);  
        } else {  
            return Err(new NewsError("transportError"));  
        }  
    }  
}
```

Провайдеры данных



```
class NewsDataProvider implements INewsDataProvider {  
    public constructor(client, storage) {}  
  
    public async fetchPosts() {  
        const response = await this.client  
            .query(fetchPostsQuery);  
        if (result.is_ok()) {  
            this.writePostsToCache(result.ok());  
            return Ok(response.posts);  
        } else {  
            return this.fetchPostsFromCache();  
        }  
    }  
}
```

Приложение

Ядро

Провайдеры

Приложение

Фреймворк

Приложение



- Реализация для конкретной платформы
- Управление жизненным циклом приложения
- Пользовательский интерфейс
- Навигация
- Конфигурация
- Сложно перенести на другую платформу

Приложение



- React Native и его библиотеки
- React Navigation
- Библиотеки из слоя фреймворка, свои и обертки
- Графический интерфейс на React

Синглтон Application



```
export class Application {  
    public apiClient!: ApiClient;  
    public config = new Config();  
    public dispatcher = new EventDispatcher();  
    public dataProvider!: IDataProvider;  
  
    public init(): void {}  
    private subscribeToEvents(): void {}  
}
```

Создание экрана



```
class PostsScreenContent extends PureComponent {  
}  
  
export const Posts = createScreen(  
  {title: "Posts", fullscreen: true}  
);
```

Экраны и ядро



```
class PostsScreenContent extends PureComponent {  
}  
  
export const Posts = createScreen(  
  connectNewsState(PostsScreenContent),  
  screenOptions  
);
```

Экраны и ядро



```
class PostsScreenContent extends PureComponent {  
  public componentDidMount() {  
    this.props.news.fetchPosts();  
  }  
  
  public render() {  
    const { news } = this.props;  
  }  
}
```

Фреймворк

Ядро

Провайдеры

Приложение

Фреймворк

Фреймворк



- Посредник между приложением и миром
- Реализации зависимостей для внутренних слоев
- Обертки над сторонними сервисами и библиотеками
- Хелперы для слоя приложения
- Можно выделить в отдельную библиотеку

Результат



Мало багов

- Три ошибки в рантайме после релиза
- Из-за того, что неправильно описали типы
- Продуманные правила обработки ошибок
- Жесткие настройки TypeScript
- Избавились от Apollo в пользу обертки над Fetch API

Независимость от библиотек

- Любую стороннюю библиотеку можно заменить
- Меняется только реализация
- Остальной код остается нетронутым
- Главное – реализовать нужный интерфейс

Легкость поддержки

- Изменения локализованы
- Есть четкие правила, как организовывать код
- Места взаимодействия слоев ограничены
- Для тестирования бизнес-логики не нужно мокать интерфейсы и API
- Писать код легче с заранее описанными правилами

Работа в оффлайне

- Синхронизации нет
- Есть кэш и откат на него в случае ошибки
- Логика кэширования описывается в слое провайдеров данных

Переносимость кода

- Слои ядра и провайдеров данных независимы от реализации зависимостей
- Их можно вынести в отдельные пакеты и использовать для веб- или nodejs-приложения
- Придется написать слой приложения заново
- Слой фреймворка можно перенести частично
- Или добавить туда реализации для разных платформ

Производительность

- React Native на Android не любит обертки
- Меньше компонентов-оберток – быстрее рендеринг
- Ввод текста не лагает
- Кэширование ускоряет первое отображение данных

Нюансы

- Нужно правильно подобрать количество слоев
- Кто-то должен следить за глобальной картиной в первое время
- Все ли понимают какую-то задумку?
- Где чаще возникают непонятки?
- Смотрим pull-реквесты, обсуждаем, документируем

Возможно, вам это не нужно

- Если у вас классическое веб-приложение
- Если у вас простое приложение
- Если у вас лапки



Ссылки

- Alistair Cockburn. Hexagonal architecture
- Chris Fidao. Hexagonal Architecture
- Olufemi Adeojo. The curious case of reusable JavaScript state management
- Michel Weststrate. How to decouple state and UI (a.k.a. you don't need componentWillMount)
- Michel Weststrate. UI as an afterthought

Спасибо!

Александр Мадьянкин

alexander@madyankin.name

<http://t.me/madyankin>

Злые марсиане

<http://evl.ms/blog>

<http://evl.ms/telegram>

