Слои и секторы бизнес-логики. Строим масштабируемую архитектуру с **Reflexio**

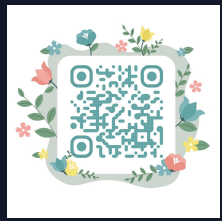
Dr. Konstantin Astapov

Rambler&Co

Reflexio

a state management framework

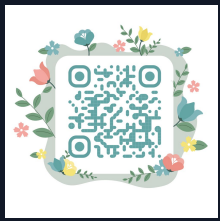
- Библиотека на основе redux
- + расширяемая экосистема плагинов
- + конвенции, примеры и паттерны



Задачи

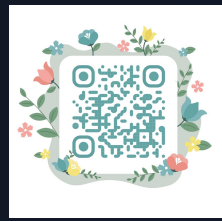
Масштабируемая архитектура

- Разделение UI и бизнес логики
- Декларативное описание БЛ
- Модульная архитектура с малой зацепленностью



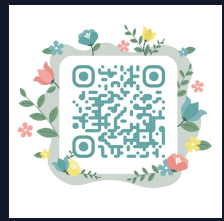
React + Redux way

```
export const Page = () => {
  const { tasks, loading } = useSelector((state: any) => ({
    tasks: state.tasks,
    loading: state.loading,
  }));

  //@ts-ignore
  useEffect(() => {
    dispatch({ type: 'LOAD_TASKS_START' });

    return () => dispatch({ type: 'CLEAR_TASKS' });
  }, []);
  useEffect(() => {
    //TO DO SMTH
  }, [tasks]);

  if (loading) {
    return <Loader />;
  }

  return <div>{tasks}</div>;
};
```



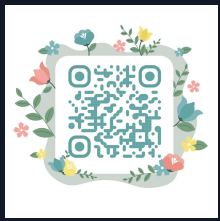
Effects chain

```
useEffect(() => () => setIsAllMessagesSelected(false), []);

useEffect(() => {
  if (isSmartCategoriesUpdateInProgress) {
    dispatch(showCategoriesCombiningNotification());
  } else {
    dispatch(removeCategoriesCombiningNotifications());
  }

  return () => dispatch(removeCategoriesCombiningNotifications());
}, [isSmartCategoriesUpdateInProgress]);

useEffect(() => {
  if (isChainsUpdateInProgress) {
    dispatch(showChainsCombiningNotification());
  } else {
    dispatch(removeChainsCombiningNotifications());
  }

  return () => dispatch(removeChainsCombiningNotifications());
}, [isChainsUpdateInProgress]);

useEffect(() => {
  if (LAYOUT.is3k && listRef.current) {
    listRef.current.scrollTop += 55 * visibleLetterIndex;
  } else {
    window.scrollBy(0, 35 * visibleLetterIndex);
  }
}, [visibleLetterIndex, activeLetter]);
```



Проблемы

- Hooks => Привязка к жизненному циклу компонентов.
- useEffect отслеживает изменение состояния, в то время как лучше отслеживать причину изменения этого состояния, то есть реагировать на конкретные события.
- Эффекты, реакция на них и последующий триггер новых эффектов - разнесены по разным слоям. В то время как удобнее это делать в одном месте в виде скрипта из последовательно выполняемых процедур.
- Оборачивание в useCallback и useMemo



Redux Toolkit

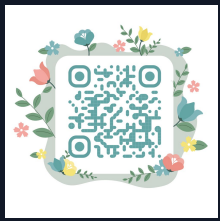
createSlice(...) =

reducerA : (state, action) => { ... },

reducerB : (state, action) => { ... },

‘sliceName’

initialState



Redux Toolkit => Reflexio

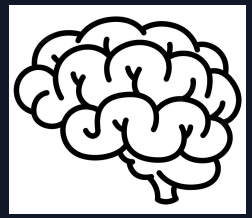
Slice(...) =

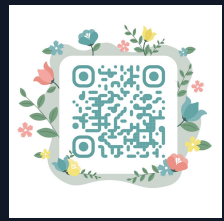
- Bite(...)
- Bite(...)
- ‘sliceName’
- initialState

(

```
{
  eventA : ( state, payload ) => { ... },
  eventB : ( state, payload ) => { ... },
    …
},
{ “script-middleware“}
```

)





Концепция reflexio

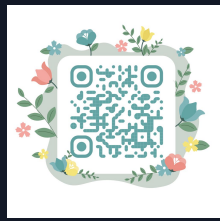
- Близкие экшены объединяются в группы (байты) - двухуровневая иерархия
- Обработка экшенов в мидлваре (одной на байт)
- Мидлвара - это инстанс класса Script
- Инстансы script создаются и умирают также по экшенам (событиям)
- Script получает через DI инструменты для работы
- Редьюсеры - это сеттеры либо они отсутствуют вовсе

Конфиг + редьюсеры

{event : reducer}

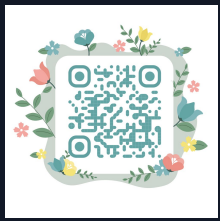
script config

```
export const notificationBite = Bite<
  IAppTriggers,
  IAppState,
  'notification',
  _ITriggers
>(
  {
    close: null,
    drop: null,
    showSmart: null,
    setState(state, payload) {
      state.notification = payload as any;
    },
    clickYes: null,
    clickYesReturnToForm: null,
    show: null,
    init(state, payload) {
      state.notification = {} as any;
    },
  },
  {
    initOn: 'init',
    instance: 'stable',
    watchScope: ['notification'],
    script: NotificationScript,
  }
);
```



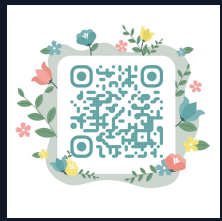
Скрипт

```ts
export class NotificationScript {
  opts: ScriptOptsType<_ITriggers, _IState, 'notification', null>;

  watch(args: WatchArgsType<_ITriggers, 'notification'>): void {
    const showEvent = this.opts.catchEvent('notification', 'show', args);
    console.log(showEvent.payload);
  }
}
```



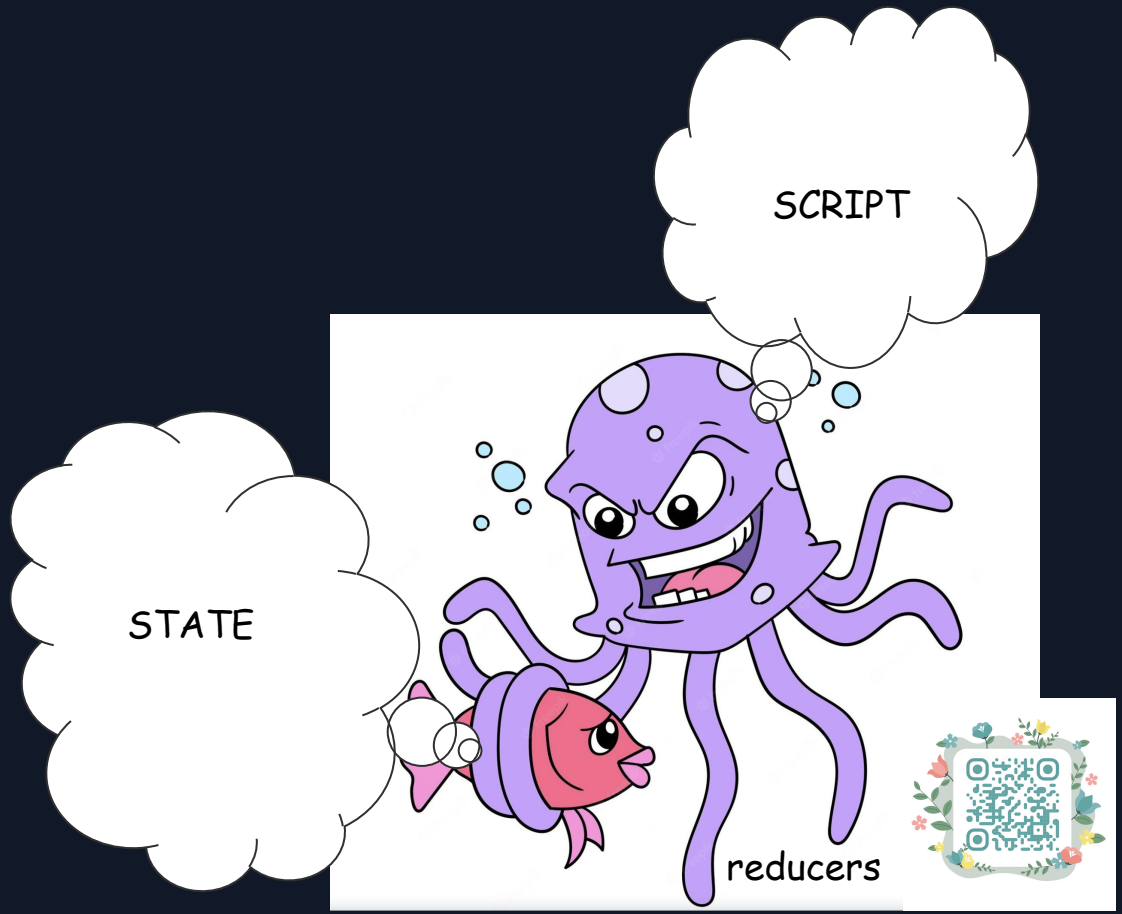
DI tools

- this.opts.catchEvent(‘biteName’, ‘eventName’)
- this.opts.trigger(‘biteName’, ‘eventName’, payload)
- const payload = wait this.opts.await(‘biteName’, ‘eventName’)



Bite

= case-reduces + script



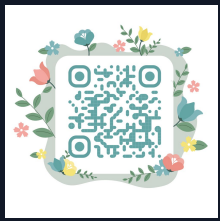
Сохранение формы

Hello1

Letter body

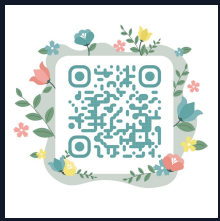
Сохранить

Закрыть



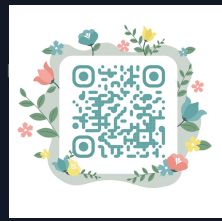
Сохранение формы

- Отправка запроса
- Закрытие формы
- Показ уведомления



Процедуры

```
public async init(
  args: InitArgsType<IComposeTriggers, 'submitLetter', 'init'>
) {
  const { openedComposeId } = this.opts.getCurrentState().compose;
  this.opts.trigger('setContent', 'commitFormContent', null); // сохранили данные из локал  в стейт
  const { subject, body } = this.opts.getCurrentState().compose;
  // запускаем скрипт сохранения
  this.opts.trigger('saveLetter', 'init', {
    body: body,
    subject: subject,
    from: 'asapovk@gmail.com',
    to: '',
    uid: 123,
  });
  const savedId = await this.opts.wait('saveLetter', 'done');
  // закрываем окно
  this.opts.trigger('setContent', 'closeWindow', {
    id: openedComposeId,
    noCheck: true,
  });
  this.opts.trigger('showNotification', 'init', 'Письмо успешно сохранено'); // кидаем
  this.opts.drop(); // убиваем инстанс
}
```



reducer: null

Slice(...) =

Bite(...)

Bite(...)

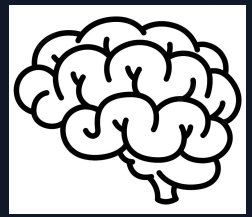
‘sliceName’

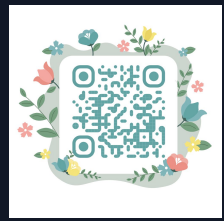
initialState

(

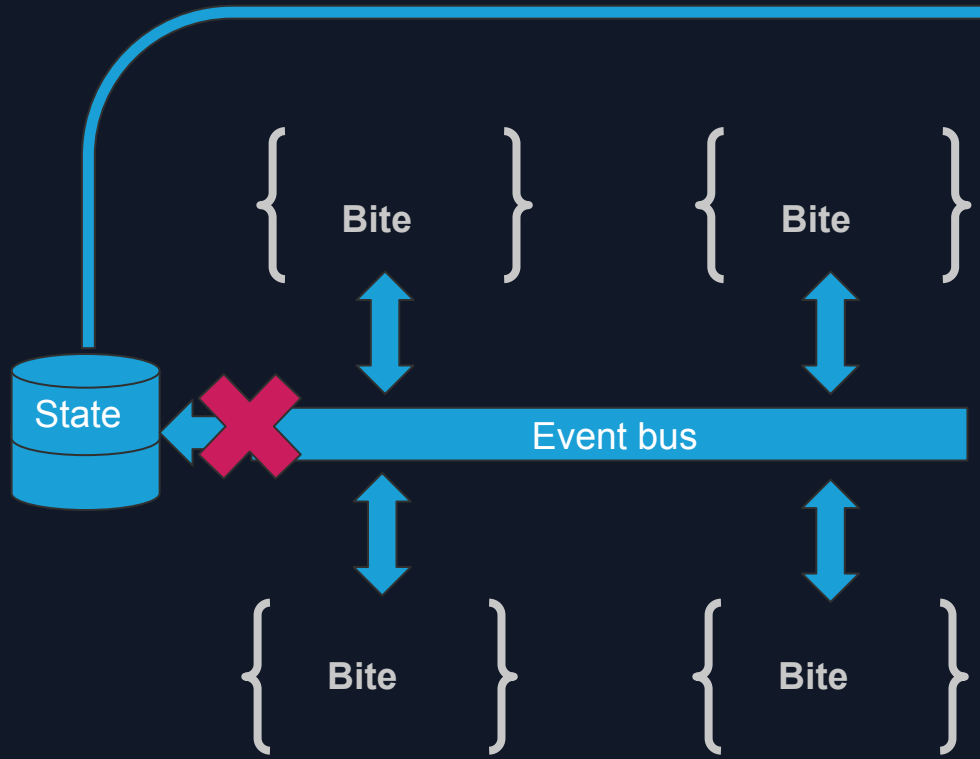
{
  eventA : null
  eventB : null
    …
},
{ “script-middleware“}

)

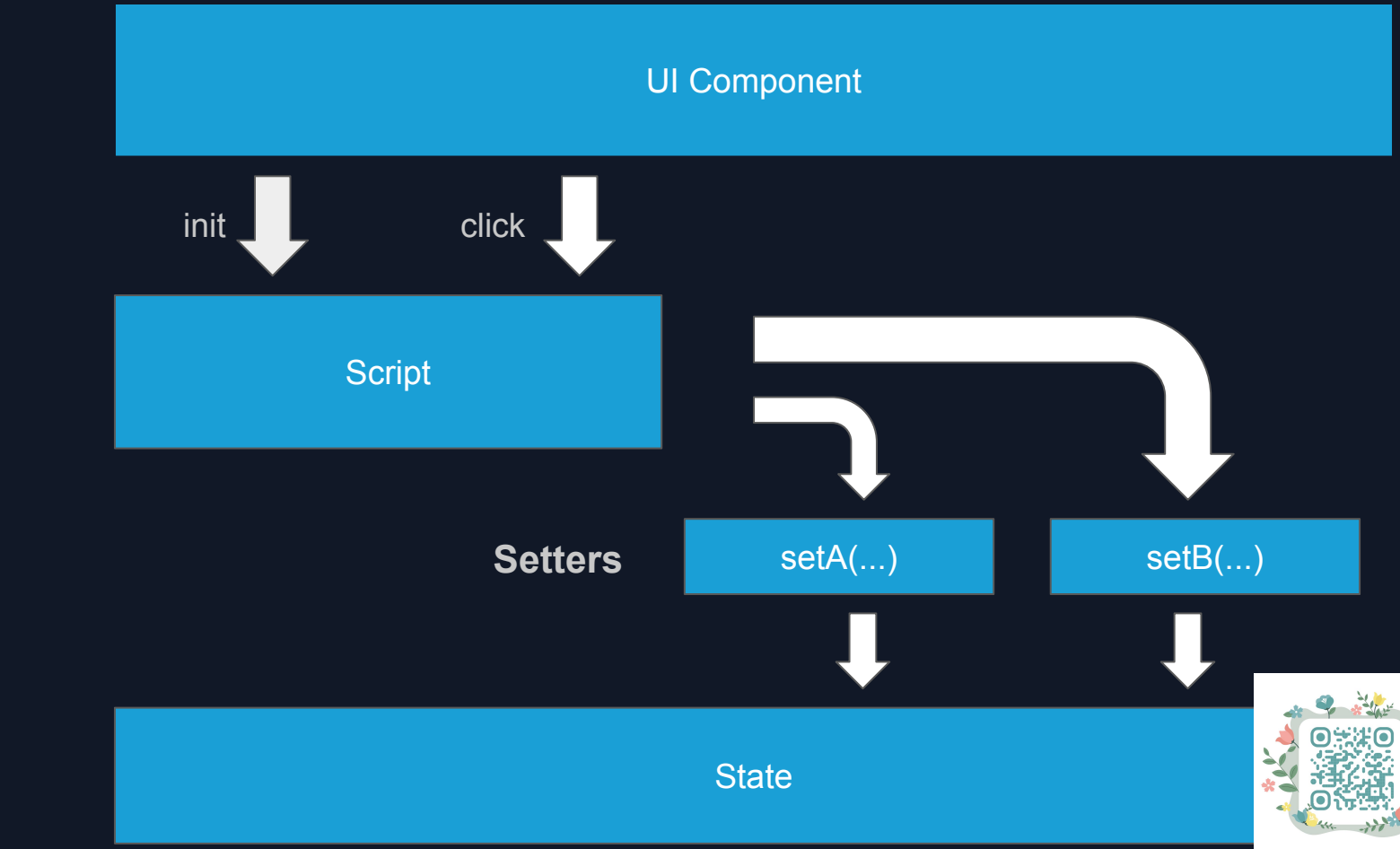




reducer: null



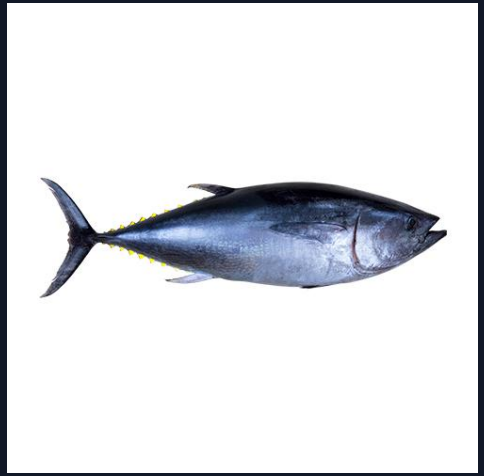
Схема



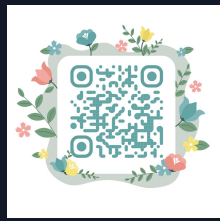
State vs Script props

```
class Script {
 data
 timeout
 flag
 DOMrefs
}
```

UI

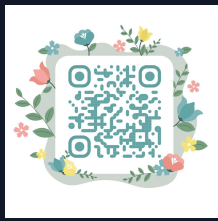


State



Benefits

- OOP => private props
- DI tools
- Safe reactive module communications
- Event bus
- Two-level events naming



Создать фрагмент (DEMO)

言っ て なく ても 似た よう な 言葉 は 誰 か が 言っ てる に 違い ない

（ サイモン・ブレジネフ ） へイ いらっしゃい

久 しぶり よう 来 たな

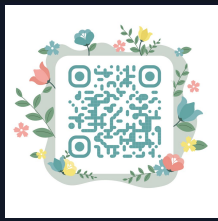
安い よ サービス する よ 門田 社長

誰 が 社長 だ 誰 が

ああ 一番 安い 握り 5 人前

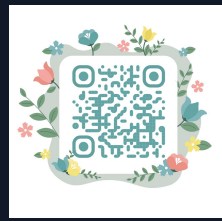
（ デニス ） あい よ

Проверить себя



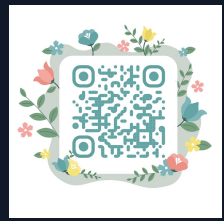
fragmentBite

```
const fragmentBite = Bite<
  IEpisodeTriggers,
  IEpisodeState,
  'fragment',
  ITriggers
>(
  {
    init(state, payload) {
      state.fragment = []
    },
    'cancelSelection': null,
    closeFragment(state,payload) {
      state.fragment = null;
    },
    'playFragment': null,
    'saveFragment': null,
    'setSelection'(state, payload) {
      state.fragment = payload;
    },
    clickRow: null,
  },
  {
    script: FragmentScript,
    initOn: 'init',
    watchScope: ['fragment'],
    instance: 'stable',
  }
);
```



Фреймворк

- Async => init, done (аналог effect)
- Event-manager => mute, unmute, forward
- Forms
- Router => goTo, goBack, setBlocker (sync bar with rfx events)
- Stager



Stage by stage (DEMO)

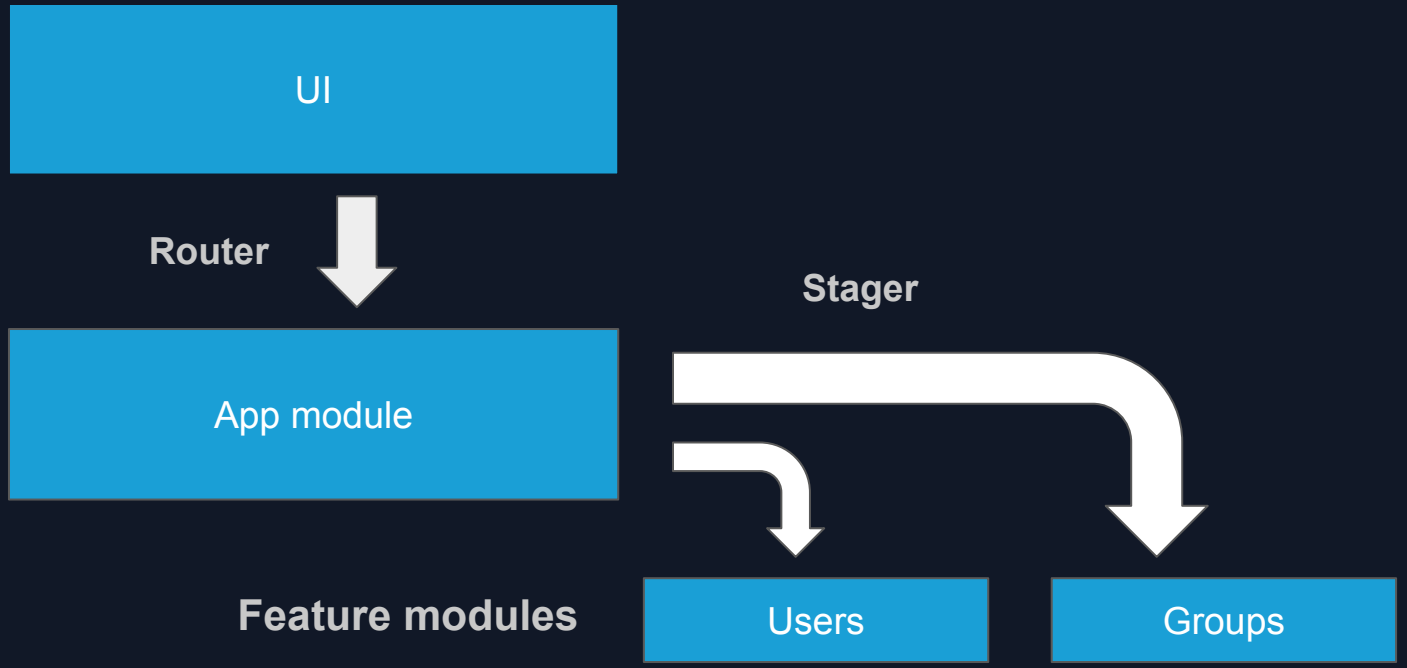
/groups

1 AUTH
2 LOAD_GROUPS
3 GROUPS_PAGE

/groups/12/edit

1 AUTH
2 LOAD_GROUPS
3 GROUPS_PAGE
4 SELECT_GROUP
5 EDIT_GROUP_DIALOG

Архитектура



Описание и примеры