SINGULARIS LAB

MobX

Шаповалов В.

MobX

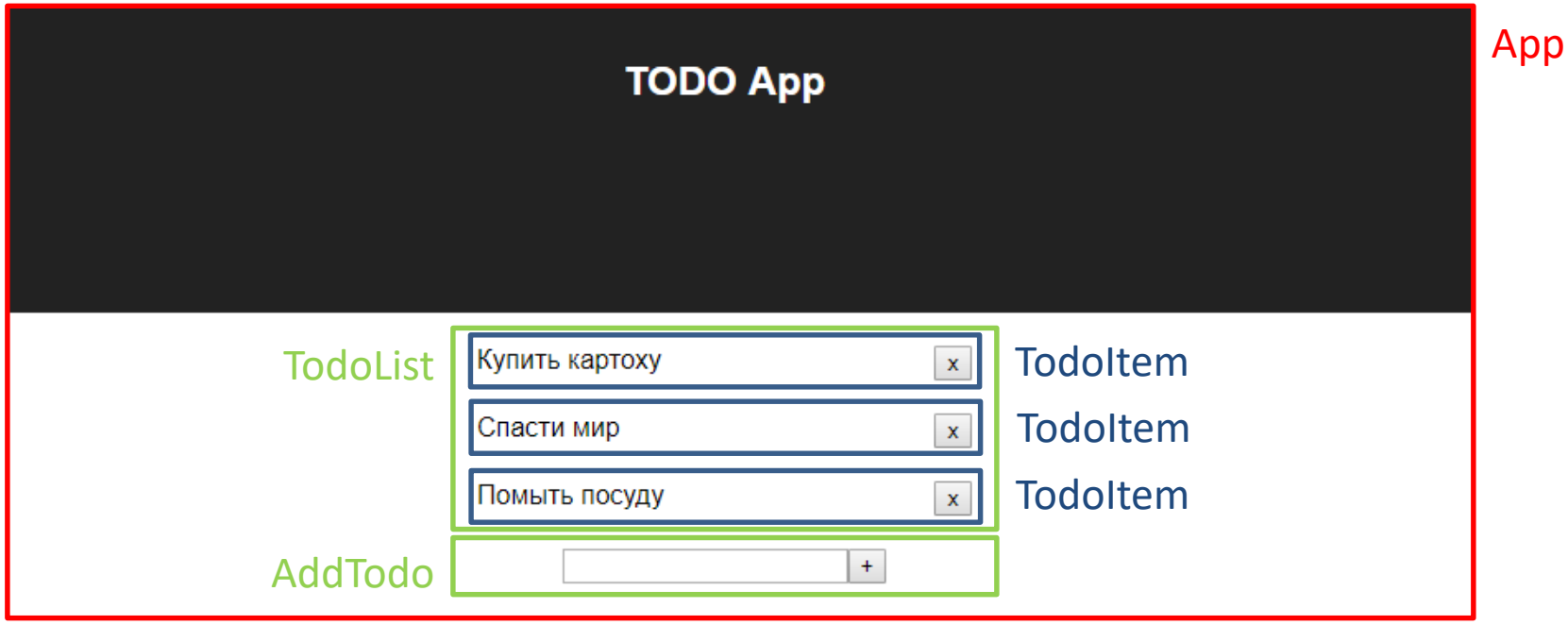
Simple, scalable state management

Давайте разберёмся

React

- State
- Props

TODO list


TODO App
App
TodoList
Купить картоху
x
TodoItem
Спасти мир
x
TodoItem
Помыть посуду
x
TodoItem
AddTodo
+

State management?

- View
- URL
- Local storage
- Cookies
- …

Состояние – это дерево

Состояние – это живое дерево

Как отслеживать изменения состояния?

Ручная подписка на изменения

1. Используем текущее состояние
2. Подписываемся на изменения
3. Реагируем на изменения

Ручная подписка на изменения

```
render(state);

state.on('change', () => {
    render(state);
});
```

Проблемы ручной подписки

- Переподписка: отслеживание изменения всего дерева избыточно.
- Недоподписка: отслеживание отдельных полей в конце концов ведёт к неконсистентности.

MobX

автоматическая

MobX – ~~автомагическая~~ подписка

Философия MobX

Anything that can be derived from the application state, should be derived. Automatically.

Основные понятия

- Observable
- Computed
- Actions
- Reactions

@observable

```
class Person {
      ...
      @observable firstName = 'Ivan';
      @observable lastName = 'Ivanov';
      @observable nickName;
      ...
}
```

@computed

```
class Person {
      ...
      @computed get fullName() {
            return this.firstName + ` ` + this.lastName;
      }
      ...
}
```

@computed

Computed зависит от исходных данных и возвращает производные данные

@action

```
class Person {
      ...
      @action setNickName(nickName) {
            this.nickName = nickName;
      }
      ...
}
```

Reactions

- Не возвращают результат
- Дают побочный эффект

Reactions

Реакции срабатывают при каждом изменении исходных данных, от которых они зависят

autorun

```
autorun(() => {
    console.log(person.nickName || person.fullName);
});
```

autorun

```
autorun(() => {
    console.log(person.nickName || person.fullName);
});
```

autorun

```
autorun(() => {
    console.log(person.nickName || person.fullName);
});
```

@computed

```
class Person {
      ...
      @computed get fullName() {
            return this.firstName + ` ` + this.lastName;
      }
      ...
}
```

Наблюдаемые поля

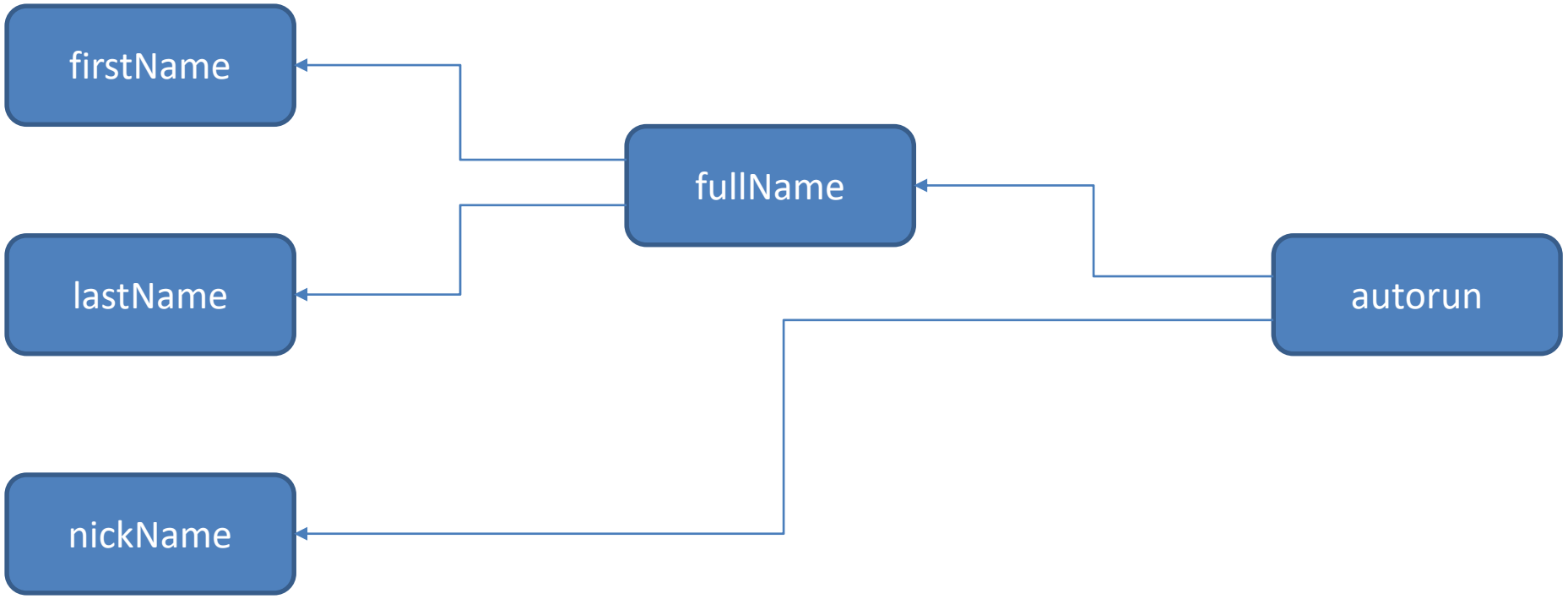
> person.nickName

> person.fullName

> person.firstName

> person.lastName

Дерево зависимостей



Уведомление об изменении данных

```
person.setNickName('Aloha');
```

@action

```
class Person {
      ...
      @action setNickName(nickName) {
            this.nickName = nickName;
      }
      ...
}
```

autorun

```
autorun(() => {
    console.log(person.nickName || person.fullName);
});
```

autorun

```
autorun(() => {
    console.log(person.nickName || person.fullName);
});
```

Наблюдаемые поля

> person.nickName

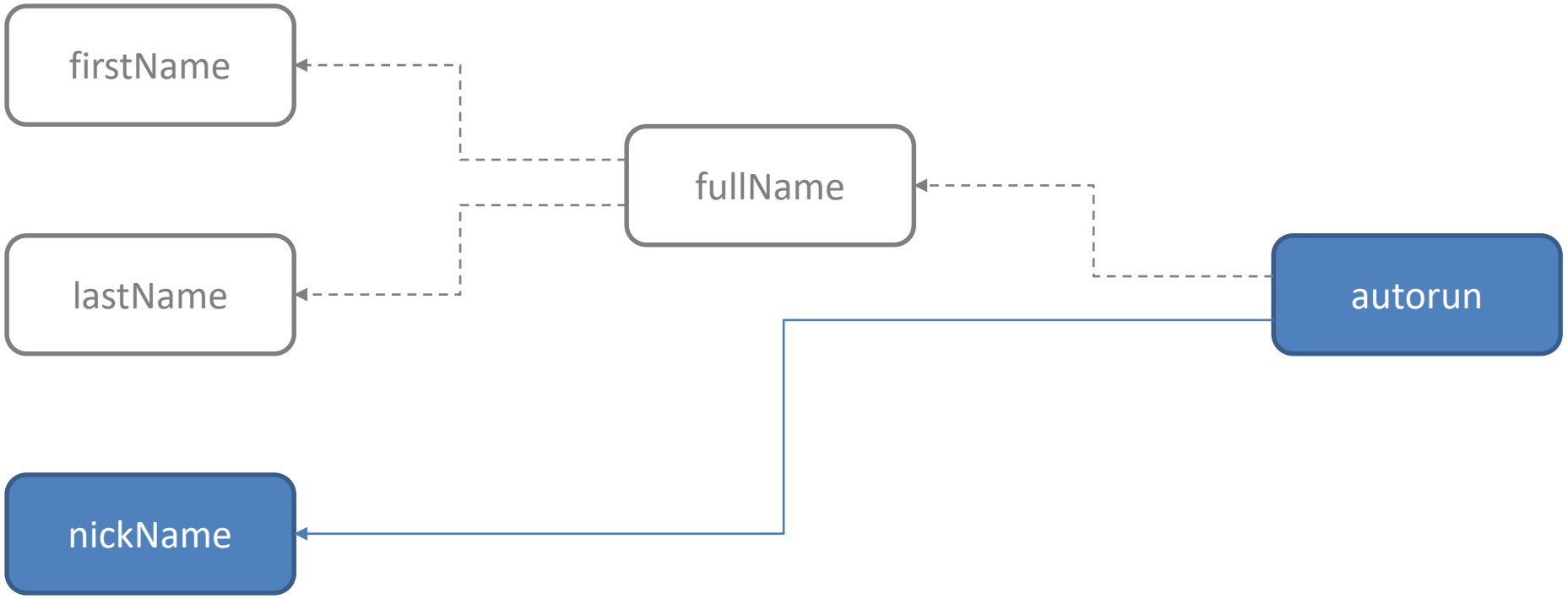
Ненаблюдаемые поля

> person.fullName

> person.firstName

> person.lastName

Дерево зависимостей



autorun

Autorun при каждом выполнении заново собирает список зависимостей

@observer

```
@observer
class ProfileView extends React.Component {
      render() {
            const person = { this.props };
            if(person.nickName) {
                  return <div>{person.nickName}</div>;
            }
            return <div>{person.fullName}</div>;
      }
}
```

@observer

```
@observer
class SomeComponent extends React.Component {
      render() {
            if(!this.props.isEnabled) {
                  return null;
            }
            ... много кода ...
      }
}
```

Динамическая подписка позволяет кардинально минимизировать количество перерисовок React-компонента

Если React экономит нам операции с DOM, то MobX экономит нам операции с виртуальным DOM.

Кэширование @computed

- Оно есть
- Кэширование происходит, пока есть хотя бы один подписчик
- Если подписчиков нет, кэш выкидывается, а @computed работает как обычный геттер

MobX vs Redux

Redux

- ООП -> функциональное программирование
- Модели -> immutable-структуры
- Методы -> экшены + редьюсеры
- Связи -> нормализация + селекторы

MobX

- Минимум boilerplate
- Автоматическая подписка в рантайме
- Производительность из коробки

Что ещё?

- DevTools
- mobx-utils
- mobx-state-tree

Бонус

- React Context Api

React Context API

- Обновлён в React 16.3
- Теперь открыт для использования (официально)

React Context API

- React.createContext
- Provider
- Consumer

React Context API

- Назначение – избавление от «props hell»
- Следует использовать, когда приложение ещё слишком мало, а прокидывать props уже надоело.

Ссылки

- https://mobx.js.org/
- MobX: управление состоянием без боли https://www.youtube.com/watch?v=yU_hJ2trMg4
- Redux против React Context API https://habr.com/ru/post/419449/
- Редакс в реальной жизни https://iamakulov.com/talks/redux-in-real-life/