Мифы и легенды о современном Angular

что не так с вашими представлениями

Даня Данилов

Кто рассказывает

Данила Данилов

ведущий разработчик · фронтенд

КОМПАНИЯ

ОИС оператор информационной системы

ПРОЕКТ

ГИС ЖКХ

ОПЫТ

7+ лет фронта

СТЕК

Angular
React · Vue · legacy

Почему Angular кажется сложным

Первое впечатление

- до экрана нужно понять много правил
- компонент как будто живет не сам по себе
- обновления происходят “где-то внутри”
- простое состояние быстро уходит в инфраструктуру

Современный вход

- компонент можно читать как отдельный блок
- локальное состояние лежит рядом с UI
- зависимости показывают, что обновится
- роутинг, формы и тесты остаются стандартным путем

> **Проблема входа:** слишком много невидимых правил до первого полезного экрана.

01

«Angular всё ещё старый и тяжёлый»

ИСТОРИЧЕСКИЙ БАГАЖ ПРОТИВ СОВРЕМЕННОГО ВХОДА

Почему миф появился

NgModules

Раньше компонент часто нужно было сначала подключить к отдельному “каталогу” приложения.

Новичок хотел сделать кнопку, а видел структуру проекта.

Zone.js

Angular сам замечал клики, таймеры и HTTP-ответы и запускал обновление экрана.

Это удобно, но со стороны выглядит как “магия”.

RxJS

Мощный способ работать с событиями во времени: поиск с паузой, отмена запроса, повтор после ошибки.

Проблема была в том, что его часто давали слишком рано.

> **Итог:** миф появился из-за входной сложности, а не из-за одной конкретной фичи.

Современный стартовый набор Angular

Компонент

без module-first

`standalone`

Состояние

локальное + производное

`signal / computed`

Платформа

Angular way

`встроенный набор`

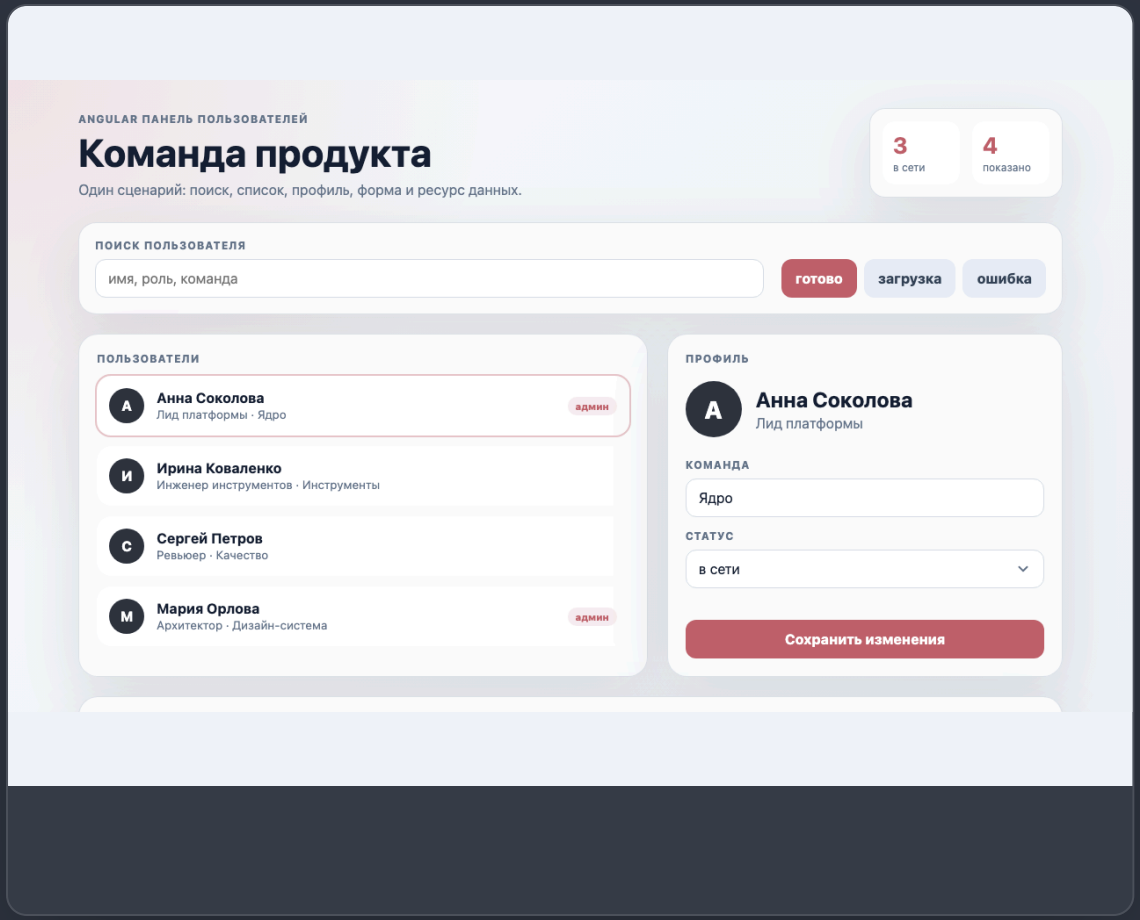
> **Цена:** принять правила платформы. **Польза:** меньше решений “с нуля” на каждую фичу.

02

«Компоненты Angular сильно сложнее»

ОДИН ПОЛЬЗОВАТЕЛЬСКИЙ СЦЕНАРИЙ, ТРИ РЕАЛИЗАЦИИ

Одна UI-задача, три реализации



Сценарий

поиск, список, детали, профиль, загрузка/ошибка

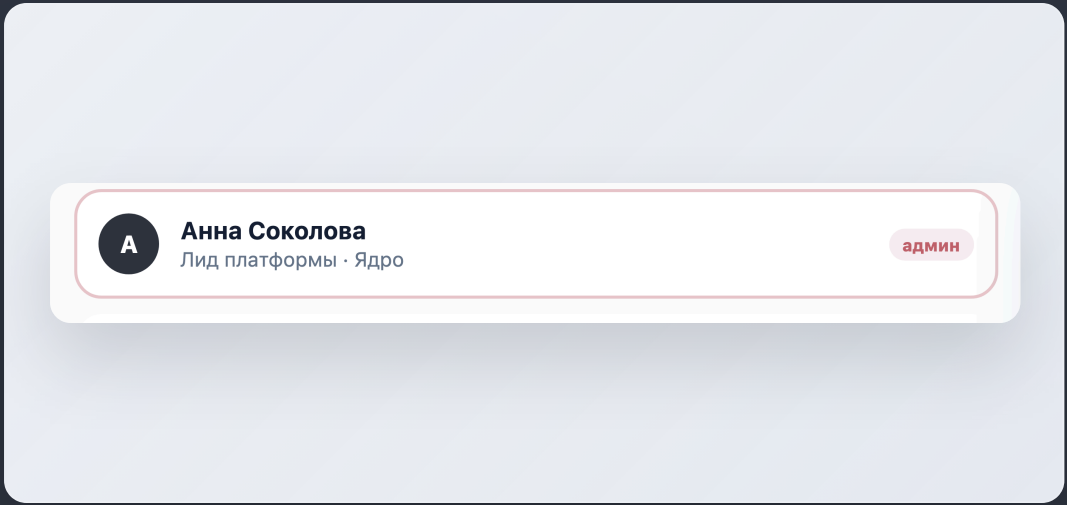
Сравнение

не “кто красивее”, а где живут состояние, контракт и соглашения

Как читать

сначала UI, потом код; подсветка показывает строки, о которых говорим

Сначала UI: что именно сравниваем



Один маленький блок: данные → состояние → событие.

1 **Данные**

имя, роль, команда, метка администратора

2 **Состояние строки**

выбранная строка получает отдельное визуальное состояние

3 **Событие выбора**

компонент сообщает наверх, родительский экран меняет выбранного пользователя

Контракт строки: React и Vue

React apps/react-users/src/app/user-row.tsx

```tsx
type UserRowProps = {
  user: User;
  selected: boolean;
  onSelect: (user: User) => void;
};

export function UserRow(
  { user, selected, onSelect }: UserRowProps,
) {
  const className = selected
    ? 'user-row selected'
    : 'user-row';

  return (
    <button
      className={className}
      onClick={() => onSelect(user)}
    >
      <span>{user.name}</span>
      <small>{user.role} · {user.team}</small>
    </button>
  );
}
```

Vue apps/vue-users/src/app/UserRow.vue

```vue
<script setup lang="ts">
defineProps<{
  user: User;
  selected: boolean;
}>();

const emit = defineEmits<{
  select: [user: User];
}>();
</script>

<template>
  <button
    :class="['user-row', { selected }]"
    @click="emit('select', user)"
  >
    <span>{{ user.name }}</span>
    <small>{{ user.role }} · {{ user.team }}</small>
  </button>
</template>
```

Одна модель: данные вниз, событие вверх.

Angular: тот же контракт компонента

Angular apps/angular-users/src/app/user-row.ts

```ts
@Component({
  selector: 'app-user-row',
  template: `
    <button class="user-row">
      <span></span>
      <small></small>
    </button>
  `,
})
export class UserRow {
}
```

Angular: тот же контракт компонента

Angular apps/angular-users/src/app/user-row.ts

```ts
@Component({
  selector: 'app-user-row',
  template: `
    <button class="user-row">
      <span>{{ user().name }}</span>
      <small>{{ user().role }} · {{ user().team }}</small>
    </button>
  `,
})
export class UserRow {
  user = input.required<User>();
}
```

1. ДАННЫЕ

`user = input.required<User>()`

строка не имеет смысла без пользователя, поэтому вход обязательный.

Angular: тот же контракт компонента

Angular apps/angular-users/src/app/user-row.ts

```ts
@Component({
  selector: 'app-user-row',
  template: `
    <button
      class="user-row"
      [class.selected]="selected()"
    >
      <span>{{ user().name }}</span>
      <small>{{ user().role }} · {{ user().team }}</small>
    </button>
  `,
})
export class UserRow {
  user = input.required<User>();
  selected = input(false);
}
```

1. ДАННЫЕ

`user = input.required<User>()`

строка не имеет смысла без пользователя, поэтому вход обязательный.

2. СОСТОЯНИЕ СВЕРХУ

`selected = input(false)`

состояние строки приходит сверху; сам компонент не решает, кто выбран.

Angular: тот же контракт компонента

Angular apps/angular-users/src/app/user-row.ts

```ts
@Component({
  selector: 'app-user-row',
  template: `
    <button
      class="user-row"
      [class.selected]="selected()"
      (click)="select.emit(user())"
    >
      <span>{{ user().name }}</span>
      <small>{{ user().role }} · {{ user().team }}</small>
    </button>
  `,
})
export class UserRow {
  user = input.required<User>();
  selected = input(false);
  select = output<User>();
}
```

1. ДАННЫЕ

`user = input.required<User>()`

строка не имеет смысла без пользователя, поэтому вход обязательный.

2. СОСТОЯНИЕ СВЕРХУ

`selected = input(false)`

состояние строки приходит сверху; сам компонент не решает, кто выбран.

3. СОБЫТИЕ НАРУЖУ

`select = output<User>()`

строка сообщает о клике, а родительский экран уже вызывает `selected.set($event)`.

Angular: тот же контракт компонента

Angular — apps/angular-users/src/app/user-row.ts

```ts
@Component({
  selector: 'app-user-row',
  template: `
    <button
      class="user-row"
      [class.selected]="selected()"
      (click)="select.emit(user())"
    >
      <span>{{ user().name }}</span>
      <small>{{ user().role }} · {{ user().team }}</small>
    </button>
  `,
})
export class UserRow {
  user = input.required<User>();
  selected = input(false);
  select = output<User>();
}
```

Parent template — users-page.html

```html
<app-user-row
  [user]="user"
  [selected]="selectedUser() === user"
  (select)="selectedUser.set($event)"
/>
```

1. ДАННЫЕ

`user = input.required<User>()`

строка не имеет смысла без пользователя, поэтому вход обязательный.

2. СОСТОЯНИЕ СВЕРХУ

`selected = input(false)`

состояние строки приходит сверху; сам компонент не решает, кто выбран.

3. СОБЫТИЕ НАРУЖУ

`select = output<User>()`

строка сообщает о клике, а родительский экран уже вызывает `selected.set($event)`.

4. РОДИТЕЛЬ

`[] вниз, () вверх`

отдельный блок показывает, как экран подключает компонент и связывает контракт.

Angular: тот же контракт компонента

Angular — apps/angular-users/src/app/user-row.ts

```ts
@Component({
  selector: 'app-user-row',
  template: `
    <button
      class="user-row"
      [class.selected]="selected()"
      (click)="select.emit(user())"
    >
      <span>{{ user().name }}</span>
      <small>{{ user().role }} · {{ user().team }}</small>
    </button>
  `,
})
export class UserRow {
  user = input.required<User>();
  selected = input(false);
  select = output<User>();
}
```

Parent template — users-page.html

```html
<app-user-row
  [user]="user"
  [selected]="selectedUser() === user"
  (select)="selectedUser.set($event)"
/>
```

1. ДАННЫЕ

`user = input.required<User>()`

строка не имеет смысла без пользователя, поэтому вход обязательный.

2. СОСТОЯНИЕ СВЕРХУ

`selected = input(false)`

состояние строки приходит сверху; сам компонент не решает, кто выбран.

3. СОБЫТИЕ НАРУЖУ

`select = output<User>()`

строка сообщает о клике, а родительский экран уже вызывает `selected.set($event)`.

4. РОДИТЕЛЬ

`[] вниз, () вверх`

отдельный блок показывает, как экран подключает компонент и связывает контракт.

5. ИТОГ

Та же модель компонента

Angular не меняет задачу: данные вниз, событие вверх. Он просто явно называет эти части.

Что реально отличается

данные → UserRow → событие

Одинаковая задача: получить пользователя, показать выбранное состояние и сообщить о клике наверх.

REACT

контракт собирается из props

гибко, но форму callback и соглашения вокруг компонента выбирает команда.

VUE

props и emit видны отдельно

шаблон отделен от script, а событие явно названо через emit.

ANGULAR

input/output — публичный API

контракт сразу лежит в классе, рядом с template и metadata компонента.

ЦЕНА

больше структуры видно сразу

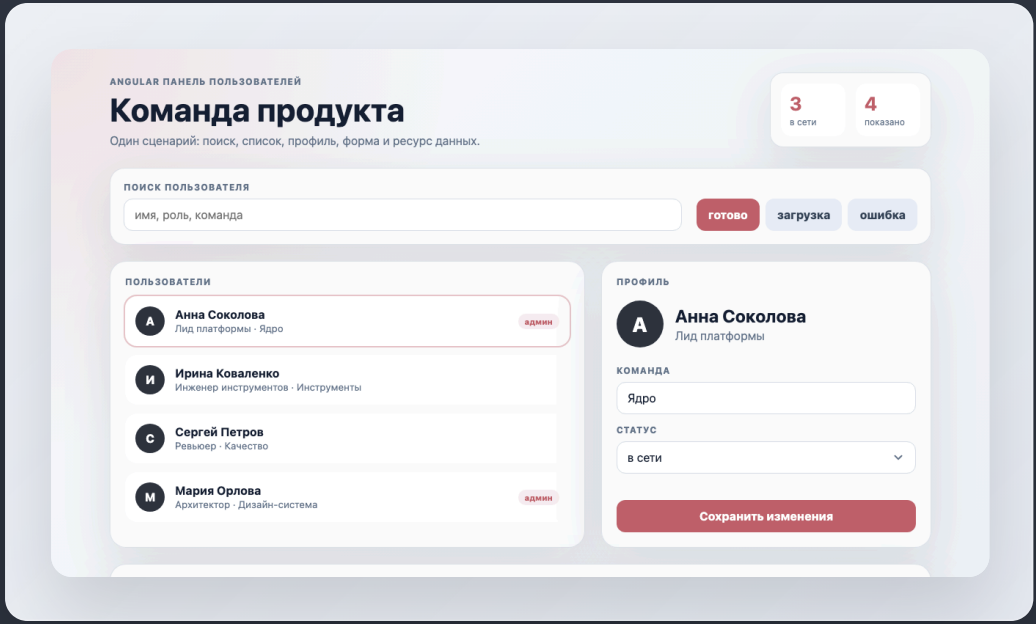
template syntax, selector, metadata, input/output появляются уже в маленьком компоненте.

ПОЛЬЗА

меньше неявных договоренностей

в команде быстрее видно, где входы, где события и как компонент подключает родитель.

Список: где появляется сложность



Список — это не только цикл по массиву. В реальном экране рядом появляются состояния, ключи строк и событие выбора.

1 **Состояния**

loading, error, empty и найденные строки должны читаться как один сценарий

2 **Идентичность**

каждая строка должна иметь стабильный ключ, иначе список тяжело обновлять правильно

3 **Выбор строки**

строка сообщает наверх, а родитель хранит выбранного пользователя

Панель пользователей: список — React / Vue

React apps/react-users/src/app/app.tsx

```tsx
function UsersList({ mode, filteredUsers, selected, onSelect }) {
  if (mode === 'загрузка') {
    return (
      <div className="state-card loading">
        Загружаем...
      </div>
    );
  }
  if (mode === 'ошибка') {
    return (
      <div className="state-card error">
        API недоступен.
      </div>
    );
  }
  if (!filteredUsers.length) {
    return <div className="state-card">Ничего не найдено.</div>;
  }
  return filteredUsers.map((user) => (
    <UserRow
      key={user.id}
      user={user}
      selected={selected.id === user.id}
      onSelect={onSelect}
    />
  ));
}
```

Vue apps/vue-users/src/app/App.vue

```vue
<template>
  <div v-if="mode === 'загрузка'" class="state-card loading">
    Загружаем пользователей...
  </div>
  <div v-else-if="mode === 'ошибка'" class="state-card error">
    API недоступен. Повторите запрос.
  </div>
  <template v-else-if="filteredUsers.length">
    <UserRow
      v-for="user in filteredUsers"
      :key="user.id"
      :user="user"
      :selected="selected.id === user.id"
      @select="selected = $event"
    />
  </template>
  <div v-else class="state-card">Ничего не найдено.</div>
</template>
```

Одинаковый сценарий: состояния списка, ключ строки, событие выбора.

Панель пользователей: Angular список

Angular apps/angular-users/src/app/app.html

```
<aside class="users-list">
  <div class="panel-title">Пользователи</div>
</aside>
```

Панель пользователей: Angular список

Angular apps/angular-users/src/app/app.html

```
@if (mode() === 'загрузка') {
  <div class="state-card loading">
    Загружаем пользователей...
  </div>
} @else if (mode() === 'ошибка') {
  <div class="state-card error">
    API недоступен. Повторите запрос.
  </div>
} @else {
  <aside class="users-list">
    <div class="panel-title">Пользователи</div>
  </aside>
}
```

ВЕТКИ СОСТОЯНИЯ

`@if / @else if / @else`

loading, error и обычный список лежат в одном template, рядом с визуальной разметкой.

Панель пользователей: Angular список

Angular apps/angular-users/src/app/app.html

```
@if (mode() === 'загрузка') {
  <div class="state-card loading">
    Загружаем пользователей...
  </div>
} @else if (mode() === 'ошибка') {
  <div class="state-card error">
    API недоступен. Повторите запрос.
  </div>
} @else {
  @for (user of filteredUsers(); track user.id) {
    <app-user-row
      [user]="user"
    />
  }
}
```

ВЕТКИ СОСТОЯНИЯ

@if / @else if / @else

loading, error и обычный список лежат в одном template, рядом с визуальной разметкой.

ИДЕНТИЧНОСТЬ СПИСКА

@for (...; track user.id)

`track` нельзя забыть “где-то в голове”: он стоит в самой строке цикла.

Панель пользователей: Angular список

Angular apps/angular-users/src/app/app.html

```
@if (mode() === 'загрузка') {
  <div class="state-card loading">
    Загружаем пользователей...
  </div>
} @else if (mode() === 'ошибка') {
  <div class="state-card error">
    API недоступен. Повторите запрос.
  </div>
} @else {
  @for (user of filteredUsers(); track user.id) {
    <app-user-row
      [user]="user"
      [selected]="selected().id === user.id"
      (select)="selected.set($event)"
    />
  }
}
```

ВЕТКИ СОСТОЯНИЯ

@if / @else if / @else

loading, error и обычный список лежат в одном template, рядом с визуальной разметкой.

ИДЕНТИЧНОСТЬ СПИСКА

@for (...; track user.id)

`track` нельзя забыть “где-то в голове”: он стоит в самой строке цикла.

ВЫБОР СТРОКИ

[] вниз, () вверх

строка получает user и selected, а select меняет выбранного пользователя у родителя.

Панель пользователей: Angular список

Angular apps/angular-users/src/app/app.html

```
@if (mode() === 'загрузка') {
  <div class="state-card loading">
    Загружаем пользователей...
  </div>
} @else if (mode() === 'ошибка') {
  <div class="state-card error">
    API недоступен. Повторите запрос.
  </div>
} @else {
  @for (user of filteredUsers(); track user.id) {
    <app-user-row
      [user]="user"
      [selected]="selected().id === user.id"
      (select)="selected.set($event)"
    />
  } @empty {
    <div class="state-card">Ничего не найдено.</div>
  }
}
```

ВЕТКИ СОСТОЯНИЯ

@if / @else if / @else

loading, error и обычный список лежат в одном template, рядом с визуальной разметкой.

ИДЕНТИЧНОСТЬ СПИСКА

@for (...; track user.id)

`track` нельзя забыть “где-то в голове”: он стоит в самой строке цикла.

ВЫБОР СТРОКИ

[] вниз, () вверх

строка получает user и selected, а select меняет выбранного пользователя у родителя.

ПУСТОЕ СОСТОЯНИЕ

@empty

пустой результат является частью цикла, а не отдельной веткой после него.

Что реально отличается в списке

1
состояния
loading / error / empty / list

2
цикл
как создаются строки

3
ключ
как строка остается той же

4
выбор
событие уходит наверх

REACT

контроль потока в JS

условия, ранние return и map остаются обычным кодом внутри компонента.

VUE

контроль потока в template

v-if, v-for и @select читаются как декларативная разметка состояния.

ANGULAR

control flow как часть template

@if, @for, track и @empty явно показывают структуру списка.

ЦЕНА

больше framework syntax

нужно понимать @if, @for, track и @empty, даже если задача кажется обычным списком.

ПОЛЬЗА

меньше скрытых решений

ветки состояния, идентичность строки и пустой результат находятся рядом с UI.

03

«В Angular всё через RxJS»

SIGNALS

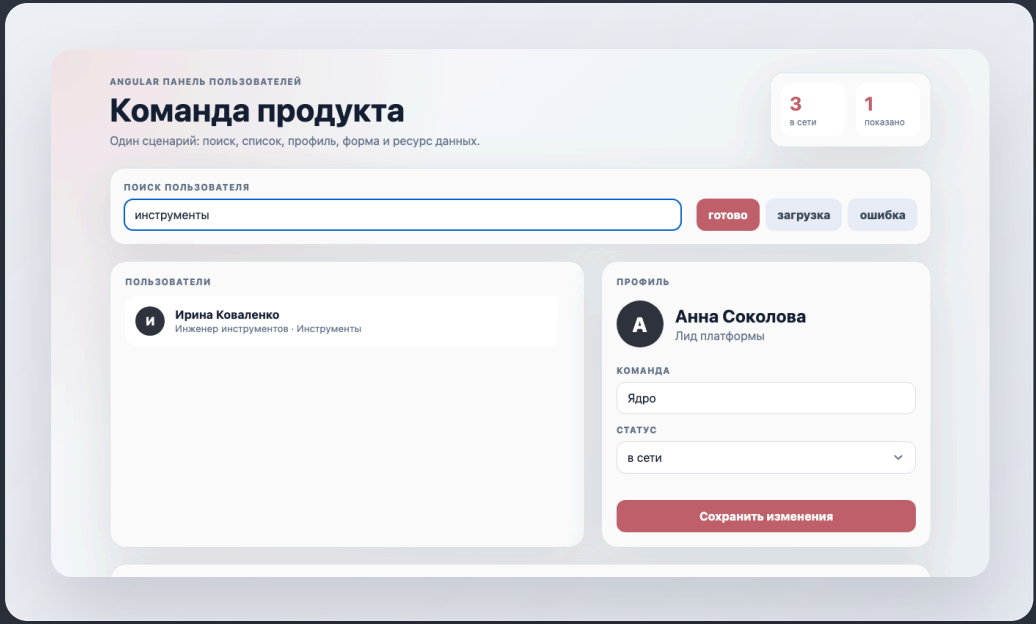
локальное состояние

RXJS

потоки событий

НЕ ЗАМЕНА, А ГРАНИЦА ОТВЕТСТВЕННОСТИ

Панель пользователей: поиск как состояние



Пользователь вводит текст, список и счетчик пересчитываются от одного источника: `query`.

1. **Источник**

   строка поиска хранится как локальное UI-состояние, без Observable и без ручной подписки

2. **Производное**

   filtered list зависит от `query` и общего массива пользователей

3. **Экран**

   список, счетчик и empty state читают уже готовое производное значение

Signals: реактивная модель простыми словами

Идея знакома по Solid и Vue: зависимость появляется в момент чтения.

1

signal хранит значение

например, текст поиска

`query`

→

2

computed читает signal

Angular запоминает зависимость

`filteredUsers`

→

3

template читает результат

экран знает, что нужно обновить

`table + count`

`query.set(...)` → `filteredUsers устаревает` → `template перечитывает результат`

> **Идея:** мы не подписываемся вручную. Мы читаем данные, и Angular строит граф зависимостей.

Панель пользователей: поиск — React / Vue

React apps/react-users/src/app/app.tsx

```tsx
const [query, setQuery] = useState('');

const filteredUsers = useMemo(() => {
  const search = query
    .trim()
    .toLowerCase();
  return allUsers.filter((user) =>
    matchesUser(user, search),
  );
}, [query]);

<input
  value={query}
  onChange={(event) =>
    setQuery(event.target.value)
  }
  placeholder="имя, роль, команда"
/>

<b>{filteredUsers.length}</b>
```

Vue apps/vue-users/src/app/App.vue

```vue
const query = ref('');

const filteredUsers = computed(() => {
  const search = query.value
    .trim()
    .toLowerCase();
  return allUsers.filter((user) =>
    matchesUser(user, search),
  );
});

<input
  v-model="query"
  placeholder="имя, роль, команда"
/>

<b>{{ filteredUsers.length }}</b>
```

Одна задача: строка поиска меняется, производный список пересчитывается, UI читает результат.

Панель пользователей: Angular поиск

Angular apps/angular-users/src/app/app.ts

```ts
export class App {
}
```

Панель пользователей: Angular поиск

Angular apps/angular-users/src/app/app.ts

```ts
import { signal } from '@angular/core';

export class App {
  protected readonly query = signal('');
}
```

1. ИСТОЧНИК

`query = signal('')`

значение поиска хранится как локальный signal, без Observable и подписки.

Панель пользователей: Angular поиск

Angular apps/angular-users/src/app/app.ts

```ts
import { computed, signal } from '@angular/core';
import { allUsers, matchesUser } from './users';

export class App {
  protected readonly query = signal('');

  protected readonly filteredUsers = computed(() => {
    const search = this.query()
      .trim()
      .toLowerCase();
    return allUsers.filter((user) =>
      matchesUser(user, search),
    );
  });
}
```

1. ИСТОЧНИК

`query = signal('')`

значение поиска хранится как локальный signal, без Observable и подписки.

2. ПРОИЗВОДНОЕ

`filteredUsers = computed(...)`

computed читает `query()`, и Angular запоминает эту зависимость.

Панель пользователей: Angular поиск

Angular apps/angular-users/src/app/app.ts

```ts
import { computed, signal } from '@angular/core';
import { allUsers, matchesUser } from './users';

export class App {
  protected readonly query = signal('');

  protected readonly filteredUsers = computed(() => {
    const search = this.query()
      .trim()
      .toLowerCase();
    return allUsers.filter((user) =>
      matchesUser(user, search),
    );
  });

  protected updateQuery(event: Event) {
    const input = event.target as HTMLInputElement;
    this.query.set(input.value);
  }
}
```

1. ИСТОЧНИК

`query = signal('')`

значение поиска хранится как локальный signal, без Observable и подписки.

2. ПРОИЗВОДНОЕ

`filteredUsers = computed(...)`

computed читает `query()`, и Angular запоминает эту зависимость.

3. ЗАПИСЬ

`query.set(input.value)`

input event меняет только источник; производное значение пересчитается само.

Панель пользователей: Angular поиск

Angular — apps/angular-users/src/app/app.ts

```ts
import { computed, signal } from '@angular/core';
import { allUsers, matchesUser } from './users';

export class App {
  protected readonly query = signal('');

  protected readonly filteredUsers = computed(() => {
    const search = this.query()
      .trim()
      .toLowerCase();
    return allUsers.filter((user) =>
      matchesUser(user, search),
    );
  });

  protected updateQuery(event: Event) {
    const input = event.target as HTMLInputElement;
    this.query.set(input.value);
  }
}
```

Template — apps/angular-users/src/app/app.html

```html
<input
  [value]="query()"
  (input)="updateQuery($event)"
  placeholder="имя, роль, команда"
/>

<b>{{ filteredUsers().length }}</b>
```

1. ИСТОЧНИК

`query = signal('')`

значение поиска хранится как локальный signal, без Observable и подписки.

2. ПРОИЗВОДНОЕ

`filteredUsers = computed(...)`

computed читает `query()`, и Angular запоминает эту зависимость.

3. ЗАПИСЬ

`query.set(input.value)`

input event меняет только источник; производное значение пересчитается само.

4. TEMPLATE

`[value] + (input) + filteredUsers()`

template читает signals как функции и показывает готовый результат.

Что реально отличается в поиске

`query` → `filteredUsers` → `UI`

Одна задача: сохранить текст поиска, получить отфильтрованный список и показать счетчик.

REACT

dependency нужно назвать

`useMemo` пересчитывается по массиву зависимостей: это явно, но можно забыть связь.

VUE

computed видит чтения

`computed` читает `query.value`, и Vue сам связывает данные с результатом.

ANGULAR

computed видит signal

`computed` читает `query()`, а template перечитывает готовый результат.

ЦЕНА

нужно привыкнуть к чтению signal

`query()`, `filteredUsers()` и `set` сначала выглядят как новый язык состояния.

ПОЛЬЗА

локальный state без потока

для синхронного UI-фильтра хватает signals; RxJS не становится обязательной стартовой точкой.

Где RxJS все еще честно нужен

Signals

- локальное UI-состояние
- производные значения
- чтения в template
- view-model

RxJS

- debounce/cancel/retry
- потоки событий
- websocket
- async-композиция

> **Реальность:** RxJS не исчез, но старт стал проще.

Signal API: быстрый переводчик

Angular расширил signals от локального state до component boundary и async data. Это карта ролей, не обещание 1:1 API.

ANGULAR	РОЛЬ	VUE	SOLID
`signal()`	записываемое состояние	`ref()` / `reactive()`	`createSignal()`
`computed()`	производное значение	`computed()`	`createMemo()`
`linkedSignal()`	зависимое состояние с записью	`writable computed` / `watch`	`signal + memo`
`effect()`	реакция на изменения	`watchEffect()` / `watch()`	`createEffect()`
`afterRenderEffect()`	DOM / сторонние API после render	`watch + nextTick`	`createRenderEffect()`
`input()`	prop как read-only signal	`props / toRef()`	`props accessor`
`model()`	двусторонний input	`defineModel()`	`value + setter props`
`output()`	событие компонента	`defineEmits()`	`callback prop`
`resource()` / `httpResource()`	async state + status	`composable + state`	`createResource()`
`viewChild()` / `contentChild()`	query как signal	`useTemplateRef()`	`ref attribute`

Обновляем ментальную модель

Компонент

Та же UI-идея, но строже контракт.

Состояние

Signals закрывают локальное UI-состояние.

RxJS

Все еще полезен для настоящих потоков.

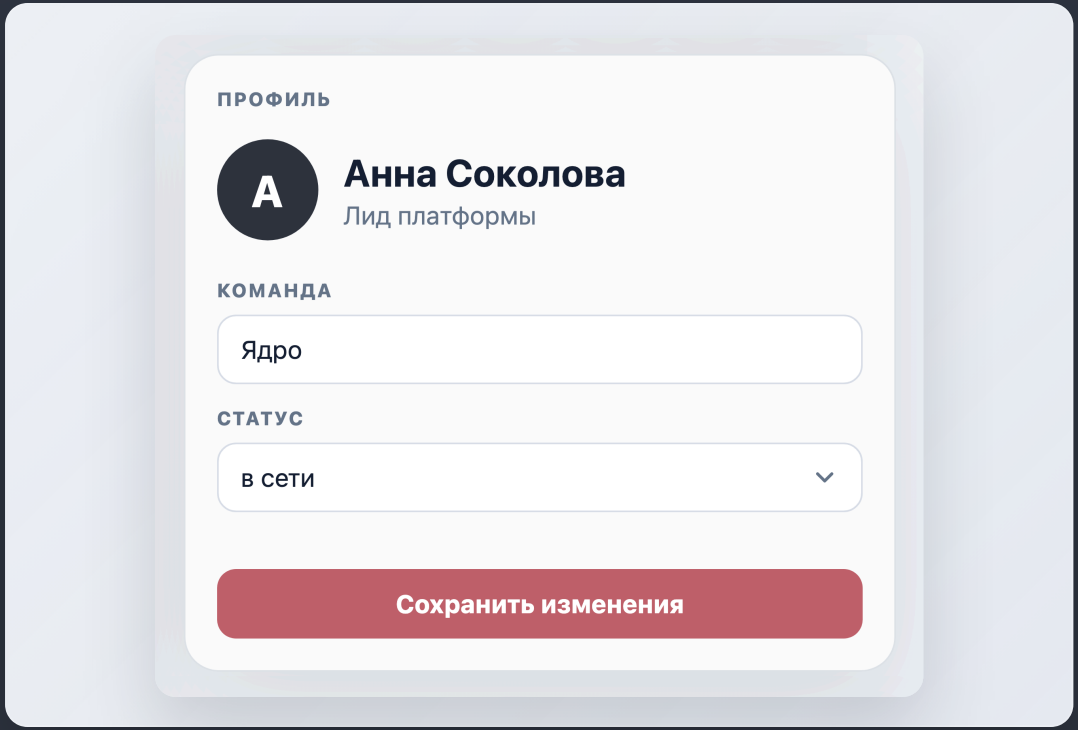
04

*«Формы и ресурсы данных — это магия **Angular?**»*

форма → submit → ресурс: данные / загрузка / ошибка

БОЛЬШЕ СТРУКТУРЫ, МЕНЬШЕ СЛУЧАЙНОГО GLUE CODE

Форма: где появляется сложность



Форма — это не просто input и select. Быстро появляется отдельный слой: значения формы, reset при смене пользователя, validation и submit.

1 **Пользователь**

выбранный пользователь приходит сверху; форма не решает, кого редактировать

2 **Значения формы**

поля можно менять, не переписывая сохраненный профиль после каждого ввода

3 **Validation / submit**

перед сохранением нужно проверить значения и отправить их наружу одним событием

Панель пользователей: форма — React / Vue

REACT
React Hook Form + Zod

VUE
VeeValidate + Zod

React

apps/react-users/src/app/profile-form.tsx

```tsx
const profileFormSchema = z.object({
  team: z.string().trim().min(1, 'Укажите команду'),
  status: z.enum(['в сети', 'не в сети']),
});

export function ProfileForm({ user, onSave }: ProfileFormProps) {
  const {
    register,
    handleSubmit,
    reset,
    formState: { errors },
  } = useForm<ProfileFormValue>({
    defaultValues: toFormValue(user),
    resolver: zodResolver(profileFormSchema),
  });

  useEffect(() => reset(toFormValue(user)), [reset, user]);

  return (
    <form onSubmit={handleSubmit(onSave)}>
      <input {...register('team')} />
      {errors.team && <small>{errors.team.message}</small>}
      <select {...register('status')}>...</select>
      <button>Сохранить изменения</button>
    </form>
  );
}
```

Vue

apps/vue-users/src/app/ProfileForm.vue

```vue
<template>
  <form @submit.prevent="submitProfile">
    <input v-model="team" />
    <small v-if="errors.team">{{ errors.team }}</small>
    <select v-model="status">...</select>
  </form>
</template>

<script setup lang="ts">
const profileFormSchema = z.object({
  team: z.string().trim().min(1, 'Укажите команду'),
  status: z.enum(['в сети', 'не в сети']),
});

const { defineField, errors, handleSubmit, resetForm } =
  useForm<ProfileFormValue>({
    initialValues: toFormValue(props.user),
    validationSchema: toTypedSchema(profileFormSchema),
  });

const [team] = defineField('team');
const [status] = defineField('status');
const submitProfile = handleSubmit((formValue) => emit('save', formValue));

watch(
  () => props.user,
  (user) => resetForm({ values: toFormValue(user) }),
);
</script>
```

Панель пользователей: Angular форма

Angular apps/angular-users/src/app/profile-form.ts

```ts
@Component({
  selector: 'app-profile-form',
  template: `
    <form>
      <input />
      <select>...</select>
      <button>Сохранить изменения</button>
    </form>
  `,
})
export class ProfileForm {
}
```

СТАРТ

сначала обычная форма

на экране уже виден компонент и template; дальше добавляем контракт, модель, поля и submit.

Панель пользователей: Angular форма

Angular apps/angular-users/src/app/profile-form.ts

```ts
@Component({
  selector: 'app-profile-form',
  template: `
    <form>
      <input />
      <select>...</select>
      <button>Сохранить изменения</button>
    </form>
  `,
})
export class ProfileForm {
  user = input.required<User>();
  save = output<ProfileFormValue>();
}
```

1. КОНТРАКТ

input.required + output

форма получает `user()` сверху и отправляет наружу только событие сохранения.

Панель пользователей: Angular форма

Angular apps/angular-users/src/app/profile-form.ts

```ts
@Component({
  selector: 'app-profile-form',
  template: `
    <form>
      <input />
      <select>...</select>
      <button>Сохранить изменения</button>
    </form>
  `,
})
export class ProfileForm {
  user = input.required<User>();
  save = output<ProfileFormValue>();

  protected model = linkedSignal<ProfileFormValue>(() => ({
    team: this.user().team,
    status: this.user().status,
  }));
}
```

1. КОНТРАКТ

input.required + output

форма получает `user()` сверху и отправляет наружу только событие сохранения.

2. ЗНАЧЕНИЯ ФОРМЫ

linkedSignal<ProfileFormValue>

при новом пользователе значения пересобираются, но редактирование остается локальным.

Панель пользователей: Angular форма

Angular apps/angular-users/src/app/profile-form.ts

```ts
import { FormField, form, required } from '@angular/forms/signals';

@Component({
  imports: [FormField],
  selector: 'app-profile-form',
  template: `
    <form>
      <input />
      <select>...</select>
      <button>Сохранить изменения</button>
    </form>
  `,
})
export class ProfileForm {
  user = input.required<User>();
  save = output<ProfileFormValue>();

  protected model = linkedSignal<ProfileFormValue>(() => ({
    team: this.user().team,
    status: this.user().status,
  }));

  protected profileForm = form(this.model, (path) => {
    required(path.team);
    required(path.status);
  });
}
```

1. КОНТРАКТ

input.required + output

форма получает `user()` сверху и отправляет наружу только событие сохранения.

2. ЗНАЧЕНИЯ ФОРМЫ

linkedSignal<ProfileFormValue>

при новом пользователе значения пересобираются, но редактирование остается локальным.

3. ФОРМА КАК ДЕРЕВО

form(model) + required(...)

validation описывается рядом с моделью, а не размазывается по обработчикам полей.

Панель пользователей: Angular форма

Angular apps/angular-users/src/app/profile-form.ts

```ts
import { FormField, form, required, submit } from '@angular/forms/signals';

@Component({
  imports: [FormField],
  selector: 'app-profile-form',
  template: `
    <form (submit)="saveProfile(); $event.preventDefault()">
      <input [formField]="profileForm.team" />
      <select [formField]="profileForm.status">...</select>
      <button>Сохранить изменения</button>
    </form>
  `,
})
export class ProfileForm {
  user = input.required<User>();
  save = output<ProfileFormValue>();

  protected model = linkedSignal<ProfileFormValue>(() => ({
    team: this.user().team,
    status: this.user().status,
  }));

  protected profileForm = form(this.model, (path) => {
    required(path.team);
    required(path.status);
  });

  protected saveProfile() {
    return submit(this.profileForm, async (field) =>
      this.save.emit(field().value()),
    );
  }
}
```

1. КОНТРАКТ

input.required + output

форма получает `user()` сверху и отправляет наружу только событие сохранения.

2. ЗНАЧЕНИЯ ФОРМЫ

linkedSignal<ProfileFormValue>

при новом пользователе значения пересобираются, но редактирование остается локальным.

3. ФОРМА КАК ДЕРЕВО

form(model) + required(...)

validation описывается рядом с моделью, а не размазывается по обработчикам полей.

4. TEMPLATE И SUBMIT

[formField] + submit(...)

поля подключаются без ручного value/change, а submit отправляет `ProfileFormValue`.

Вывод по форме: где живет состояние

`начальные значения` → `локальное состояние` → `валидный submit`

Одинаковая задача: взять пользователя, дать отредактировать поля и сохранить только валидный результат.

REACT

React Hook Form + Zod

`useForm` держит значения, `register` связывает поля, `zodResolver` проверяет submit.

VUE

VeeValidate + Zod

`defineField` дает поля для `v-model`, `toTypedSchema` подключает validation.

ANGULAR

Signal Forms

`form()` строит дерево полей, `FormField` связывает template, `submit` отдает value.

REACT / VUE

форма как выбранный стек

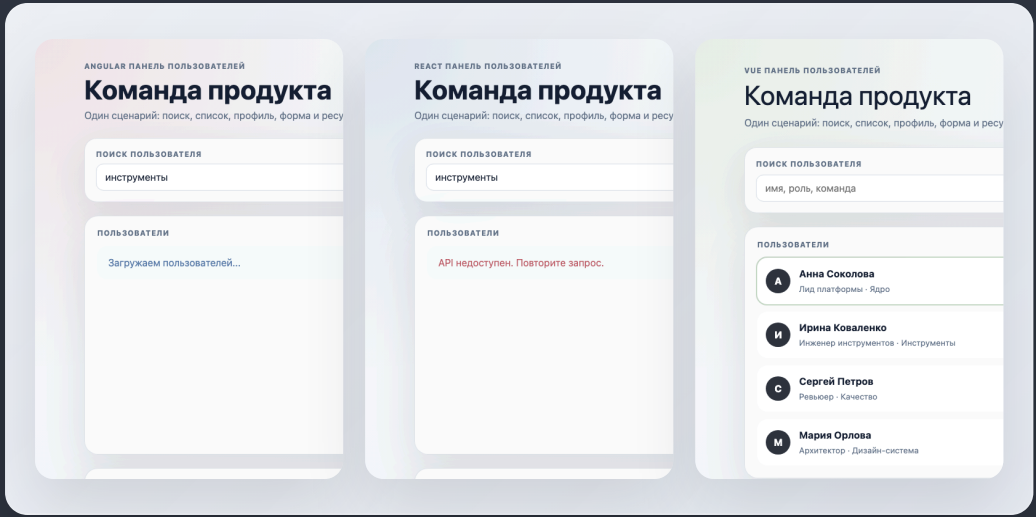
команда явно выбирает библиотеку формы, слой схемы и правила reset при смене пользователя.

ANGULAR

форма как контракт фреймворка

модель, поля, validation и submit собираются вокруг официального Angular Forms API.

Панель пользователей: ресурс данных



Визуально показываем исходы ресурса, а в коде ниже смотрим не список, а слой данных.

1 **Источник данных**

query или resource запускает загрузку пользователей из источника данных

2 **Состояние ресурса**

данные, загрузка и ошибка живут рядом с ресурсом, а не внутри списка

3 **Повтор**

refetch / reload — это действие слоя данных, а не ветка отрисовки строк

Панель пользователей: ресурс — React / Vue

REACT
TanStack Query

VUE
TanStack Vue Query

React — apps/react-users/src/app/use-users-resource.ts

```tsx
...
export function useUsersResource(resourceMode: ViewMode) {
  const usersQuery = useQuery({
    queryKey: ['users', resourceMode],
    queryFn: ({ signal }) =>
      loadUsers({
        delayMs: resourceMode === 'загрузка' ? 60_000 : 350,
        fail: resourceMode === 'ошибка',
        signal,
      }),
    retry: false,
  });

  return {
    users: usersQuery.data ?? [],
    isResourceLoading:
      usersQuery.isLoading || usersQuery.isFetching,
    resourceError: usersQuery.error,
    retryUsers: () => void usersQuery.refetch(),
  };
}
```

Vue — apps/vue-users/src/app/useUsersResource.ts

```tsx
...
export function useUsersResource(resourceMode: Ref<ViewMode>) {
  const usersQuery = useQuery({
    queryKey: computed(() => ['users', resourceMode.value]),
    queryFn: ({ signal }) =>
      loadUsers({
        delayMs: resourceMode.value === 'загрузка' ? 60_000 : 350,
        fail: resourceMode.value === 'ошибка',
        signal,
      }),
    retry: false,
  });

  return {
    users: computed(() => usersQuery.data.value ?? []),
    isResourceLoading: computed(() =>
      usersQuery.isLoading.value || usersQuery.isFetching.value,
    ),
    resourceError: computed(() => usersQuery.error.value),
    retryUsers: () => void usersQuery.refetch(),
  };
}
```

REACT
Ресурс здесь — Query: ключ, загрузчик, data/status/retry; hook только отдаёт контракт экрану.

VUE
Ресурс здесь — Vue Query: queryKey, queryFn, refs для data/status и retry.

Панель пользователей: ресурс — Angular

ANGULAR

UsersResource service

Angular apps/angular-users/src/app/users-resource.ts

```
...
@Injectable()
export class UsersResource {
  readonly mode = signal<ViewMode>('готово');
}
```

СТАРТ

page-scoped service

ресурс живет в сервисе страницы; компонент инжектит facade и рисует экран.

Панель пользователей: ресурс — Angular

ANGULAR

UsersResource service

Angular apps/angular-users/src/app/users-resource.ts

```
...
@Injectable()
export class UsersResource {
  readonly mode = signal<ViewMode>('готово');

  private readonly resourceRef = resource({
    loader: () => loadUsers(),
    defaultValue: [],
  });
}
```

1. РЕСУРС

resource(loader)

асинхронная загрузка становится отдельным ресурсом, а не набором UI-флагов.

Панель пользователей: ресурс — Angular

ANGULAR

UsersResource service

Angular apps/angular-users/src/app/users-resource.ts

```
...
@Injectable()
export class UsersResource {
  readonly mode = signal<ViewMode>('готово');

  private readonly resourceRef = resource({
    params: () => this.mode(),
    loader: ({ abortSignal, params }) =>
      loadUsers({
        delayMs: params === 'загрузка' ? 60_000 : 350,
        fail: params === 'ошибка',
        signal: abortSignal,
      }),
    defaultValue: [],
  });
}
```

1. РЕСУРС

`resource(loader)`

асинхронная загрузка становится отдельным ресурсом, а не набором UI-флагов.

2. КЛЮЧ И ОТМЕНА

`params + abortSignal`

режим становится ключом ресурса; при смене режима старая загрузка отменяется.

Панель пользователей: ресурс — Angular

ANGULAR

UsersResource service

Angular apps/angular-users/src/app/users-resource.ts

```angular
...
@Injectable()
export class UsersResource {
  readonly mode = signal<ViewMode>('готово');

  private readonly resourceRef = resource({
    params: () => this.mode(),
    loader: ({ abortSignal, params }) =>
      loadUsers({
        delayMs: params === 'загрузка' ? 60_000 : 350,
        fail: params === 'ошибка',
        signal: abortSignal,
      }),
    defaultValue: [],
  });

  readonly isLoading = this.resourceRef.isLoading;
  readonly error = this.resourceRef.error;

  readonly users = computed(() =>
    this.resourceRef.hasValue()
      ? this.resourceRef.value()
      : [],
  );
}
```

1. РЕСУРС

resource(loader)

асинхронная загрузка становится отдельным ресурсом, а не набором UI-флагов.

2. КЛЮЧ И ОТМЕНА

params + abortSignal

режим становится ключом ресурса; при смене режима старая загрузка отменяется.

3. ДАННЫЕ

users + status

сервис отдает список, loading и error; экран не собирает request state руками.

Панель пользователей: ресурс — Angular

ANGULAR

UsersResource service

Angular apps/angular-users/src/app/users-resource.ts

```ts
...
@Injectable()
export class UsersResource {
  readonly mode = signal<ViewMode>('готово');

  private readonly resourceRef = resource({
    params: () => this.mode(),
    loader: ({ abortSignal, params }) =>
      loadUsers({
        delayMs: params === 'загрузка' ? 60_000 : 350,
        fail: params === 'ошибка',
        signal: abortSignal,
      }),
    defaultValue: [],
  });

  readonly isLoading = this.resourceRef.isLoading;
  readonly error = this.resourceRef.error;

  readonly users = computed(() =>
    this.resourceRef.hasValue()
      ? this.resourceRef.value()
      : [],
  );

  reload() {
    this.resourceRef.reload();
  }
}
```

Angular template apps/angular-users/src/app/app.html

```ts
@else if (usersResource.error()) {
  <app-resource-error />
}
```

1. РЕСУРС

resource(loader)

асинхронная загрузка становится отдельным ресурсом, а не набором UI-флагов.

2. КЛЮЧ И ОТМЕНА

params + abortSignal

режим становится ключом ресурса; при смене режима старая загрузка отменяется.

3. ДАННЫЕ

users + status

сервис отдает список, loading и error; экран не собирает request state руками.

4. ГРАНИЦА RETRY

<app-resource-error />

ошибка уходит в дочерний компонент; доступ к reload покажем через DI дальше.

Вывод по ресурсу: где живет async state

параметры → loader / queryFn → data + status + retry

Одинаковая задача: прочитать пользователей, показать загрузку или ошибку и дать повторить запрос.

REACT

ресурс через TanStack Query

`queryKey` задает identity ресурса, `queryFn` грузит данные, Query отдает data/loading/error/retry.

VUE

ресурс через TanStack Vue Query

тот же lifecycle ресурса, но наружу уходят refs/computed для template.

ANGULAR

ресурс через resource()

`UsersResource` держит `params`, `loader`, status и `reload()` как facade страницы.

REACT / VUE

ресурс как server-state объект

слой доступа только доставляет контракт; async lifecycle живет в Query.

ANGULAR

ресурс как встроенный примитив

async lifecycle описывается через resource и отдается экрану через service facade.

Передача ресурса: React / Vue

REACT

props / Context Provider

VUE

props / provide/inject

React

apps/react-users/src/app/users-resource-context.tsx

```tsx
...

const UsersResourceContext =
  createContext<UsersResourceApi | null>(null);

export function UsersResourceProvider({ children }: PropsWithChildren) {
  const [resourceMode, setResourceMode] = useState<ViewMode>('готово');
  const resource = useUsersResource(resourceMode);

  const value = {
    ...resource,
    modes: VIEW_MODES,
    resourceMode,
    setResourceMode,
  };

  return (
    <UsersResourceContext.Provider value={value}>
      {children}
    </UsersResourceContext.Provider>
  );
}

export function useUsersResourceApi() {
  const resource = useContext(UsersResourceContext);
  if (!resource) throw new Error('UsersResource не подключен');
  return resource;
}
```

Vue

apps/vue-users/src/app/usersResourceProvider.ts

```ts
...

const usersResourceKey =
  Symbol() as InjectionKey<UsersResourceApi>;

export function provideUsersResource() {
  const resourceMode = ref<ViewMode>('готово');
  const resource = {
    ...useUsersResource(resourceMode),
    modes: VIEW_MODES,
    resourceMode,
  };

  provide(usersResourceKey, resource);
}

export function useUsersResourceApi() {
  const resource = inject(usersResourceKey);
  if (!resource) throw new Error('UsersResource не подключен');
  return resource;
}
```

REACT

Ресурс остается Query-объектом; Context только доставляет его ниже по дереву.

VUE

Ресурс остается Vue Query-объектом; provide/inject убирает ручное протаскивание props.

Передача ресурса: Angular DI

ANGULAR

providers + inject()

Angular app.ts + resource-error.ts

```ts
...

@Component({
  imports: [ProfileForm, UserRow],
  selector: 'app-root',
  templateUrl: './app.html',
  styleUrl: './app.css',
  providers: [UsersResource],
})
export class App {}
```

ОБЛАСТЬ

providers: [UsersResource]

экран получает свой экземпляр ресурса, а не глобальный singleton всего app.

Передача ресурса: Angular DI

ANGULAR

providers + inject()

Angular app.ts + resource-error.ts

```ts
...

@Component({
  imports: [ProfileForm, ResourceError, UserRow],
  selector: 'app-root',
  templateUrl: './app.html',
  styleUrl: './app.css',
  providers: [UsersResource],
})
export class App {
  protected readonly usersResource = inject(UsersResource);
  protected readonly resourceMode = this.usersResource.mode;
  protected readonly users = this.usersResource.users;
}
```

ОБЛАСТЬ

providers: [UsersResource]

экран получает свой экземпляр ресурса, а не глобальный singleton всего app.

ДОСТУП

inject(UsersResource)

компонент просит зависимость у injector, а не получает ее через цепочку props.

Передача ресурса: Angular DI

ANGULAR

providers + inject()

Angular — app.ts + resource-error.ts

```ts
...

@Component({
  imports: [ProfileForm, ResourceError, UserRow],
  selector: 'app-root',
  templateUrl: './app.html',
  styleUrl: './app.css',
  providers: [UsersResource],
})
export class App {
  protected readonly usersResource = inject(UsersResource);
  protected readonly resourceMode = this.usersResource.mode;
  protected readonly users = this.usersResource.users;
}

@Component({
  selector: 'app-resource-error',
  template: `
    <button class="state-card error" (click)="usersResource.reload()">
      API недоступен. Повторите запрос.
    </button>
  `,
})
export class ResourceError {
  protected readonly usersResource = inject(UsersResource);
}
```

ОБЛАСТЬ

providers: [UsersResource]

экран получает свой экземпляр ресурса, а не глобальный singleton всего app.

ДОСТУП

inject(UsersResource)

компонент просит зависимость у injector, а не получает ее через цепочку props.

КОНТРАКТ РЕСУРСА

users / status / reload

экран работает с готовой моделью ресурса, а не с ручным набором request flags.

АНАЛОГИЯ

Context / provide/inject решают доступ; Angular DI делает это правилом платформы

Итог блока: это не магия, а контракты

форма → ресурс → доступ

Один экран распадается на три повторяемых договора: локальные значения формы, async state ресурса и способ доставить этот ресурс ниже.

1. ФОРМА

initial values → validation → submit

форма держит локальные значения, проверяет их и наружу отдает только валидный результат сохранения.

2. РЕСУРС

params → loader/queryFn → retry

загрузка, ошибка, данные и повтор живут в отдельной границе, а список только потребляет готовые users.

3. ДОСТУП

Context / provide / DI

компоненты получают готовый resource contract без ручного протаскивания props через все дерево.

REACT / VUE

выбранный stack + conventions

формы, ресурсы и доступ собираются из библиотек и командных правил: это гибко, но требует выбора.

ANGULAR

platform contracts + DI

Angular дает общий язык для тех же задач, но цена — выучить API платформы и их текущий статус.

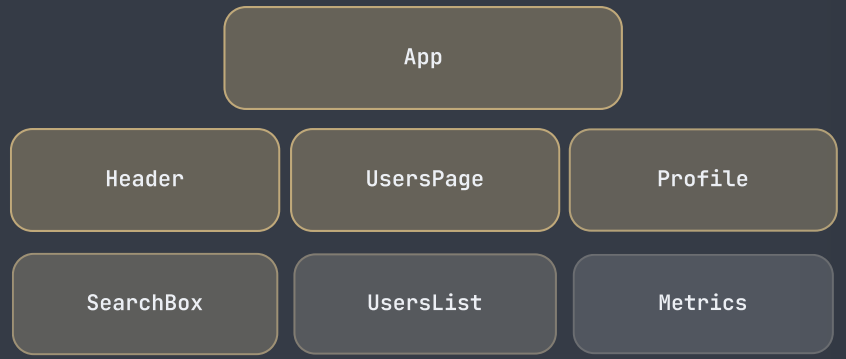
05

«Angular слишком тяжёлый: runtime, загрузка, тесты»

ZONELESS, LAZY ROUTES, @DEFER И VITEST

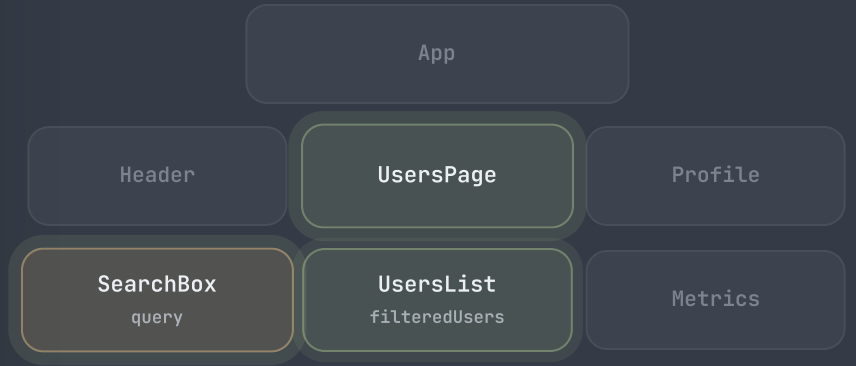
Zone.js vs Signals

Zone.js



событие → широкий проход по компонентам

Signals



query changed → пересчитать фильтр и список

Идея: от общего радара событий к графу зависимостей.

Роутинг: lazy loading на уровне маршрутов

REACT ROUTER

`layout + index + :userId`

React app/router.tsx

```tsx
...
export const router = createBrowserRouter([
  {
    path: '/users',
    lazy: () => import('./routes/users-layout'),
    children: [
      {
        index: true,
        lazy: () => import('./routes/users-list'),
      },
      {
        path: ':userId',
        lazy: () => import('./routes/user-details'),
      },
    ],
  },
]);
```

REACT

`children` задает вложенность; `lazy` грузит layout, list и details отдельно.

VUE ROUTER

`layout + index + :userId`

Vue router.ts

```ts
...
const routes = [
  {
    path: '/users',
    component: () =>
      import('./pages/UsersLayout.vue'),
    children: [
      {
        path: '',
        component: () => import('./pages/UsersPage.vue'),
      },
      {
        path: ':userId',
        component: () => import('./pages/UserDetailsPage.vue
      },
    ],
  },
];
```

VUE

`children` задает вложенность; dynamic import грузит layout, list и details отдельно.

ANGULAR ROUTER

`layout + index + :userId`

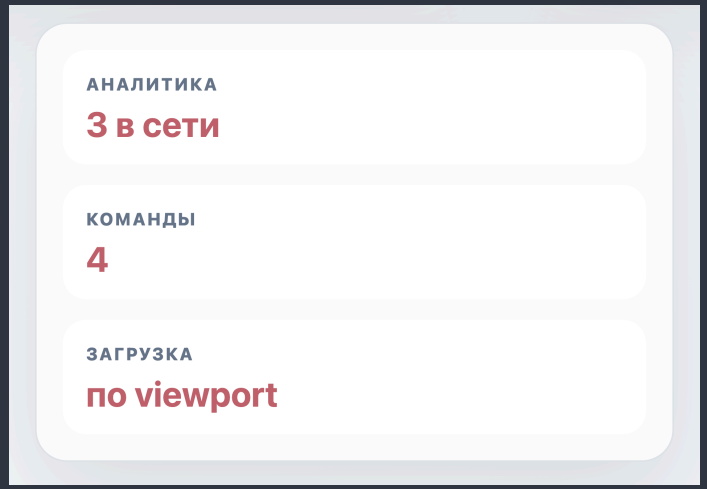
Angular app.routes.ts

```ts
...
export const routes: Routes = [
  {
    path: 'users',
    loadComponent: () => import('./users/users-layout')
      .then((m) => m.UsersLayout),
    children: [
      {
        path: '',
        loadComponent: () => import('./users/users.page')
          .then((m) => m.UsersPage),
      },
      {
        path: ':userId',
        loadComponent: () => import('./users/user-details.pa
          .then((m) => m.UserDetailsPage),
      },
    ],
  },
];
```

ANGULAR

`children` задает вложенность; `loadComponent` грузит layout, list и details отдельно.

Отложенный UI: блок аналитики



Когда грузить

блок ниже основного сценария можно грузить по viewport, а не вместе со стартовым экраном

Что показать пока грузится

placeholder до входа в viewport и loading-состояние во время загрузки chunk

Что делать при ошибке

ошибка ленивого блока должна остаться внутри блока, а не ломать всю страницу

Общая задача: аналитика полезна, но она не должна утяжелять первый экран панели

Отложенный UI: React / Vue

React — apps/react-users/src/app/users-panel.tsx

```tsx
...
const UserInsights = lazy(() => import('./user-insights'));

export function UsersPanel() {
  const { ref, visible } = useInViewport();

  return (
    <section className="insights-slot" ref={ref}>
      {visible ? (
        <Suspense fallback={
          <div className="state-card loading">
            Собираем аналитику...
          </div>
        }>
          <UserInsights users={users} />
        </Suspense>
      ) : (
        <div className="state-card">
          Аналитика появится ниже по экрану.
        </div>
      )}
    </section>
  );
}
```

Vue — apps/vue-users/src/app/UsersPanel.vue

```vue
<script setup lang="ts">
...
const UserInsights =
  defineAsyncComponent(() => import('./UserInsights.vue'));
const insightsHost = ref<HTMLElement | null>(null);
const showInsights = ref(false);
let observer: IntersectionObserver | undefined;

onMounted(() => {
  observer = new IntersectionObserver(([entry]) => {
    if (entry?.isIntersecting) showInsights.value = true;
  });
  observer.observe(insightsHost.value!);
});

onBeforeUnmount(() => observer?.disconnect());
</script>

<template>
  <section ref="insightsHost" class="insights-slot">
    <Suspense v-if="showInsights">
      <UserInsights :users="users" />
      <template #fallback>
        <div class="state-card loading">
          Собираем аналитику...
        </div>
      </template>
    </Suspense>
    <div v-else class="state-card">
      Аналитика появится ниже по экрану.
    </div>
  </section>
</template>
```

REACT

`lazy` и `Suspense` закрывают chunk и fallback; viewport-триггер живет в hook.

VUE

`defineAsyncComponent` и `Suspense` закрывают chunk и fallback; viewport-триггер добавляет observer.

Отложенный UI: Angular @defer

Angular apps/angular-users/src/app/app.html

```
...
<section class="insights-slot">
  @defer {
    <app-user-insights [users]="users()" />
  }
</section>
```

СТАРТ

@defer block

отложенный компонент уже выделен в отдельную границу прямо в template.

Отложенный UI: Angular @defer

Angular apps/angular-users/src/app/app.html

```
...
<section class="insights-slot">
  @defer (on viewport) {
    <app-user-insights [users]="users()" />
  }
</section>
```

1. ТРИГГЕР

on viewport

аналитика начинает грузиться тогда, когда пользователь доскроллил до блока.

Отложенный UI: Angular @defer

Angular apps/angular-users/src/app/app.html

```
...
<section class="insights-slot">
  @defer (on viewport; prefetch on idle) {
    <app-user-insights [users]="users()" />
  }
</section>
```

1. ТРИГГЕР

on viewport

аналитика начинает грузиться тогда, когда пользователь доскроллил до блока.

2. PREFETCH

prefetch on idle

chunk можно подкачать в свободный момент, но показать блок только по trigger.

Отложенный UI: Angular @defer

Angular apps/angular-users/src/app/app.html

```
...
<section class="insights-slot">
  @defer (on viewport; prefetch on idle) {
    <app-user-insights [users]="users()" />
  } @placeholder {
    <div class="state-card">
      Аналитика появится ниже по экрану.
    </div>
  }
</section>
```

1. ТРИГГЕР

on viewport

аналитика начинает грузиться тогда, когда пользователь доскроллил до блока.

2. PREFETCH

prefetch on idle

chunk можно подкачать в свободный момент, но показать блок только по trigger.

3. ОЖИДАНИЕ

@placeholder

до загрузки пользователь видит легкую заглушку на месте будущей аналитики.

Отложенный UI: Angular @defer

Angular — apps/angular-users/src/app/app.html

```
...
<section class="insights-slot">
  @defer (on viewport; prefetch on idle) {
    <app-user-insights [users]="users()" />
  } @placeholder {
    <div class="state-card">
      Аналитика появится ниже по экрану.
    </div>
  } @loading (minimum 400ms) {
    <div class="state-card loading">
      Собираем аналитику...
    </div>
  } @error {
    <div class="state-card error">
      Не удалось загрузить аналитику.
    </div>
  }
</section>
```

Lazy component — apps/angular-users/src/app/user-insights.ts

```
@Component({
  selector: 'app-user-insights',
  template: `
    <section class="insights-card">
      <strong>{{ activeUsers() }} в сети</strong>
    </section>
  `,
})
export class UserInsights {
  users = input.required<User[]>();
  activeUsers = computed(() =>
    this.users().filter((user) => user.status === 'в сети').length,
  );
}
```

1. ТРИГГЕР

on viewport

аналитика начинает грузиться тогда, когда пользователь доскроллил до блока.

2. PREFETCH

prefetch on idle

chunk можно подкачать в свободный момент, но показать блок только по trigger.

3. ОЖИДАНИЕ

@placeholder

до загрузки пользователь видит легкую заглушку на месте будущей аналитики.

4. СОСТОЯНИЯ

@loading / @error

loading и ошибка описаны рядом с тем UI, который они защищают.

Итог: отложенный UI — это граница

`route` → `viewport` → `state`

Роутер делит приложение на страницы, а defer/lazy component делит уже открытую страницу на основной путь и поздний блок.

REACT

lazy + Suspense + observer

chunk и fallback встроены, но viewport-trigger, retry и prefetch обычно становятся отдельным hook или library convention.

VUE

defineAsyncComponent + Suspense + observer

async component дает lazy chunk, Suspense дает fallback, а момент загрузки задается дополнительной реактивной логикой.

ANGULAR

@defer block

trigger, prefetch, placeholder, loading и error описаны рядом с тем блоком UI, который откладываем.

ЧТО НЕ МЕНЯЕТСЯ

компонент остается обычным компонентом

UserInsights все равно принимает users и считает derived state. Lazy-механизм меняет момент загрузки, а не контракт компонента.

ГДЕ ЦЕНА

границу нужно выбирать осознанно

Откладывать стоит тяжелый или вторичный UI: аналитика, графики, редакторы, карты, виджеты ниже первого экрана.

Тесты: старый образ стал легче

UNIT	KARMA БЫЛО **Karma + браузер** тяжелый образ даже для простых unit-тестов.	→
	СТАЛО БЛИЖЕ К FRONTEND DEFAULT **Vitest + jsdom** быстрее и привычнее для новых Angular-проектов.	
E2E	Protractor БЫЛО **Protractor** старый Angular-first e2e образ.	→
	СОВРЕМЕННАЯ ПРАКТИКА **Playwright** проверка пользовательских сценариев в реальном браузере.	

> **Короткий вывод:** мы не уходим в тему тестов, просто закрываем еще одну часть мифа про “тяжелый Angular”.

Итог блока: тяжесть стала управляемой

`runtime` → `loading` → `tooling`

Вопрос уже не “Angular тяжелый или нет”, а где платформа дает цену и где дает контроль.

1. RUNTIME

Signals + zoneless

модель обновлений стала явнее: не только широкий проход после каждого async-события.

2. ЗАГРУЗКА

lazy routes + @defer

код можно делить на уровне страниц и внутри уже открытой страницы.

3. TOOLING

Vitest + Playwright

unit и e2e сценарии ближе к современному frontend tooling.

ЧЕСТНАЯ ЦЕНА

платформа все еще требует правил

нужно понимать границы signals, lazy loading, defer-блоков и Angular runtime.

ЧЕСТНАЯ ПОЛЬЗА

меньше самодельной инфраструктуры

современный Angular дает официальные способы контролировать runtime, bundle и часть tooling.

06

«У Angular нет meta-framework истории»

ПЛАТФОРМА ANGULAR + СЛОЙ ANALOG

Meta-framework: один слой, разные UI-базы

UI-ФРЕЙМВОРК

React

Next

META-FRAMEWORK

маршруты, SSR/SSG, server routes, content и deployment story поверх React.

UI-ФРЕЙМВОРК

Vue

Nuxt

META-FRAMEWORK

те же задачи вокруг Vue: routing, rendering modes, server layer и content.

UI-ФРЕЙМВОРК

Angular

Analog

META-FRAMEWORK

Angular-приложение получает знакомый слой: file-based routes, SSR/SSG, API и content.

> **Идея:** спор не “Angular против Next”, а нужен ли Angular-приложению такой же слой поверх UI.

Next / Nuxt / Analog: сравнение по функциям

ФУНКЦИЯ	NEXT	NUXT	ANALOG
UI-база	React	Vue	Angular
Файловые маршруты	`app/.../page.tsx`	`pages/*.vue`	`app/pages/*.page.ts`
Rendering	static / dynamic / streaming	universal rendering + prerender	hybrid SSR / SSG
API / server routes	`route.ts`	`server/api` , `server/routes`	`server/routes/api`
Server data	Server Components / Server Functions	`useFetch` , server utils	`*.server.ts` load рядом со страницей
Content / Markdown	MDX / интеграции	Nuxt Content	Markdown content routes
Что требует инвестиций	RSC/cache mental model	конвенции Nuxt/Nitro	более молодая экосистема

> **Вывод:** Analog закрывает знакомый набор meta-framework задач, но это выбор для Angular-проектов, где этот слой реально нужен.

Вывод по SSR и meta-framework

ANGULAR SSR

базовый путь внутри Angular

подходит, когда нужен server rendering, hydration и route-level RenderMode без смены архитектурного стиля.

ANALOG

meta-framework слой поверх Angular

имеет смысл, когда нужны file-based routes, server routes, content и опыт ближе к Next/Nuxt.

НЕ МИФ

у Angular есть SSR-история

это не только SPA: можно выбирать CSR, SSR и prerender по маршрутам.

ЧЕСТНАЯ ГРАНИЦА

Analog не нужен каждому проекту

это дополнительный слой соглашений, а не обязательный вход в Angular.

07

Что делать с этими мифами

ФИНАЛЬНЫЙ ВЫБОР: ЦЕНА ПЛАТФОРМЫ И ПОЛЬЗА ДЛЯ КОМАНДЫ

Собираем картину

ПОСЛЕ ВСЕХ ПРИМЕРОВ

Миф перестал быть общим ярлыком

Мы больше не спорим “Angular сложный”. Мы видим конкретные зоны: компонент, состояние, форма, ресурсы, загрузка и SSR.

ЦЕНА

принять Angular Way

термины, стиль фреймворка, встроенные соглашения и меньше локальной свободы.

ПОЛЬЗА

не собирать фундамент заново

готовые связки для маршрутов, форм, данных, загрузки, SSR и поддержки команды.

ФИНАЛЬНЫЙ ВОПРОС **окупается ли готовый Angular Way именно в вашем продукте?**

Как выбрать путь

Angular

Повторяются одни и те же решения

много экранов, форм, ролей, запросов и маршрутов; важнее общий рецепт, чем выбор каждой библиотеки заново

React

Нужно точно собрать свой стек

команда осознанно выбирает router, forms, data layer, SSR и сама держит соглашения между проектами

Vue

Нужен мягкий вход и постепенность

меньше обязательных соглашений на старте, проще внедрять частями и наращивать структуру по мере роста

> **Ключевой выбор:** свобода сборки или готовый Angular Way.

Как выбрать путь

Angular

Повторяются одни и те же решения

много экранов, форм, ролей, запросов и маршрутов; важнее общий рецепт, чем выбор каждой библиотеки заново

React

Нужно точно собрать свой стек

команда осознанно выбирает router, forms, data layer, SSR и сама держит соглашения между проектами

Vue

Нужен мягкий вход и постепенность

меньше обязательных соглашений на старте, проще внедрять частями и наращивать структуру по мере роста

> **Ключевой выбор:** свобода сборки или готовый Angular Way.

Безопасная проверка

01 **Docs** `angular.dev`

официальная точка входа: components, signals, routing, forms, SSR

02 **Playground** `angular.dev/playground`

потрогать Angular Way в браузере, без локальной установки

03 **Local starter** `angular.dev/installation`

если нужно проверить tooling, сборку и реальный editor workflow

МИНИМАЛЬНЫЙ ЭКСПЕРИМЕНТ

Один продуктовый сценарий

- `route`
- `form`
- `data loading`
- `error state`

Берём маленький, но реальный кусок продукта. Не абстрактную todo-демку.

Смотрим: код · вопросы · время · поддержку

Следующий шаг: не миграция, а короткий проверочный сценарий.

Что меняется в работе команды

ДО

Каждая команда собирает свой набор

- router, forms, data layer, SSR, testing
- локальные договоренности и разные стили
- сложнее переносить людей между проектами

Angular Way

готовое комплексное решение, которое активно развивается и поддерживается

ПОСЛЕ

У команды появляется общий путь

- единая модель компонентов, DI, форм и маршрутов
- меньше повторных архитектурных обсуждений
- понятнее onboarding и поддержка через год

Цена: принять правила Angular. **Польза:** не собирать фундамент заново.

Финальная проверка мифов

Современный Angular

standalone / signals

Платформа

Router / Forms / DI

Meta-framework

Analog

> **Итог:** платформа с ценой и пользой.

Angular 21 — это не **React, но сложнее**.
Это набор общих решений.

Router, Forms, DI, HTTP, SSR *и* ***tooling*** *как общий путь команды.*