

Angular

Полнофункциональный фреймворк для веб-приложений

Занятие 8

Frontend Course 2025

Что мы уже знаем?

Базовый стек современного фронтенда:

- **HTML** - структура контента
- **CSS** - стилизация и адаптивность
- **JavaScript** - интерактивность и DOM манипуляции
- **TypeScript** - статическая типизация
- **Vite** - быстрая сборка и разработка

Вопрос: Зачем нужен фреймворк, если у нас уже всё это есть?

Проблемы "ванильного" подхода

Без фреймворка: много ручной работы и повторяющегося кода

Типичные задачи в приложениях:

- Роутинг между страницами
- Управление состоянием приложения
- Работа с формами и валидация
- Тестирование компонентов
- Оптимизация производительности

Пример без фреймворка:

```
// Ручное обновление DOM
function updateUserList(users) {
  const container = document.getElementById("user-list");
  container.innerHTML = "";
  users.forEach((user) => {
    const div = document.createElement("div");
    div.innerHTML = `<h3>${user.name}</h3><p>${user.email}</p>`;
    container.appendChild(div);
  });
}
```

Что такое Angular?

Вопрос: Это библиотека или фреймворк?

Полнофункциональный фреймворк! Complete solution для приложений.

Что включает Angular:

- **Компоненты** - переиспользуемые UI блоки
- **Роутер** - навигация между страницами
- **Forms** - работа с формами и валидацией
- **HTTP Client** - общение с сервером
- **DI Container** - внедрение зависимостей
- **RxJS** - реактивное программирование
- **Testing** - инструменты для тестирования

Философия Angular:

"Одни и те же инструменты для всех задач"

История Angular

AngularJS (v1.x) - 2010

- Первый фреймворк для SPA
- Двусторонняя привязка данных
- MVC/MVVM архитектура
- JavaScript (без TypeScript)

Angular (v2+) - 2016

- Полный rewrite с TypeScript
- Компонентный подход
- RxJS для реактивности
- Mobile-first дизайн
- Регулярные мажорные релизы (каждые 6 месяцев)

Текущая версия: Angular 21 (2025)

Когда выбирать Angular?

Enterprise приложения: большие команды, долгая поддержка

Идеально подходит для:

- **Корпоративные системы** - CRM, ERP, банковские порталы
- **Административные панели** - сложные формы и таблицы
- **Dashboards** - много данных и графиков
- **Large-scale SPA** - сотни экранов

Реальные примеры:

- **Microsoft Office** - веб-версия Office 365
- **Google Cloud Console** - управление облаком
- **Forbes** - новостной портал
- **BMW, Mercedes** - конфигураторы автомобилей

Архитектура Angular

Структура:

```
src/
├── app/
│   ├── components/    # UI компоненты
│   ├── services/      # Бизнес-логика
│   ├── models/        # Типы данных
│   ├── guards/        # Защита роутов
│   ├── interceptors/  # Перехватчики HTTP
│   ├── app.config.ts  # Init
│   ├── app.component.ts
│   └── app-routing.module.ts
├── assets/            # Статика
└── environments/      # Конфигурации
```

Ключевые концепции:

- **Module** (Deprecated) - контейнер для связанного кода
- **Component** - UI с логикой
- **Service** - переиспользуемая логика
- **Dependency Injection** - автоматическое подключение зависимостей

Компоненты - основа Angular

Вопрос: Что такое компонент?

Компонент = Шаблон + Логика + Стили

```
// user-card.component.ts
@Component({
  selector: "app-user-card", // HTML тег
  templateUrl: "./user-card.component.html",
  styleUrls: ["./user-card.component.css"],
})
export class UserCardComponent {
  @Input() user: User; // Входные данные
  @Output() edit = new EventEmitter<User>();

  onEdit() {
    this.edit.emit(this.user);
  }
}
```

```
<!-- user-card.component.html -->
<div class="user-card">
  <h3>{{ user.name }}</h3>
  <p>{{ user.email }}</p>
  <button (click)="onEdit()">Редактировать</button>
</div>
```

Двусторонняя привязка данных

Синхронизация между данными и UI

One-way binding:

```
<!-- Данные → шаблон -->
<h1>{{ title }}</h1>
<img [src]="imageUrl" />
<!-- Шаблон → данные -->
<input (input)="onInputChange($event)" />
<button (click)="save()">Сохранить</button>
```

Two-way binding:

```
<!-- Данные ↔ шаблон -->
<input [(ngModel)]="username" />
```

Свойства и события:

```
<!-- Property binding -->
<div [class.active]="isActive"></div>
<div [style.color]="textColor"></div>
<!-- Event binding -->
<div (mouseover)="onHover()"></div>
<form (submit)="onSubmit($event)"></form>
```

Директивы - расширение HTML

Типы директив:

1. Structural (меняют структуру DOM):

```
<!-- Условный рендеринг -->
<div *ngIf="isLoggedIn">
  <p>Добро пожаловать!</p>
</div>

<!-- Циклы -->
<div *ngFor="let item of items; trackBy: trackByFn">
  <span>{{ item.name }}</span>
</div>

<!-- Switch -->
<div [ngSwitch]="userRole">
  <p *ngSwitchCase="'admin'">Администратор</p>
  <p *ngSwitchCase="'user'">Пользователь</p>
  <p *ngSwitchDefault>Гость</p>
</div>
```

2. Attribute (меняют внешний вид):

```
<p [ngClass]="{ 'active': isActive, 'disabled': isDisabled }">
  Текст с классами
</p>

<p [ngStyle]="{ 'color': textColor, 'font-size': fontSize + 'px' }">
  Стилизованный текст
</p>
```

Services, Dependency Injection

Вопрос: Как делиться логикой между компонентами?

Services + DI! Автоматическое внедрение зависимостей.

```
// user.service.ts
@Injectable({
  providedIn: "root", // Синглтон на всё приложение
})
export class UserService {
  private apiUrl = "https://api.example.com/users";

  constructor(private http: HttpClient) {}

  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.apiUrl);
  }

  createUser(user: User): Observable<User> {
    return this.http.post<User>(this.apiUrl, user);
  }
}
```

Services, Dependency Injection

Вопрос: Как делиться логикой между компонентами?

```
// user-list.component.ts
@Component({...})
export class UserListComponent implements OnInit {
  private readonly userService = inject(UserService);

  users: User[] = [];

  // Angular автоматически создаст и передаст UserService
  constructor(private userService: UserService) {} // Deprecated

  ngOnInit() {
    this.userService.getUsers().subscribe(users => {
      this.users = users;
    });
  }
}
```

RxJS - реактивное программирование

Observables = потоки данных во времени

Операторы RxJS:

- `map` - трансформация данных
- `filter` - фильтрация
- `debounceTime` - задержка выполнения
- `switchMap` - отмена предыдущих запросов
- `combineLatest` - объединение потоков

Преимущества:

- Автоматическая отписка (AsyncPipe)
- Отмена HTTP запросов
- Сложные сценарии обработки данных

Angular Forms

Два подхода к формам:

Template-driven (просто):

```
<form #userForm="ngForm" (ngSubmit)="save(userForm.value)">
  <input name="name" ngModel required />
  <input name="email" ngModel email />
  <button [disabled]="!userForm.valid">Сохранить</button>
</form>
```

Reactive Forms (мощно):

```
this.userForm = this.fb.group({
  name: ['', [Validators.required, Validators.minLength(3)]],
  email: ['', [Validators.required, Validators.email]],
  address: this.fb.group({
    street: ['', ],
    city: ['', ],
  }),
});
```

```
<form [formGroup]="userForm" (ngSubmit)="save()">
  <input formControlName="name" />
  <div *ngIf="userForm.get('name')?.errors?.required">Имя обязательно</div>
</form>
```

Angular Router

Клиентский роутинг с lazy loading

Конфигурация роутов:

```
const routes: Routes = [  
  { path: "", component: HomeComponent },  
  {  
    path: "users",  
    loadChildren: () =>  
      import("./users/users.module").then((m) => m.UsersModule),  
  },  
  {  
    path: "users/:id",  
    component: UserDetailComponent,  
    resolve: { user: UserResolver },  
  },  
  { path: "**", redirectTo: "/" },  
];
```

Использование в шаблоне:

```
<nav>  
  <a routerLink="/">Главная</a>  
  <a routerLink="/users">Пользователи</a>  
  <a [routerLink]="['/users', userId]">Детали</a>  
</nav>  
  
<router-outlet></router-outlet>
```

Angular CLI - профессиональные инструменты

Все готовые инструменты из коробки

Генерация кода:

```
# Создание проекта
ng new my-app

# Генерация компонентов, сервисов и т.д.
ng generate component user-list
ng generate service user
ng generate module admin
ng generate interface User
ng generate guard auth
```

Сборка и разработка:

```
# Запуск dev сервера
ng serve
# Production сборка
ng build --prod
# Запуск тестов
ng test
# E2E тесты
ng e2e
# Линтинг
ng lint
```

Angular vs React vs Vue

Сравнение фреймворков 2025:

| Критерий | Angular | React | Vue |
|--------------------|-------------------|-------------------|-------------------------|
| Подход | Фреймворк | Библиотека | Прогрессивный фреймворк |
| Язык | TypeScript | JavaScript/TS | JavaScript/TS |
| Размер | ~143KB | ~42KB | ~34KB |
| Кривая обучения | ★★★★ | ★★ | ★ |
| Производительность | Высокая | Очень высокая | Высокая |
| Экосистема | Полная | Большая | Средняя |
| Разработчик | Google | Facebook | Evan You |
| Компании | Microsoft, Google | Facebook, Netflix | Alibaba, GitLab |

Преимущества Angular

Почему выбирают Angular:

Для бизнеса:

- Долгосрочная поддержка (6 месяцев LTS)
- Предсказуемые релизы
- Большая экосистема enterprise-решений
- Много готовых специалистов

Для разработчиков:

- Все инструменты из коробки
- Строгая архитектура
- Отличная TypeScript интеграция
- Генераторы кода
- Автоматические миграции

Преимущества Angular

Почему выбирают Angular:

Для команд:

- Единый стиль кода
- Легко onboarding новых
- Масштабируемость проектов
- Встроенное тестирование

Недостатки Angular

Когда Angular не лучший выбор?

Сложность и избыточность для простых задач

Минусы:

- **Высокий порог входа** - нужно изучить много концепций
- **Больной размер бандла** - даже для простого приложения
- **Сложность** - RxJS, Dependency Injection, Modules
- **Меньше гибкости** - "сделай как Angular"
- **Медленнее для небольших проектов** - из-за overhead

Когда начинать с Angular?

Идеальный сценарий для Angular:

Начните с Angular если:

- Делаете enterprise приложение
- Работаете в большой команде
- Нужна долгосрочная поддержка проекта
- Требуется строгая архитектура
- Команда уже знает TypeScript
- Нужны встроенные решения для всех задач

Выберите альтернативу если:

- Делаешь простой сайт-визитку
- Нужен максимальный перфоманс
- Начинающий разработчик
- Проект с быстро меняющимися требованиями
- Минимальный размер bundle критичен

Спасибо!

Вопросы?

Frontend Course 2025