**Підручник AngularJS**

**Зміст**

1 Введення в AngularJS  
2 Engineering concepts in JavaScript frameworks  
3 Modules  
4 Understanding $scope  
5 Controllers  
6 Services and Factories  
7 Templating with the Angular core  
8 Directives (Core)  
9 Directives (Custom)  
10 Filters (Core)  
11 Filters (Custom)  
12 Dynamic with routing $routeProvider  
13 Form Validation  
14 communication Server with $http and $resource

**1. Введення у AngularJS**

Angular — MVW-фреймворк для розробки якісних клієнтських веб-додатків на JavaScript. Він створений і підтримується в Google і пропонує поглянути на майбутнє веба, на те, які нові можливості і стандарти готує для нас.  
MVW означає Model-View-Whatever (модель — вид — що завгодно), тобто гнучкість у виборі шаблонів проектування при розробці додатків. Ми можемо вибрати моделі MVC (Model-View-Controller) або MVVM (Model-View-ViewModel).  
Цей навчальний матеріал замислювався як відправна точка для вивчення AngularJS, його концепцій і API, щоб допомогти вам створювати чудові веб-додатки сучасним способом.  
AngularJS позиціонує себе як фреймворк, поліпшує HTML. Він зібрав концепції з різних мов програмування, як JavaScript, так і серверних, і робить з HTML також щось динамічне. Ми отримуємо підхід, заснований на даних, до розробки додатків. Немає потреби оновлювати Модель, DOM або робити якісь інші витратні часу операції, наприклад, виправляти помилки браузерів. Ми концентруємося на даних, дані ж піклуються про HTML, а ми просто займаємося програмуванням програми.

**Інженерні концепції в фрейморках JavaScript**

Позиція AngularJS по роботі з даними та іншими інженерними концепціями відрізняється від таких фреймворків, як Backbone.js and Ember.js. Ми задовольняємося вже відомим нам HTML, а Angular самостійно покращує його. Angular оновлює DOM при будь-яких змінах Моделі, яка живе собі в чистих Об'єктах JavaScript з метою зв'язку з даними. Коли оновлюється Модель, Angular оновлює Об'єкти, які містять актуальну інформацію про стан програми.

**2.1 MVC і MVVM**

Якщо ви звикли робити статичні сайти, вам знайомий процес створення HTML вручну, шматочок за шматочком, коли ви вписуєте в сторінку потрібні дані і повторюєте подібні HTML знову і знову. Це можуть бути стовпці решітки, структура для навігації, список посилань або картинок, і т. п. Коли змінюється одна маленька деталь, доводиться оновлювати весь шаблон, і всі наступні його використання. Також доводиться копіювати однакові шматки коду для кожного елемента навігації.  
Тримайтеся за крісло — Angular існує поділ обов'язків і динамічний HTML. А це означає, що наші дані живуть в Моделі, наш HTML живе у вигляді маленького шаблону, який буде перетворено на Вигляд, а Контролер ми використовуємо для з'єднання двох цих понять, забезпечуючи підтримку змін Моделі і Виду. Тобто, навігація може виводитися динамічно, створюючись з одного елемента списку, автоматично повторюватися для кожного пункту з Моделі. Це спрощена концепція, пізніше ми ще поговоримо про шаблони.  
Різниця між MVC і MVVM в тому, що MVVM спеціально призначений для розробки інтерфейсів. Вид складається з шару презентації, ВидМодель містить логіку презентації, а Модель містить бізнес-логіку і дані. MVVM була розроблена для полегшення двостороннього зв'язку даних, на чому і процвітають фреймворки типу AngularJS. Ми зосередимося на шляху MVVM, так як в останні роки Angular схиляється саме туди.

**2.2 Двосторонній зв'язок даних**

Двостороння зв'язок даних — дуже проста концепція, що надає синхронізацію між шарами Моделі і Виду. Зміни Моделі передаються у Вигляд, а зміни Виду автоматично відображаються в Моделі. Таким чином, Модель стає актуальним джерелом даних про стан програми.  
Angular використовує прості Об'єкти JavaScript для синхронізації Моделі і Виду, у результаті чого оновлювати будь-який з них легко і приємно. Angular перетворює дані в JSON і краще всього спілкується методом REST. За допомогою такого підходу простіше будувати фронтенд-додатки, тому що все стан додатки зберігається в браузері, а не передається з сервера по шматочках, і немає побоювання, що стан буде зіпсоване або втрачене.  
Пов'язуємо ми ці значення через вираження Angular, які доступні у вигляді керуючих шаблонів. Також ми можемо пов'язувати Моделі через атрибут під назвою ng-model. Angular використовує свої атрибути для різних API, які звертаються до ядра Angular.

**2.3 Ін'єкція залежності (Dependency Injection, DI)**

DI — шаблон розробки програм, який визначає, як компоненти зв'язуються зі своїми залежностями. Ін'єкція — це передача залежності до залежного Об'єкта, і ці залежності часто називають Сервісами.  
У AngularJS ми хитрим чином використовуємо аргументи функції для оголошення потрібних залежностей, а Angular передає їх нам. Якщо ми забудемо передати залежність, але пошлемося на неї там, де вона потрібна нам, Сервіс буде не визначено і в результаті станеться помилка компіляції всередині Angular. Але не хвилюйтеся, angular викидає свої помилки і їх дуже зручно влаштовані для налагодження

**2.3 Додатка на одну сторінку (Single Page Application, SPA), управління станом і Ajax (HTTP)**

У додатку на одну сторінку (SPA) або весь необхідний код (HTML, CSS and JavaScript) викликається за одне завантаження сторінки, або потрібні ресурси динамічно підключаються і додаються до сторінки за необхідності, зазвичай у відповідь на дії користувача. Сторінка не перезавантажується під час роботи, не передає управління іншій сторінці, хоча сучасні технології HTML5 дозволяють одній з додатком працювати на декількох логічних сторінках. Взаємодія з SPA часто відбувається за допомогою фонового спілкування з сервером.  
У більш старих додатках, коли стан програми зберігалося на сервері, траплялися відмінності між тим, що бачить користувач і тим, що зберігалося на сервері. Також відчувалася нестача стану програми в моделі, так як всі дані зберігалися в шаблонах HTML і динамічними не були. Сервер готував статичний темплейт, користувач вводив туди інформацію і браузер відправляв її назад, після чого відбувалася перезавантаження сторінки і бекенд оновлював стан. Будь збережене стан втрачалося, і браузеру потрібно було скачувати всі дані після оновлення сторінок заново.  
Часи змінилися, браузер зберігає стан додаток, складна логіка і фреймворки придбали популярність. AngularJS зберігає стан в браузері і передає зміни при необхідності через Ajax (HTTP) з використанням методом GET, POST, PUT і DELETE. Краса в тому, що сервер може бути незалежний від фротенда, а фронтенд — від сервера. Ті ж самі сервера можуть працювати з мобільними додатками з абсолютно іншим фронтендом. Це дає нам гнучкість, так як на бэкенде ми працюємо з JSON-даними будь-яким зручним нам способом на будь-якому сервері ЯП.

**2.5 Структура програми**

У Angular є різні API, але структура програми зазвичай одна і та ж, тому майже всі програми будуються подібним чином і розробники можуть включатися в проект без зусиль. Також це дає передбачувані API і процеси налагодження, що зменшує час розробки і швидке прототипування. Angular побудований навколо можливості тестування («testability»), щоб бути найбільш простим як у розробці, так і в тестуванні.  
Давайте вивчати.

**3 Модулі**

Всі програми створюються через модулі. Модуль може залежати від інших, або бути поодиноким. Модулі служать контейнерами для різних розділів програми, таким чином роблячи код придатним для повторного використання. Для створення модуля застосовується глобальний Object, простір імен фреймворку, і метод module.

**3.1 Сетери (setters).**

У додатку є один модуль app.

angular.module('app', []);  
  
Другим аргументом йде [] — зазвичай цей масив містить залежності модуля, які нам потрібно підключити. Модулі можуть залежати від інших модулів, які у свою чергу теж можуть мати залежності. У нашому випадку масив порожній.

**3.2 Геттери (Getters)**

Для створення Controllers, Directives, Services і інших можливостей нам треба послатися на існуючий модуль. У синтаксисі є непомітне відмінність — ми не використовуємо другий аргумент.

angular.module('app');

3.3 Робота модулів  
  
Модулі можуть зберігатися і викликатися і через змінну. Ось приклад зберігання модуля в змінної.

var app = angular.module('app', []);

Тепер ми можемо використовувати змінну app для побудови програми.

**3.4 HTML бутстрап**

Для опису того, де додаток знаходиться в DOM, а зазвичай це елемент

<html>

, нам треба зв'язати атрибут ng-app з модулем. Так ми повідомляємо Angular, куди присобачити наш додаток.

<html ng-app="app">  
<head></head>  
<body></body>  
</html>

Якщо ми вантажимо файли з JavaScript асинхронно, нам треба присобачити додаток вручну через angular.bootstrap(document.documentElement, ['app']);.

**4 Розбираємося з $scope**

Одне з основних понять програмування — область видимості. У Angular область видимості — це один з головних об'єктів, який робить можливим цикли двостороннього зв'язку даних і зберігає стан програми. $scope — досить хитрий об'єкт, який не тільки має доступ до даних і значенням, але і надає ці дані в DOM, коли Angular рендерить наш додаток.  
Уявіть, що $scope — це автоматичний міст між JavaScript і DOM, зберігає синхронізовані дані. Це дозволяє простіше працювати з шаблонами, коли ми використовуємо при цьому синтакс HTML, а Angular рендерить відповідні значення $scope. Це створює зв'язок між JavaScript і DOM. Загалом, $scope грає роль ViewModel.  
$scope використовується тільки всередині Контролерів. Там ми прив'язуємо дані Контролера до Виду. Ось приклад того, як ми оголошуємо дані Контролера:

$scope.someValue = 'Hello';

Щоб це відобразилося в DOM, ми повинні приєднати Контролер до HTML і повідомити Angular, куди вставляти значення.

<div ng-controller="AppCtrl">  
{{ someValue }}  
</div>

Перед вами концепція області видимості Angular, що підкоряється деяким правилам JavaScript в плані лексичних областей видимості. Зовні елемента, до якого приєднаний Контролер, дані знаходяться поза зоною видимості — так само, як мінлива вийшла б за область видимості, якщо б ми послалися на неї зовні її області видимості.  
Ми можемо прив'язати будь-які типи JavaScript $scope. Таким чином ми беремо дані від сервісу, спілкується з сервером, і передаємо їх в View, шар презентації.

Чим більше ми створимо Контролерів та зв'язків з даними, тим більше з'являється областей видимості. Розібратися в їх ієрархії нічого не варто — тут нам допоможе змінна $rootScope

#### 4.1 $rootScope

$rootScope не сильно відрізняється від $scope, просто це об'єкт $scope самого верхнього рівня, від якого походять всі інші області видимості. Коли Angular починає створення вашого додаток, він створює об'єкт $rootScope, і все прив'язки і логіка програми створюють об'єкти $scope, які є спадкоємцями $rootScope.  
  
Зазвичай ми не використовуємо $rootScope, але з його допомогою можна забезпечити передачу даних між різними областями видимості.

#### 5 Контролери

Контролер дозволяє взаємодіяти Увазі і Моделі. Це те місце, де логіка презентації синхронізує інтерфейс з моделлю. Мета Контролера — приводити в дію зміни до Моделі і Вигляді. У Контролері Angular зводить разом бізнес-логіку і логіку презентації.  
Ми вже торкалися Контролерів, оголосивши атрибут ng-controller для показу даних $scope. Цей атрибут пов'язує область видимості і екземпляр Контролера, і забезпечує доступ до даних і методів Контролера з DOM.  
Перед використанням Контролера його треба створити. Скористаємося вже пройденим матеріалом:

angular  
.module('app', [])  
.controller('MainCtrl', function () {  
});

Контролер приймає два аргументи — ім'я, за яким на нього можна посилатися, і функцію зворотного виклику. При цьому насправді це буде функція, що описує тіло Контролера.  
Щоб методи Angular не виглядали, як зворотні дзвінки, я розміщую функції поза синтаксису Angular.

function MainCtrl () {  
  
}  
  
angular  
.module('app', [])  
.controller('MainCtrl', MainCtrl);

Так воно виглядає чистіше і читається легше. Я назвав функцію MainCtrl, тому що іменовані функції легше налагоджувати.  
Наступні приклади увазі, що модуль був створений.

##### 5.1 Методи і логіка презентацій

Мета Контролера — інтерпретувати бізнес-логіку моделі і перетворювати її в формат презентації. У Angular це можна робити по-різному, в залежності від того, які ми отримуємо дані.  
Контролер спілкується з Сервісом, і передає дані у тому, чи зміненому форматі в наш Вид через об'єкт $scope. Коли оновлений Вигляд, логіка Контролера також оновлюється, і її можна передавати назад на сервер через Сервіс. Але спочатку ми створимо кілька об'єктів у Контролері і зв'яжемо їх з $scope, щоб розібратися, як працює Контролер. Бізнес-логіку ми не розглядатимемо, для стислості ми додамо її нижче і отримаємо дані від Сервісу.

function MainCtrl ($scope) {  
$scope.items = [{  
name: 'Набір водолаза',  
id: 7297510  
},{  
name: 'Шноркель',  
id: 0278916  
},{  
name: 'Гідрокостюм',  
id: 2389017  
},{  
name: 'Рушник',  
id: 1000983  
}];  
}  
  
angular  
.module('app')  
.controller('MainCtrl', MainCtrl);

За допомогою $scope ми прив'язуємо масив. Після цього він буде доступний в DOM, де його можна передати однією з вбудованих Директив Angular, ng-repeat, щоб у циклі пройтися за даними і створити структуру в DOM на основі шаблону і даних.

<div ng-controller="MainCtrl">  
<ul>  
<li ng-repeat="item in items">  
{{ item.name }}  
</li>  
</ul>  
</div>

##### 5.2 Новий синтакс «controllerAs»

Контролери схожі на класи, але використання їх через об'єкти $scope не схоже на використання класів. Вони пропонували використовувати ключове слово this замість $scope. Розробники Angular ввели таку можливість у рамках синтакса controllerAs, де Контролер оформляється у вигляді примірника, що зберігається у змінній — приблизно так само, як використання new із змінною для створення нового об'єкта.  
Ми опускаємо виклик $scope і використовуємо this.

function MainCtrl () {  
this.items = [{  
name: 'Набір водолаза',  
id: 7297510  
},{  
name: 'Шноркель',  
id: 0278916  
},{  
name: 'Гідрокостюм',  
id: 2389017  
},{  
name: 'Рушник',  
id: 1000983  
}];  
}  
  
angular  
.module('app')  
.controller('MainCtrl', MainCtrl);

Потім ми додаємо «as» там, де нам потрібно створити екземпляр Контролера в DOM.  
Запис MainCtrl as main означає, що всі дані знаходяться у змінній main, тому items з минулого прикладу перетворюються в main.items.

<div ng-controller="MainCtrl as main">  
<ul>  
<li ng-repeat="item in main.items">  
{{ item.name }}  
</li>  
</ul>  
</div>

Так зрозуміліше, до якого Контролеру належить властивість, коли у нас є багато Контролерів або вкладені Контролери. До того ж, це допомагає уникнути конфлікт імен.  
У кожного створюваного $scope є об'єкт $parent. Без використання controllerAs нам треба б було використовувати посилання на $parent для будь-яких методів з області видимості $parent, і $parent.$parent для області на рівень вище, і так далі. Тепер же ми можемо просто використовувати ім'я змінної.

#### 6 Сервіси і Фабрики

Сервіси дозволяють зберігати дані Моделі та бізнес-логіку, наприклад, спілкування з сервером через HTTP. Часто плутають Сервіси і Фабрики — відмінність їх у тому, як створюються відповідні об'єкти.  
Важливо пам'ятати, що всі Сервіси — синглтоны, на кожну ін'єкцію є тільки один Сервіс. За угодою імена Сервісів даються в стилі паскаля, тобто «my service» пишеться як «MyService».   
6.1 Метод service  
Метод створює новий об'єкт, який спілкується з бекендом і надає інструменти для роботи з бізнес-логікою. Сервіс — це об'єкт constructor, який викликають через ключове слово new, у зв'язку з чим наша логіка зв'язується з сервісом за допомогою ключового слова this. Сервіс створює об'єкт-одинак.

function UserService () {  
this.sayHello = function (name) {  
return 'Привіт тобі ' + name;  
};  
}  
  
angular  
.module('app')  
.service('UserService', UserService);

Тепер можна вставляти Сервіс в Контролер.

function MainCtrl (UserService) {  
this.sayHello = function (name) {  
UserService.sayHello(name);  
};  
}  
  
angular  
.module('app')  
.controller('MainCtrl', MainCtrl);

Перед Сервісом можна виконувати код, оскільки всі методи створюються у вигляді об'єктів. У Фабрики все інакше.

##### 6.2 Фабричні методи

Фабричні методи повертають об'єкт або функцію, тому ми можемо використовувати замикання, або повертати об'єкт host, до якого можна прив'язувати методи. Можна створювати приватну і публічну області видимості. Всі Фабрики стають Сервісами, тому ми так і називаємо їх.  
Ми відтворимо UserService з прикладу, для порівняння використовуючи фабричний метод.

function UserService () {  
var UserService = {};  
function greeting (name) {  
return 'Привіт тобі ' + name;  
}  
UserService.sayHello = function (name) {  
return greeting(name);  
};  
return UserService;  
}  
  
angular  
.module('app')  
.factory('UserService', UserService);

За допомогою замикань можна емулювати приватну область видимості. Можна було б створити щось подібне і всередині методу service constructor, але тут видно, що повертається, а що залишається всередині області видимості Сервісу. Так само можна створити приватні допоміжні функції, які залишаються в області видимості, коли функція вже повернулася, при цьому їх можна використовувати з публічних методів. Всередині Контролера використовуються такі Сервіси точно так само.

function MainCtrl (UserService) {  
this.sayHello = function (name) {  
UserService.sayHello(name);  
};  
}  
  
angular  
.module('app')  
.controller('MainCtrl', MainCtrl);

Сервіси зазвичай використовують не для логіки презентацій, а для шару бізнес-логіки. Це спілкування з бекендом через REST з Ajax.

#### 7 Використання шаблонів через ядро Angular

Поки ми розглядали программисткую бік Angular, але не показували, як же його можна використовувати з HTML. Наступний крок — використовувати потужні можливості шаблонізації.

##### 7.1 Вираження

Вирази Angular — це схожі на JavaScript фрагменти, які можна використовувати шаблони, щоб проводити умовні зміни в DOM — як в елементах, так і їх властивості, а також у тексті. Вони живуть всередині посилань {{}} і виконуються в межах $scope. Там немає циклів або if/else, і ми не можемо викидати виключення. Можна робити тільки дрібні операції або викликати значення властивостей $scope.  
Наприклад, {{ value }} — це вираз. В виразах можна використовувати логічні оператори зразок||&&, або тернарний оператор value? true: false.  
З їх допомогою можна створювати гнучкі шаблони і оновлювати змінні без перезавантаження сторінки.

function MainCtrl () {  
this.items = [{  
name: 'Набір водолаза',  
id: 7297510  
},{  
name: 'Шноркель',  
id: 0278916  
},{  
name: 'Гідрокостюм',  
id: 2389017  
},{  
name: 'Рушник',  
id: 1000983  
}];  
}  
  
angular  
.module('app')  
.controller('MainCtrl', MainCtrl);

Можна отримати довжину масиву, оголосивши вираз, який використовує властивість length.

<div ng-controller="MainCtrl as main">  
{{ main.items.length }} шт. у наявності  
</div>

Angular замінить length на значення, і ми побачимо «4 шт. в наявності»

##### 7.2 Використання основних директив

Після такого використання значення автоматично зв'язуються всередині Angular, тому будь-які оновлення впливають на значення виразів. Якщо ми приберемо з масиву один предмет, зовнішній вигляд автоматично оновиться до «3 шт. в наявності». Ніякої метушні, ніяких зайвих зворотних викликів.  
Є ціла купа директив з префіксом ng-\*. Почнемо з ng-click і прив'яжемо функцію до нового шматочку HTML-темплейта. У прикладі я проходжу по масиву і показую кількість товару в наявності.

<div ng-controller="MainCtrl as main">  
<div>  
{{ main.items.length }} шт. у наявності  
</div>  
<ul>  
<li ng-repeat="item in main.items" ng-click="main.removeFromStock(item, $index)">  
{{ item.name }}  
</li>  
</ul>  
</div>

Атрибут ng-click пов'язується з функцією main.removeFromStock(). Я передаю їй товар, який в даний момент ми обробляємо в циклі. Властивість $index корисно для видалення елементів з масиву — нам не потрібно вручну підраховувати індекс поточного елемента.  
Тепер функцію можна додати до Контролера. У функцію передаються $index й елемент масиву, і метод робить наступне:

function MainCtrl () {  
this.removeFromStock = function (item, index) {  
this.items.splice(index, 1);  
};  
this.items = [...];  
}  
  
angular  
.module('app')  
.controller('MainCtrl', MainCtrl);

При створенні методів треба враховувати, що значення this може різнитися, в залежності від використання і контексту виконання. Зазвичай я створюю посилання на Контролер

var vm = this;  
  
де vm - це ViewModel. При такому підході посилання не загубляться. Перероблений контролер виглядає так:  
  
function MainCtrl () {  
var vm = this;  
vm.removeFromStock = function (item, index) {  
vm.items.splice(index, 1);  
};  
vm.items = [...];  
}  
  
angular  
.module('app')  
.controller('MainCtrl', MainCtrl);

Так і створюється логіка презентації, що працює з інтерфейсом. Пропущено лише один крок — оновлення Моделі. Перед видаленням файлів з vm.items ми повинні відправити запит DELETE на бекенд, і після успішного відповіді видалити його з масиву. Тоді DOM оновиться лише в разі успішної обробки запиту і не введе користувача в оману.   
Хоча движок шаблонів досить потужний, завжди потрібно пам'ятати про можливості повторного використання коду в майбутніх програмах. Для цього добре підходять Директиви.

#### 8 Директиви

Є два види Директив — одні працюють зі зв'язками всередині Angular, а інші ми створюємо самі. Директива може робити, що завгодно — надавати логіку для заданого елемента, або сама бути елементом і надавати шаблон з логікою всередині себе. Ідея Директив у розширенні можливостей HTML.  
Подивимося на вбудовані директиви Angular, потім спробуємо створити свої.

##### 8.1 ng-repeat

Ми вже зустрічалися з нею, тому я дам тільки простий приклад.

<ul>  
<li ng-repeat="item in main.items">  
{{ item }}  
</li>  
</ul>

ng-repeat копіює елемент і відтворює його, заповнюючи при цьому даними об'єктів з масиву. Якщо з масиву видалити елемент, DOM оновиться автоматично.

##### 8.2 ng-model

Використовується для ініціалізації нової неіснуючої моделі, або для прив'язки існуючої.  
Повідомлення: {{ main.message }}  
Якщо у властивості $scope main.message міститься значення, воно буде передано в input. Якщо в $scope немає такого значення, воно просто буде проинициализировано. Ми можемо передавати ці значення в інші директиви, наприклад в ng-click.

##### 8.3 ng-click

Краса його в тому, що нам не треба самим навішувати обробники подій. Angular сам підрахує значення виразу всередині директиви і повісить обробник на подію. На відміну від onClick=", директива ng-click належить до своєї області видимості, тобто не є глобальною.

<input type="text" ng-model="main.message">  
<a href=" ng-click="main.showMessage(main.message);">Показати повідомлення</a>

Тут я передаю main.message в метод main.showMessage, а там Angular обробляє його як простий об'єкт JavaScript. В цьому полягає принадність Angular — всі зв'язки даних у DOM є об'єктами, ми можемо просто парсити їх, маніпулювати ними, перетворювати в JSON і відправляти на бекенд.

##### 8.4 ng-href/ng-src

Щоб Angular сам дбав про особливості роботи браузерів з параметрами href і src, замість них ми використовуємо ng-href=" and ng-src=".

<a ng-href="{{ main.someValue }}">Go</a>  
<img ng-src="{{ main.anotherValue}}" alt=">

##### 8.5 ng-class

Ця Директива виглядає як опис властивостей і значень об'єкта. Замість традиційних викликів elem.addClass(className) і elem.removeClass(className), Angular додає і видаляє класи на підставі заданих виразів.

<div class="notification" ng-class="{  
warning: main.response == 'error',  
ok: main.response == 'success'  
}">  
{{ main.responseMsg }}  
</div>

Angular з'ясовує значення main.response і в залежності від нього додає клас warning або ok.

##### 8.6 ng-show/ng-hide

Ці Директиви часто зустрічаються при використанні Angular. Це зручний спосіб показувати або ховати в залежності від значення властивості.  
Для перемикання видимості елементу ми використовуємо ng-click.

<a href=" ng-click="showMenu = !showMenu">Переключити меню!</a>  
<ul ng-show="showMenu">  
<li>1</li>  
<li>2</li>  
<li>3</li>  
</ul>

Різниця тільки в тому, що ng-show або ng-hide визначають, повинен елемент спочатку бути видимим або бути захованим.

##### 8.7 ng-if

ng-if не просто ховає елементи, а видаляє їх. Якщо елемент потрібно відтворити після зміни значення властивості, для нього створюється новий $scope. Це також позитивно впливає на швидкодію фреймворка.

<div ng-if="main.userExists">  
Введіть логін  
</div>

##### 8.8 ng-switch

Директива, схожа з оператором case / switch в програмуванні, або просунутий ng-if. В залежності від значення з $scope вибирається один з декількох елементів.

<div ng-switch on="main.user.access">  
<div ng-switch-when="admin">  
<!-- code for admins -->  
</div>  
<div ng-switch-when="user">  
<!-- code for users -->  
</div>  
<div ng-switch-when="author">  
<!-- code for authors -->  
</div>  
</div>

##### 8.9 ng-bind

Значення можна вставляти в DOM через синтаксис {{ value }}, але є ще один варіант, ng-bind. Відмінності синтаксису видно на прикладі.

<p>{{ main.name }}</p>  
<p ng-bind="main.name"></p>

ng-bind можна використовувати, щоб уникнути миготіння при завантаженні сторінки. Angular автоматично приховує контент, який підвантажується через Директиви. При використанні фігурних дужок вони можуть бути видні в тексті документа. При використанні ng-bind вставки не видно, поки Angular не вирахує потрібні значення.

##### 8.10 ng-view

Односторінкові програми використовують одну сторінку без перезавантажень, вміст якої оновлюється автоматично. Це досягається використанням атрибута ng-view з порожнім елементом зразок

<div></div>

, який служить контейнером для будь-якого динамічно вставляється коду, одержуваного через XMLHttpRequest.  
На місце ng-view вставляються різні Види, залежно від шляху в URL. Наприклад, можна сказати Angular, щоб він вставляв login.html якщо URL-адреса містить myapp.com/#/login і змінювати вміст при зміні URL.

##### 8.11 Розширення HTML

Хоча вбудовані Директиви додають функціональності в HTML, іноді необхідно додати свої функції для подальшого розширення можливостей. Розглянемо API для створення своїх директив.

##### 9 Настроювані Директиви

Настроювані Директиви — одна з найбільш складних концепцій в API Angular, тому що вони не схожі на звичні програмні концепції. Вони виступають у ролі способу Angular реалізувати концепції Веб найближчого майбутнього — настроювані елементи, тіньовий DOM, шаблони та імпорт HTML. Давайте поступово вивчимо Директиви, розбивши їх на шари.

##### 9.1 Настроювані

НЕ використовуються у випадках повторного використання коду або шаблонного коду, коли ми можемо оголосити один елемент, а код, пов'язаний з ним, буде автоматично приєднаний до нього.  
У Angular є чотири способи використання Директив — Настроювані елементи, що Настроюються атрибути, імена класів і коментарі. Останніх двох я намагаюся уникати, тому що в них легко заплутатися, і до того ж, у коментарів є проблеми з IE. Самий безпечний і кросбраузерність спосіб — Настроювані атрибути. Давайте розглянемо ці способи в наступному порядку: Element, Attribute, Class, Comment.

<my-element></my-element>  
  
<div my-element></div>  
  
<div class="my-element"></div>  
  
<!-- directive: my-element -->

У Директив є властивість restrict, через яке можна обмежити їх використання одним із цих способів. За замовчуванням, Директиви використовуються через 'EA', що означає Element, Attribute. Інші варіанти — C для класів і М для коментарів.

##### 9.2 Тіньової DOM

Тіньовий DOM працює так, що всередині певних частин звичайного DOM документа містяться вкладені документи. Вони підтримують HTML, CSS і області видимості JavaScript.  
У Тіньовому DOM можна визначати як чистий HTML, так і контент, який буде імпортовано. Ми можемо помістити текст в Настроюється елемент:

<my-element>  
Всім привіт!  
</my-element>

І текст «привіт Усім!» доступний у Тіньовому DOM.

##### 9.3 Шаблонизация та імпорт HTML

Існує три різних способи використання шаблонів в Директивах. Два майже однакових, а третій зберігає шаблон в рядку.

###### 9.3.1 властивість template

Оголошує потрібний шаблон. Шаблон форматується як рядок, а потім Angular компілює його і вставляє в DOM  
Приклад:

{  
template: '<div>' +   
'<ul>' +  
'<li ng-repeat="item in vm.items">' +  
'{{ item }}' +  
'</li>' +  
'</ul>' +  
'</div>'  
}

При використанні такого підходу зручно вписувати логіку на JavaScript між рядками. Я використовую конструкцію з [].join("), вона робить текст більш читабельним.

{  
template: [  
'<div>',  
'<ul>',  
'<li ng-repeat="item in vm.items">',  
'{{ item }}',  
'</li>',  
'</ul>',  
'</div>'  
].join(")  
}

###### 9.3.2 властивість templateUrl

Властивість templateUrl дозволяє вказувати на зовнішній ресурс або на елемент

<script>

, які містять потрібний шаблон.  
Якщо задати:

{  
templateUrl: 'items.html'  
}

Angular спочатку пошукає в DOM елемент

<script>

з відповідним id, а якщо не знайде, тоді запросить документ через HTTP GET.

<script type="text/ng-template" id="/hello.html">  
<div>  
<ul>  
<li ng-repeat="item in vm.items">  
{{ item }}  
</li>  
</ul>  
</div>  
</script>

Спочатку ми задаємо тип шаблону як text/ng-template, щоб браузер не інтерпретував його текст, як JavaScript. Таким чином можна включати текст шаблонів у файл, а не завантажувати його окремими файлами. Також можна використовувати властивість template, коли шаблон зберігається в рядку. Коли шаблон завантажується, Angular зберігає його, і потім його можна використовувати в ng-include та ng-view. В іншому випадку Angular здійснить GET-запит файлу з шаблоном.  
Всі завантажені шаблони зберігаються в $templateCache протягом усього життя додатки.

##### 9.4 API Директив

Давайте детальніше розглянемо API Директив, щоб можна було створювати свої директиви. Перша інструкція — return, яка повертає об'єкт. Це все, що потрібно для створення директиви.

function someDirective () {  
return {  
  
};  
}  
  
angular  
.module('app')  
.controller('someDirective', someDirective);

Подивимося на найбільш популярні властивості об'єктів директив.

function someDirective () {  
return {  
restrict: 'EA',  
replace: true,  
scope: true,  
controllerAs: 'something',  
controller: function () {  
  
},  
link: function ($scope, $element, $attrs) {  
  
},  
template: [  
'<div class="some-directive">',  
'My directive!',  
'</div>'  
].join(")  
};  
}  
  
angular  
.module('app')  
.controller('someDirective', someDirective);

###### 9.4.1 restrict

Ми вже згадували restrict — вона дозволяє обмежувати використання директиви. Якщо ми хочемо ставити директиву тільки через атрибути, можна обмежити їх 'A'. Для обмеження роботи директив в якості елементів використовується 'E', коментарів — 'M' і імен класів — 'C'.

###### 9.4.2 replace

Замінює оригінальний елемент директиви. Якщо ми використовуємо

<some-directive></some-directive>

і задаємо replace: true, після створення сторінки початковий елемент буде замінено результатом роботи скрипта.

###### 9.4.3 scope

Дозволяє успадковувати $scope поточного або батьківського контексту, в який входить директива. Можна створити автономний $scope і передавати певні значення, зазвичай через настроюються атрибути.

###### 9.4.4 controllerAs

Ми вже пробували це властивість, яка визначає ім'я контролера всередині директиви. Якщо ми задаємо controllerAs: 'something', то всі посилання на властивості контролера будуть виглядати як something.myMethod()

###### 9.4.5 controller

Захопити існуючий контролер або створити новий. Якщо MainCtrl вже існує, можна визначити його як controller: 'MainCtrl'. Для збереження інкапсуляції ми просто оголошуємо новий контролер кожен раз через controller: function () {}. Функція зворотного виклику контролера повинна обробляти зміни в ViewModel і спілкуватися з Сервісами.

###### 9.4.6 link

Функція link викликається після того, як елемент компілюється і вставляється в DOM, тому тут можна зробити що-то з контентом після компіляції або щось, не пов'язане з Angular.  
У контролері ми не маніпулюємо DOM, але це можливо у функції link. Вона також може вставляти $scope, кореневий елемент шаблон $element і об'єкт $attrs, що містить всі властивості елемента DOM, що відбиває поточний стан {{ }}. Всередині link можна прив'язати обробники подій, визначити плагіни і навіть вставити сервіси Angular.

##### 9.5 Створення директив

Пройдемося за прикладом створення директиви, яка дозволяє вставляти компонент «email», у якого є поля To Subject та Message.  
Ми створюємо елемент

<compose-email>

, на місце якого відбувається вставка вмісту. Його можна вставити в різні місця DOM багаторазово, і скрізь він буде замінений на наш компонент, кожен з яких буде окремим екземпляром. Почнемо з шаблонів.

function composeEmail () {  
return {  
restrict: 'EA',  
replace: true,  
scope: true,  
controllerAs: 'compose',  
controller: function () {  
  
},  
link: function ($scope, $element, $attrs) {  
  
},  
template: [  
'<div class="compose-email">',  
'<input type="text" placeholder="To..." ng-model="compose.to">',  
'<input type="text" placeholder="Subject..." ng-model="compose.subject">',  
'<textarea placeholder="Message..." ng-model="compose.message"></textarea>',  
'</div>'  
].join(")  
};  
}  
  
angular  
.module('app')  
.controller('composeEmail', composeEmail);

Тепер ми можемо використовувати директиву composeEmail в декількох місцях, без необхідності копіювати шматки HTML. Пам'ятайте, що Angular парсити назва директиви так, що composeEmail перетворюється в

<compose-email></compose-email>

#### 10 Фільтри

Фільтри обробляють інформацію і видають певні набори даних, ґрунтуючись на будь-якій логіці. Це може бути що завгодно, від форматування дати в читаний формат, до списку імен, які починаються на задану букву. Подивимося на популярні вбудовані фільтри.  
Їх можна використовувати або DOM через символ вертикальної риси | в виразах, які парсити Angular, або через сервіс $filter, який можна використовувати у JavaScript коді замість HTML.  
HTML синтакс:

{{ filter\_expression | filter : expression : comparator }}

JavaScript синтакс:

$filter('filter')(array, expression, comparator);

Частіше використовується більш простий варіант з HTML. Ось деякі з фільтрів.

##### 10.1 Фільтри дати

З датами працювати клопітно, але Angular полегшує це завдання. Прив'яжемо фільтр до дати в мілісекундах до $scope.timeNow = new Date().getTime():

<p>  
Сьогодні у нас: {{ timeNow | date:'dd-MM-yyyy' }}  
</p>

##### 10.2 Фільтр JSON

Вбудовані фільтри JSON перетворять об'єкт JavaScript в рядок JSON. Це зручно для виводу значень Моделі DOM при розробці. Для гарного форматування оберніть фільтри JSON в теги

<pre>

<pre>  
{{ myObject | json }}  
</pre>

##### 10.3 limitTo і orderBy

Описані фільтри працюють просто — отримують значення і видають значення. Як же працювати з наборами даних?  
limitTo обмежує кількість даних, переданих у Вид — його зручно використовувати всередині ng-repeat.

<ul>  
<li ng-repeat="user in users | limitTo:10">  
{{ user.name }}  
</li>  
</ul>

Фрагмент виведе не більш 10 користувачів. Зверніть увагу на використання limitTo всередині ng-repeat.  
orderBy задає порядок виведення масивів, сортуючи за однією з властивостей об'єктів. Якщо у нас об'єкт користувача виглядає так:

{  
name: 'Todd Motto',  
}

ми можемо вивести лист за алфавітом наступним чином:

<ul>  
<li ng-repeat=" user in users | orderBy:'name' ">  
{{ user.name }}  
</li>  
</ul>

Справжні можливості фільтрів розкриваються при створенні своїх власних.

#### 11 Власні фільтри

Ми все робили фільтри в об'єктах і масивах. Для використання їх в Angular існує API. Ми маємо двосторонній зв'язок даних без всякого праці. Всі задані нами фільтри будуть викликані в циклі $digest.

##### 11.1 Фільтри одного значення

Приймають одне значення, і видають відфільтрований контент. Для створення використовується метод .filter(). Всі такі фільтри глобально доступні в будь-якій області видимості.  
Ось заготівля фільтра, від якої можна відштовхуватися.

function myFilter () {  
return function () {  
// повернення результату  
};  
}  
angular  
.module('app')  
.filter('myFilter', myFilter);

Всі аргументи для фільтрів передаються всередину автоматично. Для прикладу зробимо фільтр форматування тексту в нижній регістр (хоча у Angular є такий вбудований фільтр).

function toLowercase () {  
return function (item) {  
return item.toLowerCase();  
};  
}  
angular  
.module('app')  
.filter('toLowercase', toLowercase);

item передається фільтру як локальна змінна. Використовується фільтр точно так само, як вбудовані:

<p>{{ user.name | toLowercase }}</p>

##### 11.2 Фільтри наборів даних

Іноді треба обробити набір даних і повернути їх перетвореними. Давайте зробимо фільтр, що відбирає імена по заданій першій букві. Наприклад, ось фільтр слів, що починаються з 'A'.

function namesStartingWithA () {  
return function (items) {  
return items.filter(function (item) {  
return /$a/i.test(item.name);  
});  
};  
}  
angular  
.module('app')  
.filter('namesStartingWithA', namesStartingWithA);

Використовуємо його всередині ng-repeat:

<ul>  
<li ng-repeat="item in items | namesStartingWithA">  
{{ item }}  
</li>  
</ul>

Передавати аргументи на фільтри можна через двокрапку:

<ul>  
<li ng-repeat="item in items | namesStartingWithA:something">  
{{ item }}  
</li>  
</ul>

something автоматично передається у функцію фільтра:

function namesStartingWithA () {  
return function (items, something) {  
// є доступ до "items", і до "something"  
};  
}  
angular  
.module('app')  
.filter('namesStartingWithA', namesStartingWithA);

##### 11.3 Фільтри контролерів

Фільтри можна створювати поза методу .filter(), просто передаючи функцію в контролер, яка буде виконувати роль фільтра. Продовжуючи попередній приклад, ми можемо створити функцію контролера this.namesStartingWithA, тоді фільтр буде доступний тільки з цього контролера, а не глобально. У цьому випадку використовується синтакс controllerAs.

function SomeCtrl () {  
this.namesStartingWithA = function () {  
  
};  
}  
angular  
.module('app')  
.controller('SomeCtrl', SomeCtrl);

Синтакс виклику фільтра в DOM злегка відрізняється:

<ul>  
<li ng-repeat="item in vm.items | filter:namesStartingWithA">  
{{ item }}  
</li>  
</ul>

#### 12 Динамічний роутинг через $routeProvider

Ми розбираємося з різними концепціями Angular, але поки не дійшли до головної: як же Angular допомагає створити одностраничное додаток. А для цього використовується роутер, який ми налаштовуємо так, щоб він змінював стан нашого додатки залежно від URL. Потрібний нам роутер називається ngRoute

angular  
.module('app', [  
'ngRoute'  
]);

Підключивши таким чином потрібний модуль, ми можемо налаштувати шляхи, вставляючи $routeProvider і налаштовуючи його в методі .config(). Після цього ми отримуємо доступ до методу .when().   
Уявімо, що у мене є додаток для читання пошти — по натисненню на «вхідні» /inbox потрібно, щоб воно показувало список листів. Тоді перший аргумент .when() рядок, що описує потрібний URL. Другий аргумент — об'єкт з додатковими налаштуваннями. Є ще метод .otherwise(), який обробляє випадки неіснуючих шляхів.

function router ($routeProvider) {  
$routeProvider  
.when('/inbox', {})  
.otherwise({  
redirectTo: '/inbox'  
});  
}  
  
angular  
.module('app')  
.config(router);

Для налаштування шляху спочатку необхідно вибрати шаблон, який буде використовуватися з потрібним Видом. Припустимо, ми використовуємо шаблон inbox.html.

$routeProvider  
.when('/inbox', {  
templateUrl: 'views/inbox.html'  
})  
.otherwise({  
redirectTo: '/inbox'  
});

Всіма Видами необхідний Контролер. У Angular для цього є властивості controller і controllerAs.

$routeProvider  
.when('/inbox', {  
templateUrl: 'views/inbox.html',  
controller: 'InboxCtrl',  
controllerAs: 'inbox'  
})  
.otherwise({  
redirectTo: '/inbox'  
});

Тепер припустимо, ми хочемо налаштувати ще один Вид — коли ми клацаємо по вхідному письма, ми повинні побачити саме лист. Для цього необхідний динамічний роутинг, оскільки у різних листів повинні бути різні URL (містять id тощо). Для передачі даних динамічного роутінга використовується двокрапка перед назвою динамічної групи. Наприклад, у випадку листи з id 173921938 шлях буде /inbox/email/173921938, а опис цього шляху — '/inbox/email/:id'.  
Коли додаток отримує шлях /inbox/email/173921938, Angular завантажує ті ж самі шаблон і контролер, що і для /inbox/email/902827312.  
Підсумковий приклад виглядає так:

function router ($routeProvider) {  
$routeProvider  
.when('/inbox', {  
templateUrl: 'views/inbox.html',  
controller: 'InboxCtrl',  
controllerAs: 'inbox'  
})  
.when('/inbox/email/:id', {  
templateUrl: 'views/email.html',  
controller: 'EmailCtrl',  
controllerAs: 'email'  
})  
.otherwise({  
redirectTo: '/inbox'  
});  
});  
  
angular  
.module('app')  
.config(router);

Це основа для програми, що використовує шляху. Але потрібно ще вказати, куди на сторінці необхідно вставляти оброблений шаблон. Для цього використовується ng-view. Зазвичай достатньо написати щось на кшталт:

<div ng-view></div>

Всі зміни URL обробляються Angular, який визначає, чи не потрібно вставляти в сторінку інший шаблон. Ось і все, що потрібно для роботи програми.

##### 12.1 $routeParams

Сервіс $routeParams автоматично парсити URL і виділяє з нього набір параметрів, який перетворює в об'єкт. Наприклад, в останньому прикладі буде витягнутий id листа на підставі завдання динамічного роутінга через :id  
Простий приклад контролера, передавального параметри з $routeParams:

function EmailCtrl ($routeParams, EmailService) {  
// $routeParams { id: 20999851 }  
EmailService  
.get($routeParams.id) // передати об'єкт  
.success(function (response) {})  
.error(function (reason) {});  
}  
angular  
.module('app')  
.('EmailCtrl', EmailCtrl);

#### 13 Перевірка форм

Перевірка форм здійснює кілька дій, від перевірки змін Моделі та звірки їх з існуючими правилами зв'язки, до зміни DOM для зворотного зв'язку з користувачем.  
Для створення такої форми необхідно задати їй ім'я, яке визначає область видимості форми.

<form name="myForm"></form>

Angular розпізнає таку форму і буде перевіряти і відстежувати введення користувача, наприклад, заповнив той обов'язкові поля форми, і т. п.

##### 13.1 HTML5

В HTML5 був доданий атрибут pattern, який дозволяв браузеру перевіряти enter згідно із заданою регуляркой. Angular включає такі можливості. Крім цього, Angular переробив перевірку полів, позначених як required у своїй директиві під назвою ng-required, яка теж постійно відстежує стан Моделі. Давайте подивимося ще на кілька можливостей Angular.

##### 13.2 $pristine

Після початкового створення сторінки Angular додає до форми властивість $pristine — це означає, що користувач до неї ще не торкався. Angular додасть клас ng-pristine до тих елементів, які ще не змінювалися.

##### 13.4 $dirty

Протилежність pristine — це елементи, які користувач змінив. До них додаються класи ng-dirty, а класи ng-pristine видаляються. Форма не може повернутися до стану $pristine без перезавантаження сторінки.

##### 13.5 $valid

Кожне поле може бути оголошено $valid. Наприклад, якщо у поля є атрибут ng-required, а користувач заповнив його, то йому присвоюється клас ng-valid

##### 13.6 $invalid

Протилежність valid. За замовчуванням, всі форми знаходяться у стані $invalid — їм присвоєно клас ng-invalid. Переходи між цими двома станами відбуваються по мірі того, як користувач вводить інформацію.

##### 13.7 Перевірка на основі моделі

В деяких випадках потрібно відключати або включати поля введення або кнопки — наприклад, поки що користувач не введе. Простий приклад перемикання станів кнопки, заснований на даних Моделі. Якщо користувач не ввів ім'я, форму можна оновлювати.

<input type="text" ng-model="user.name" placeholder="Введіть назву">  
<button ng-disabled="!user.name.length">  
Оновити ім'я  
</button>

Якщо user.name.length-повертає true, кнопка включається. І цей стан постійно відстежується весь час, поки відбувається робота з формою.

#### 14 Спілкування з сервером через $http і $resource

Для зв'язку з сервером Angular є два API, $http і $resource. Це високорівневі засоби зв'язку з сервером, які також запускають цикли $digest, підтримуючи наші дані в актуальному стані.

##### 14.1 $http

Якщо ви користувалися методом jQuery $.ajax, то вам все відразу буде зрозуміло. Метод $http можна використовувати як функцію і як об'єкт. Ми можемо передати йому об'єкт з налаштуваннями $http({...}), або ж використовувати доступні методи подібне $http.get(...). Метод $http базується на API обіцянок, яке надається вбудованим сервісом $q.  
Мені більше подобаються методи з короткої записом, тому для демонстрації роботи я буду використовувати саме їх. Ось простий HTTP-запит GET:

$http.get('/url')  
.success(function (data, status, headers, config) {  
  
})  
.error(function (data, status, headers, config) {  
  
});

Успішні і неуспішні зворотні дзвінки відбуваються асинхронно, і у відповіді містяться дані, статус і заголовки. Зазвичай всі відразу вони нам не потрібні, тому можна використати простий підхід:

$http.get('/url')  
.success(function (response) {  
  
})  
.error(function (reason) {  
  
});

Мені подобаються використовувати шаблони обіцянок за допомогою методу .then(), що підтримується в Angular:

$http.get('/url')  
.then(function (response) {  
// успіх  
}, function (reason) {  
// провал  
});

На відміну від бібліотеки jQuery, Angular обертає виклики $http $scope.$apply(), що запускає цикл запиту даних з сервера і оновлює зв'язку.

##### 14.2 $resource

Разом з $http можна використовувати модуль ngResource. У ньому є API $resource, зручне для CRUD-операцій (create, read, update, delete). $resource створює об'єкт, через який можна спілкуватися з джерелами даних по протоколах REST.

function MovieService ($resource) {  
return $resource('/api/movies/:id', { id: '@\_id' },  
{  
update: {  
method: 'PUT'   
}  
}  
);  
}  
angular  
.module('app')  
.factory('MovieService', MovieService);

Можна вставити в залежність просту фабрику Movies в наші контролери і отримувати таким чином всі потрібні дані.

function MovieCtrl (MovieService) {  
var movies = new MovieService();  
// відбулося оновлення фільму  
movies.update(/\* some data \*/);  
}  
angular  
.module('app')  
.controller('MovieCtrl', MovieCtrl);