Angular Master Book

따라 하며 배우는

앵귤러 마스터북

ng-book: The Complete Guide to Angular
by Nate Murray, Felipe Coury, Ari Lerner, and Carlos Taborda
Copyright © 2017 Fullstack.io
All rights reserved.

Korean Translation Copyright © 2018 by J-Pub.
The Korean edition was published by arrangement with Fullstack.io through Agency-One, Seoul.

이 책의 한국어판 저작권은 에이전시 원을 통해 저작권사와의 독점계약으로 제이펍에 있습니다.
신저작권법에 의해 한국 내에서 보호를 받는 저작물이므로 무단전재와 복제를 금합니다.

따라 하며 배우는
앵귤러 마스터북

초판 1쇄 발행 2018년 1월 24일

지은이 네이트 머레이, 필리피 커리, 아리 러너, 칼로스 타보다
옮긴이 배장열
펴낸이 장성두
펴낸곳 제이펍

출판신고 2009년 11월 10일 제406-2009-000087호
주소 경기도 파주시 회동길 159 3층 3-B호
전화 070-8201-9010 / **팩스** 02-6280-0405
홈페이지 www.jpub.kr / **원고투고** jeipub@gmail.com
독자문의 readers.jpub@gmail.com / **교재문의** jeipubmarketer@gmail.com

편집부 이민숙, 황혜나, 이 슬, 이주원 / **소통·기획팀** 민지환 / **회계팀** 김유미
교정·교열 이 슬 / **본문디자인** 한지혜 / **표지디자인** 미디어픽스
용지 에스에이치페이퍼 / **인쇄** 한승인쇄 / **제본** 광우제책사

ISBN 979-11-88621-01-9 (93000)
값 34,000원

※ 이 책은 저작권법에 따라 보호를 받는 저작물이므로 무단 전재와 무단 복제를 금지하며,
이 책 내용의 전부 또는 일부를 이용하려면 반드시 저작권자와 제이펍의 서면동의를 받아야 합니다.
※ 잘못된 책은 구입하신 서점에서 바꾸어 드립니다.

제이펍은 독자 여러분의 아이디어와 원고 투고를 기다리고 있습니다. 책으로 펴내고자 하는 아이디어나 원고가 있으신 분께서는 책의 간단한 개요와 차례, 구성과 저(역)자 약력 등을 메일로 보내주세요. **jeipub@gmail.com**

Angular Master Book

따라 하며 배우는

앵귤러 마스터북

네이트 머레이, 필리피 커리, 아리 러너, 칼로스 타보다 지음 | 배장열 옮김

Jpub 제이펍

※ 드리는 말씀

- 이 책에 기재된 내용을 기반으로 한 운용 결과에 대해 저자, 역자, 소프트웨어 개발자 및 제공자, 제이펍 출판사는 일체의 책임을 지지 않으므로 양해 바랍니다.
- 이 책에 등장하는 각 회사명, 제품명은 일반적으로 각 회사의 등록 상표 또는 상표입니다. 본문 중에는 ™, ©, ® 마크 등이 표시되어 있지 않습니다.
- 이 책에서 사용하고 있는 제품 버전은 독자의 학습 시점이나 환경에 따라 책의 내용과 다를 수 있습니다.
- 이 책의 예제 코드는 학습용으로만 사용할 수 있으며, 상업적으로 이용하거나 온라인에 무단으로 게시할 경우 저작권법에 저촉됨을 알립니다.
- 책 내용과 관련된 문의사항은 옮긴이나 출판사로 연락해 주시기 바랍니다.
  - 옮긴이: justdoit709@gmail.com
  - 출판사: readers.jpub@gmail.com

차례

옮긴이 머리말 xvii
이 책에 대하여 xviii
베타리더 후기 xxiv

CHAPTER 1 첫 앵귤러 웹 애플리케이션 만들기 1

1.1 단순한 레딧 클론 1
1.2 시작하기 4
1.2.1 Node.js와 npm 4
1.2.2 타입스크립트 4
1.2.3 브라우저 5
1.2.4 윈도우 사용자를 위한 특별 과정 5
1.3 앵귤러 CLI 5
1.3.1 예시 프로젝트 6
1.4 애플리케이션 코드 작성하기 10
1.4.1 애플리케이션 실행하기 10
1.5 컴포넌트에 데이터 추가하기 17
1.5.1 템플릿 렌더링하기 19
1.5.2 테스트하기 19
1.6 배열 다루기 20
1.7 UserItemComponent 사용하기 23
1.7.1 UserItemComponent 렌더링하기 23
1.7.2 입력 받기 24
1.7.3 입력값 전달하기 25
1.8 부트스트랩 속성 과정 27
1.8.1 declarations 28
1.8.2 imports 28
1.8.3 providers 28
1.8.4 bootstrap 29
1.9 애플리케이션 확장하기 29
1.9.1 CSS 추가하기 30
1.9.2 애플리케이션 컴포넌트 31
1.9.3 인터랙션 추가하기 32
1.9.4 ArticleComponent 추가하기 36

1.10 여러 행 렌더링하기 44
1.10.1 Article 클래스 만들기 44
1.10.2 여러 Article 저장하기 49
1.10.3 Input으로 ArticleComponent 설정하기 50
1.10.4 Articles 리스트 렌더링하기 52
1.11 새 Article 추가하기 54
1.12 마무리 손길 55
1.12.1 Article 도메인 표시하기 55
1.12.2 점수별로 재정렬하기 56
1.13 배포 57
1.13.1 앱을 제품으로 빌드하기 57
1.13.2 서버로 업로드하기 58
1.14 전체 코드 59
1.15 정리 59

CHAPTER 2 타입스크립트 60

2.1 앵귤러 4, 타입스크립트로 만들어지다 60
2.2 타입스크립트로 무엇을 얻을 수 있나? 62
2.3 타입 62
2.3.1 REPL 경험하기 64
2.4 내장 타입 64
2.4.1 string 64
2.4.2 number 64
2.4.3 boolean 65
2.4.4 Array 65
2.4.5 열거형 65
2.4.6 any 66
2.4.7 void 66
2.5 클래스 66
2.5.1 프로퍼티 67
2.5.2 메서드 67
2.5.3 컨스트럭터 69
2.5.4 상속 71
2.6 유틸리티 72
2.6.1 살찐 화살표 함수 72
2.6.2 템플릿 문자열 74
2.7 정리 75

CHAPTER 3 앵귤러는 어떻게 동작하는가 76

3.1 애플리케이션 77
3.1.1 내비게이션 컴포넌트 78
3.1.2 브레드크럼 컴포넌트 78
3.1.3 제품 리스트 컴포넌트 78
3.2 이 장의 사용법 80
3.3 제품 모델 81
3.4 컴포넌트 82
3.5 컴포넌트 데코레이터 84
3.5.1 컴포넌트 selector 84
3.5.2 컴포넌트 template 84
3.5.3 Product 추가하기 85
3.5.4 템플릿 바인딩으로 Product 보기 87
3.5.5 Product 더 많이 추가하기 87
3.5.6 Product 선택하기 89
3.5.7 <products-list>를 사용하여 제품 리스트 출력하기 89
3.6 ProductsListComponent 컴포넌트 92
3.6.1 ProductsListComponent의 @Component 옵션 설정하기 93
3.6.2 컴포넌트 입력 94
3.6.3 컴포넌트 출력 95
3.6.4 커스텀 이벤트 배출하기 96
3.6.5 ProductsListComponent 컨트롤러 클래스 작성하기 98
3.6.6 ProductsListComponent 뷰 템플릿 작성하기 99
3.6.7 ProductsListComponent 컴포넌트 전체 코드 101
3.7 ProductRowComponent 컴포넌트 103
3.7.1 ProductRowComponent 설정 103
3.7.2 ProductRowComponent 템플릿 104
3.8 ProductImageComponent 컴포넌트 105
3.9 PriceDisplayComponent 컴포넌트 106
3.10 ProductDepartmentComponent 컴포넌트 106
3.11 NgModule과 앱 시동하기 108
3.11.1 앱 시동하기 109
3.12 완성된 프로젝트 110
3.13 앱 배포하기 111
3.14 덧붙여: 데이터 아키텍처 111

CHAPTER 4 내장 지시자 113

4.1 소개 113
4.2 ngIf 114
4.3 ngSwitch 114
4.4 NgStyle 116

4.5 NgClass 118
4.6 NgFor 121
4.6.1 인덱스 얻기 125
4.7 NgNonBindable 126
4.8 결론 127

CHAPTER 5 앵귤러의 폼 128

5.1 폼은 중요하다. 그리고 복잡하다 128
5.2 FormControl과 FormGroup 129
5.2.1 FormControl 129
5.2.2 FormGroup 130
5.3 첫 번째 폼 130
5.3.1 FormsModule 로드하기 131
5.3.2 단순 SKU 폼: @Component 데코레이터 133
5.3.3 단순 SKU 폼: template 133
5.3.4 단순 SKU 폼: 컴포넌트 정의 클래스 136
5.3.5 테스트하기 137
5.4 FormBuilder 사용하기 138
5.5 FormBuilder로 만드는 리액티브 폼 139
5.5.1 FormBuilder 사용하기 139
5.5.2 뷰에서 myForm 사용하기 140
5.5.3 테스트하기 141
5.6 검증 추가하기 143
5.6.1 명시적으로 sku FormControl을 인스턴스 변수로 설정하기 144
5.6.2 맞춤형 검증 149
5.7 변경 내용 주시하기 150
5.8 ngModel 152
5.9 정리 154

CHAPTER 6 의존성 주입 155

6.1 주입 예: PriceService 156
6.2 의존성 주입 요소들 160
6.3 인젝터 다루기 161
6.4 NgModule과 의존성 다루기 164
6.4.1 providers는 키다 166
6.5 제공자 166
6.5.1 클래스 사용하기 166
6.5.2 팩토리 사용하기 170
6.6 앱의 의존성 주입 173
6.7 참고 리소스 174

CHAPTER 7 HTTP 175

7.1 소개 175
7.2 @angular/common/http 사용하기 176
7.2.1 @angular/http에서 가져오기 176
7.3 기본 요청 177
7.3.1 SimpleHttpComponent 컴포넌트 만들기 178
7.3.2 SimpleHttpComponent 템플릿 만들기 178
7.3.3 SimpleHttpComponent 컨트롤러 만들기 179
7.3.4 SimpleHttpComponent 전체 코드 181
7.4 YouTubeSearchComponent 작성하기 181
7.4.1 SearchResult 작성하기 183
7.4.2 YouTubeSearchService 작성하기 184
7.4.3 SearchBoxComponent 작성하기 189
7.4.4 SearchResultComponent 작성하기 196
7.4.5 YouTubeSearchComponent 작성하기 197
7.5 @angular/http API 201
7.5.1 POST 요청 수행하기 201
7.5.2 PUT / PATCH / DELETE / HEAD 201
7.6 맞춤형 HTTP 헤더 202
7.7 정리 202

CHAPTER 8 라우팅 203

8.1 라우팅은 왜 필요할까? 203
8.2 클라이언트 사이드 라우팅의 동작 방식 204
8.2.1 시작: 앵커 태그 사용하기 205
8.2.2 진화: HTML5 클라이언트 사이드 라우팅 206
8.3 첫 라우트 작성하기 206
8.4 앵귤러 라우팅의 구성 요소들 207
8.4.1 imports 207
8.4.2 라우트 설치하기 209
8.4.3 RouterOutlet: <router-outlet> 사용하기 209
8.4.4 RouterLink: [routerLink] 사용하기 211
8.5 모두 합치기 212
8.5.1 컴포넌트 만들기 213
8.5.2 애플리케이션 컴포넌트 215
8.5.3 라우트 설정하기 216
8.6 라우팅 전략 218
8.7 애플리케이션 실행하기 219
8.8 라우트 파라미터 220
8.8.1 ActivatedRoute 220

8.9 음악 검색 앱 221
8.9.1 첫 단계 223
8.9.2 SpotifyService 224
8.9.3 SearchComponent 225
8.9.4 검색 시도하기 232
8.9.5 음악 검색 마무리하기 235
8.10 라우터 훅 236
8.10.1 AuthService 236
8.10.2 LoginComponent 238
8.10.3 ProtectedComponent와 라우트 가드 239
8.11 라우트 중첩 244
8.11.1 라우트 설정하기 244
8.11.2 ProductsModule 245
8.12 정리 249

CHAPTER 9 앵귤러 4의 데이터 아키텍처 250

9.1 데이터 아키텍처 개요 250
9.1.1 앵귤러 4의 데이터 아키텍처 252

CHAPTER 10 옵저버블 데이터 아키텍처 1부 — 서비스 253

10.1 옵저버블과 RxJS 253
10.1.1 노트: 필요한 RxJS 지식 253
10.1.2 반응형 프로그래밍과 RxJS 배우기 254
10.2 대화 앱 개요 255
10.2.1 컴포넌트 256
10.2.2 모델 257
10.2.3 서비스 258
10.2.4 정리 258
10.3 모델 구현하기 258
10.3.1 User 259
10.3.2 Thread 259
10.3.3 Message 260
10.4 UsersService 구현하기 261
10.4.1 currentUser의 스트림 262
10.4.2 새 사용자 설정하기 262
10.4.3 UserService.ts 264
10.5 MessagesService 264
10.5.1 newMessages 스트림 265
10.5.2 messages 스트림 266
10.5.3 Operation 스트림 패턴 266

10.5.4 스트림 공유하기 268
10.5.5 messages 스트림에 Message 추가하기 269
10.5.6 완성된 MessagesService 273
10.5.7 MessagesService 테스트하기 275
10.6 ThreadsService 277
10.6.1 현재 Thread 집합의 맵(threads에서) 277
10.6.2 최신 스레드가 맨 위에 보이는 시간별 Thread 리스트(orderedthreads에서) 281
10.6.3 현재 선택된 Thread(currentThread에서) 282
10.6.4 현재 선택된 Thread에 사용할 Message 리스트(currentThreadMessages에서) 283
10.6.5 완성된 ThreadsService 286
10.7 데이터 모델 정리 288

CHAPTER 11 옵저버블 데이터 아키텍처 2부 — 뷰 컴포넌트 289

11.1 뷰 빌드하기: 최상위 컴포넌트, ChatApp 289
11.2 ChatThreadsComponent 291
11.2.1 ChatThreadsComponent 템플릿 292
11.3 단일 ChatThreadComponent 293
11.3.1 ChatThreadComponent 컨트롤러와 ngOnInit 294
11.3.2 ChatThreadComponent 템플릿 295
11.4 ChatWindowComponent 295
11.4.1 ChatWindowComponent 클래스 프로퍼티 296
11.5 ChatMessageComponent 305
11.5.1 ChatMessageComponent 템플릿 306
11.6 ChatNavBarComponent 308
11.6.1 ChatNavBarComponent @Component 308
11.6.2 ChatNavBarComponent 템플릿 310
11.7 정리 310

CHAPTER 12 리덕스와 함께하는 타입스크립트 312

12.1 리덕스 313
12.1.1 리덕스: 핵심 개념 314
12.2 코어 리덕스 개념 315
12.2.1 리듀서란 무엇일까? 315
12.2.2 Action과 Reducer 인터페이스 정의하기 315
12.2.3 첫 Reducer 만들기 316
12.2.4 첫 Reducer 실행하기 317
12.2.5 동작으로 카운터 조정하기 318
12.2.6 리듀서 switch 319
12.2.7 동작의 '인수' 320

12.3 상태 저장하기 321
12.3.1 저장소 사용하기 323
12.3.2 subscribe로 알림 받기 323
12.3.3 리덕스 코어 327
12.4 메시지 앱 327
12.4.1 메시지 앱의 state 328
12.4.2 메시지 앱의 action 328
12.4.3 메시지 앱의 reducer 329
12.4.4 동작 테스트하기 332
12.4.5 동작 생성자 333
12.4.6 진짜 리덕스 사용하기 335
12.5 앵귤러에서 리덕스 사용하기 336
12.6 앱 기획하기 337
12.7 리덕스 설정하기 338
12.7.1 애플리케이션 상태 정의하기 338
12.7.2 리듀서 정의하기 338
12.7.3 동작 생성자 정의하기 339
12.7.4 저장소 만들기 340
12.8 저장소 제공하기 341
12.9 앱 시동하기 343
12.10 AppComponent 344
12.10.1 import 344
12.10.2 템플릿 345
12.10.3 constructor 346
12.10.4 모두 합치기 347
12.11 앞으로 갈 곳 348
12.12 참조 348

CHAPTER 13 중급자를 위한 앵귤러 리덕스 350

13.1 이 장의 관점 351
13.2 대화 앱 조감도 351
13.1 이 장의 관점 351
13.2.1 컴포넌트 352
13.2.2 모델 352
13.2.3 리듀서 353
13.2.4 정리 353
13.3 모델 구현하기 353
13.3.1 User 354
13.3.2 Thread 354
13.3.3 Message 355
13.4 앱 상태 355
13.4.1 효율적인 코드 레이아웃 356

13.4.2 루트 리듀서 356
13.4.3 UsersState 357
13.4.4 ThreadsState 357
13.4.5 AppState 시각화하기 359
13.5 리듀서(그리고 동작 생성자) 빌드하기 360
13.5.1 현재 사용자 동작 생성자 설정하기 360
13.5.2 UsersReducer — 현재 사용자 설정하기 361
13.5.3 스레드와 메시지 개요 362
13.5.4 새 스레드 동작 생성자 추가하기 362
13.5.5 새 스레드 리듀서 추가하기 363
13.5.6 동작 생성자에 새 메시지 추가하기 364
13.5.7 새 메시지 리듀서 추가하기 365
13.5.8 스레드 동작 생성자 선택하기 366
13.5.9 스레드 리듀서 선택하기 367
13.5.10 리듀서 정리 368
13.6 앵귤러 대화 앱 빌드하기 369
13.6.1 ChatApp 최상위 컴포넌트 370
13.6.2 ChatPage 컴포넌트 371
13.6.3 컨테이너 vs. 표현용 컴포넌트 372
13.7 ChatNavBarComponent 빌드하기 374
13.7.1 리덕스 셀렉터 375
13.7.2 Threads 셀렉터 376
13.7.3 읽지 않은 메시지 개수 셀렉터 378
13.8 ChatThreadsComponent 빌드하기 379
13.8.1 ChatThreadsComponent 컨트롤러 379
13.8.2 ChatThreadsComponent 템플릿 381
13.9 단일 ChatThreadComponent 381
13.9.1 ChatThreadComponent 템플릿 383
13.10 ChatWindowComponent 빌드하기 384
13.11 ChatMessageComponent 391
13.11.1 incoming 설정하기 391
13.11.2 ChatMessageComponent 템플릿 392
13.12 정리 393

CHAPTER 14 컴포넌트 고급 395

14.1 스타일 396
14.1.1 뷰(스타일) 캡슐화 398
14.1.2 Shadow DOM 캡슐화 401
14.1.3 캡슐화 금지 403
14.2 팝업 만들기 — 호스트 요소 참조하기와 수정하기 405
14.2.1 팝업의 구조 405
14.2.2 ElementRef 사용하기 407

14.2.3 host에 바인딩하기 408
14.2.4 exportAs를 사용하여 버튼 추가하기 410
14.3 콘텐트 프로젝션으로 메시지 창 만들기 412
14.3.1 호스트의 CSS 변경하기 413
14.3.2 ng-content 사용하기 413
14.4 이웃 지시자 조회하기 — 탭 작성하기 415
14.4.1 ContentTabsetComponent 컴포넌트 417
14.4.2 ContentTabsetComponent 사용하기 419
14.5 라이프사이클 훅 421
14.5.1 OnInit과 OnDestroy 422
14.5.2 OnChanges 425
14.5.3 DoCheck 430
14.5.4 CommentsListComponent로 댓글 리스트 렌더링하기 435
14.5.5 AfterContentInit, AfterViewInit, AfterContentChecked, AfterViewChecked 441
14.6 템플릿 고급 446
14.6.1 ngIf 다시 작성하기 — ngBookIf 447
14.6.2 ngFor 다시 작성하기 — NgBookFor 449
14.7 변경 감지 454
14.7.1 변경 감지 커스터마이징하기 458
14.7.2 존 464
14.7.3 옵저버블과 OnPush 464
14.8 정리 468

CHAPTER 15 테스트하기 469

15.1 테스트 주도? 469
15.2 엔드투엔드 vs. 유닛 테스트 470
15.3 테스트 도구 471
15.3.1 재스민 471
15.3.2 카마 472
15.4 유닛 테스트 작성하기 472
15.5 앵귤러 유닛 테스트용 프레임워크 472
15.6 테스트 설정하기 473
15.7 서비스와 HTTP 테스트하기 474
15.7.1 HTTP 고려 사항 475
15.7.2 스텁 476
15.7.3 목 476
15.7.4 getTrack 테스트하기 479
15.8 컴포넌트 라우팅 테스트하기 485
15.8.1 테스트용 라우터 만들기 486
15.8.2 의존성 모킹하기 489
15.8.3 스파이 489

15.9 테스트 코드 다시 살펴보기 492
15.9.1 fakeAsync와 advance 494
15.9.2 inject 495
15.9.3 ArtistComponent의 초기화 테스트하기 496
15.9.4 ArtistComponent 메서드 테스트하기 497
15.9.5 ArtistComponent DOM 템플릿 값 테스트하기 498
15.10 폼 테스트하기 500
15.10.1 ConsoleSpy 만들기 502
15.10.2 ConsoleSpy 설치하기 503
15.10.3 테스트 모듈 설정하기 504
15.10.4 폼 테스트하기 504
15.10.5 폼 테스트 리팩토링하기 507
15.11 HTTP 요청 테스트하기 510
15.11.1 POST 테스트하기 510
15.11.2 DELETE 테스트하기 513
15.11.3 HTTP 헤더 테스트하기 513
15.11.4 YouTubeSearchService 테스트하기 515
15.12 정리 521

CHAPTER 16 앵귤러JS 1.x 앱을 앵귤러 4로 변환하기 522

16.1 배경지식 522
16.2 우리가 만들려는 것 523
16.3 앵귤러 4에 앵귤러JS 1 매핑하기 524
16.4 상호운영성의 요건 526
16.5 앵귤러JS 1 앱 527
16.5.1 ng1-app HTML 528
16.5.2 코드 살펴보기 529
16.5.3 ng1: PinsService 530
16.5.4 ng1: 라우트 설정하기 531
16.5.5 ng1: HomeController 532
16.5.6 ng1: / HomeController 템플릿 532
16.5.7 ng1: pin 지시자 533
16.5.8 ng1: pin 지시자 템플릿 533
16.5.9 ng1: AddController 535
16.5.10 ng1: AddController 템플릿 537
16.5.11 ng1: 정리 539
16.6 하이브리드 앱 만들기 539
16.6.1 하이브리드 프로젝트 구조 540
16.6.2 하이브리드 앱 부트스트랩하기 542
16.6.3 업그레이드해야 하는 것들 544
16.6.4 기타: 타입 지정용 파일 546
16.6.5 ng2 PinControlsComponent 작성하기 549

16.6.6 ng2 PinControlsComponent 사용하기 550
16.6.7 ng2 PinControlsComponent를 ng1으로 다운그레이드하기 552
16.6.8 ng2에서 핀 추가하기 554
16.6.9 ng1 PinsService와 $state를 ng2로 업그레이드하기 555
16.6.10 ng2 AddPinComponent 작성하기 556
16.6.11 AddPinComponent 사용하기 561
16.6.12 ng2 서비스를 ng1으로 나타내기 562
16.6.13 AnalyticsService 작성하기 562
16.6.14 ng2 AnalyticsService를 ng1으로 다운그레이드하기 563
16.6.15 ng1에서 AnalyticsService 사용하기 564
16.7 정리 565

CHAPTER 17 네이티브스크립트 — 앵귤러 개발자를 위한 모바일 애플리케이션 566

17.1 네이티브스크립트란 무엇일까? 566
17.1.1 네이티브스크립트의 차별성 567
17.1.2 네이티브스크립트에 필요한 시스템과 개발 요건은 무엇일까? 568
17.2 네이티브스크립트와 앵귤러로 첫 모바일 애플리케이션 만들기 570
17.2.1 크로스 플랫폼 개발을 위한 빌드 플랫폼 추가하기 571
17.2.2 안드로이드와 iOS용으로 빌드하기와 테스트하기 571
17.2.3 자바스크립트, 안드로이드, iOS 플러그인과 패키지 설치하기 572
17.3 웹과 네이티브스크립트 UI 및 UX의 다른 점 이해하기 573
17.3.1 네이티브스크립트 페이지 레이아웃 기획하기 573
17.3.2 페이지에 UI 컴포넌트 추가하기 574
17.3.3 CSS로 컴포넌트 스타일링하기 576
17.4 위치 기반 사진 애플리케이션 개발하기 577
17.4.1 네이티브스크립트 프로젝트 만들기 578
17.4.2 다중 페이지 마스터-디테일 인터페이스 만들기 579
17.4.3 사진과 데이터를 받기 위한 플리커 서비스 만들기 582
17.4.4 기기 위치와 거리를 계산하기 위한 서비스 만들기 587
17.4.5 네이티브스크립트 애플리케이션에 맵박스 포함하기 590
17.4.6 지오로케이션 애플리케이션의 첫 번째 페이지 구현하기 592
17.4.7 지오로케이션 애플리케이션의 두 번째 페이지 구현하기 596
17.5 테스트하기! 598
17.6 앵귤러 개발자를 위한 네이티브스크립트 598

업데이트 이력 601
찾아보기 615

옮긴이 머리말

이 책이 어떤 책인지 그리고 어떻게 읽어야 하는지는 저자가 꼼꼼하게 설명하고 있으므로 구태여 제가 말을 보탤 이유가 없을 것 같습니다. 저는 이 책을 우리말로 옮긴 사람으로서 몇 가지를 말씀드리고자 합니다.

생성자, 장식자, 반복자…

전부 '아들 자'를 넣어 만든 말입니다. 영어로는 아시다시피 constructor, decorator, iterator입니다. 저는 '아들 자'가 붙은 이런 말들을 선호하지 않습니다. 아직 마땅한 조어를 생각하지 못해 최선이 없으면 차악이라도 고른다는 마음으로 컨스트럭터나 데코레이터, 이터레이터를 씁니다. 다만 지금도 어떤 말이 더 어울릴지 고민하고 있고, 문장에 자연스럽게 적용할 수 있는 말이 생각나면 실제로 써 볼 것입니다. 이 과정에서 다른 사람의 입에도 착착 감기는 말이라면 살아남겠죠. 사실 컨스트럭터나 데코레이터, 이터레이터로 표현하고픈 생각도 없습니다. constructor, decorator, iterator 그대로 표현하고픈 마음이 더 크지만, 이 또한 최선이 아닌 차악으로 쓰고 있을 뿐입니다. 다음은 스택오버플로에 올라온 질문 가운데 하나입니다. 우리말로 어떻게 표현해야 할까요?

Why are generator methods constructors?

'생성자 메서드는 왜 생성자입니까?'로 생각하시나요? 최첨단 기술이 하루가 다르게 발전하고 있으니 쏟아져 나오는 용어들을 우리말로 바꾸기란 거의 불가능에 가깝습니다. 저는 오히려 우리말로 바꾸지 않고 쓰는 것이 더 좋다고 생각하고 있습니다. 어울리지도 않는 우리말을 억지로 가져다 붙이는 것보다 문장 자체를 우리말답게 다듬는 노력이 더 필요하다고 생각합니다. 저 같은 사람이 더욱 열심히 노력해야겠죠.

아무쪼록 이 책이 여러분께 도움이 되기를 바랍니다. 마지막으로, 번역할 기회를 주신 장성두 대표님께 감사의 말씀을 드립니다.

옮긴이 **배장열**

이 책에 대하여

이 책의 유일한 목적은 독자들이 앵귤러를 배울 때 유용한 리소스 역할을 하는 것이다. 여러분(그리고 여러분의 팀)이 이 책을 끝까지 읽는다면 신뢰할 만하고 강력한 앵귤러 앱을 만드는 데 필요한 모든 것을 갖추게 될 것이다.

앵귤러는 다양하고 풍부한 기능으로 무장한 프레임워크다. 하지만 이는 모든 요소요소를 빠짐없이 이해하기가 만만치 않다는 뜻이기도 하다. 이 책에서는 각종 도구의 설치를 비롯해 컴포넌트 작성, 폼 사용, 페이지 간 라우팅, API 호출 등 모든 것을 다룰 것이다.

하지만 그보다 먼저 이 책의 활용성을 최대한 끌어올리기 위한 몇 가지 가이드라인을 제시하고자 한다. 요약하자면 다음과 같다.

- 예시 코드에 접근하는 법
- 문제가 생겼을 때 도움을 받는 법

예시 코드 실행하기

이 책에는 실행 가능한 예시 코드가 상당량 제공된다. 이 코드는 https://github.com/Jpub/ng5MasterBook에서 다운로드할 수 있다.

모든 예시 코드는 npm 프로그램을 사용해 실행할 수 있다. 다시 말해, 어떤 예시 코드든 다음 명령으로 실행할 수 있다.

```
npm install
npm start
```

npm 사용법을 잘 모르는 독자를 위해 1장 '시작하기'에 설치 방법을 설명해 두었다.

npm start를 실행하면 앱을 볼 수 있는 URL이 화면에 출력된다.

특정 앱을 어떻게 실행하는지 잘 모르겠다면 해당 프로젝트의 디렉터리에 제공되는 README.md 파일을 확인해 보기 바란다. 모든 샘플 프로젝트에는 각 앱을 실행하는 데 필요한 여러 지침이 담긴 README.md 파일이 함께 제공된다.

앵귤러 CLI

한두 가지 예외를 빼면 나머지 모든 프로젝트는 앵귤러 CLI를 바탕으로 진행되었다. 달리 언급하지 않았다면 프로젝트마다 ng 명령을 사용할 수 있다.

예를 들어, 샘플 앱을 실행하려면 ng serve를 실행한다(이는 일반적으로 npm start를 입력하면 실행된다). 대개 ng build를 사용해 프로젝트를 자바스크립트로 컴파일할 수 있다(1장에서 언급할 것이다). 그리고 ng e2e를 실행해 엔드투엔드 테스트를 실행할 수도 있다.

앵귤러 CLI는 속을 깊이 파헤치지 않아도 웹팩(Webpack) 기반임이 쉽게 드러난다. 웹팩은 다양한 타입스크립트, 자바스크립트, CSS, HTML, 이미지 파일 등을 함께 처리할 수 있는 도구다. 앵귤러 CLI는 앵귤러를 사용하기 위한 필수 조건이 아니다. 단지 쉽게 시작하기 위한 웹팩(혹은 다른 도구들)의 래퍼(wrapper)일 뿐이다.

코드 블록과 표기 의도

이 책의 모든 코드는 실행 가능한 예시로서, 샘플 코드에서 전체를 찾아볼 수 있다. 예를 들어 다음은 1장에 사용된 코드 블록이다.

code/first-app/angular-hello-world/src/app/app.component.ts

```
8  export class AppComponent {
9    title = 'app works!';
10 }
```

코드 블록의 제목인 code/first-app/angular-hello-world/src/app/app.component.ts는 이 코드가 포함된 파일 경로를 나타낸다.

코드 블록만 봐서 무슨 일을 하는지 헷갈린다면 텍스트 편집기에서 이 경로에 해당하는 전체 코드를 살펴보기 바란다. 이 책은 독자 여러분이 예시 코드를 옆에 두고 보면서 책의 내용도 함께 읽는다고 가정하고 구성했다.

예를 들어 코드가 동작하려면 여러 라이브러리를 가져와야(import) 할 때가 있다. 책 초반부에서는 이 import문을 지면에 실었다. 그러지 않으면 라이브러리가 어디서 비롯된 것인지 명확하게 알기 어려울 수도 있기 때문이다. 하지만 후반부로 갈수록 내용이 깊어지므로 단순한 보일러판 코드를 반복하기보다 핵심 개념을 중점적으로 다뤄야 한다. 따라서 코드 이해력을 높이려면 전체 코드를 열어 직접 확인할 것을 권고한다.

코드 블록 번호

책에서는 긴 예시 코드를 몇 단계로 나눠 구성한 곳도 많다. 번호가 접미사처럼 붙은 파일이 있다면 이는 더 큰 파일의 일부라는 의미로 받아들여야 한다.

예를 들어, 6장 의존성 주입에 price.service.1.ts라는 파일이 있다. 여기서 .N.ts 구조는 번호가 붙지 않은 최종 파일을 위한 일부분을 나타낸다. 따라서 이 경우에는 최종 파일이 price.service.ts다. 이런 식으로 번호를 매긴 이유는 첫째로 중간 코드를 유닛 테스트할 수 있고, 둘째로 특정 단계에서 전체 파일을 확인할 수 있기 때문이다.

버전 관리

이미 알고 있겠지만, 이 책에서 다루고 있는 앵귤러는 이전 프레임워크인 앵귤러JS의 후속작이다. 이 때문에 여러 혼동이 발생하기도 한다. 특히 블로그 포스트나 관련 문서를 읽을 때가 그렇다. 이름 규정에 관한 공식 가이드라인에 따르면 앵귤러JS는 앵귤러JS 1.x, 즉 앵귤러의 초기 버전을 가리키는 용어다. 앵귤러의 새 버전은 자바스크립트가 아닌 타입스크립트를 주 언어로 사용하기 때문에 JS라는 단어가 떨어져 나가고 앵귤러라는 이름만 남았다. 이후 오랫동안 이 둘을 구별하는 유일한 방법은 새 앵귤러를 앵귤러 2로 언급하는 것이었다. 하지만 앵귤러 팀은 2017년에 6개월 단위로 하는 새로운 메이저 버전 출시 업그레이드 계획에 맞춰 시맨틱 버저닝(semantic versioning)으로 변경했다. 다음 버전을 앵귤러 4, 앵귤러 5 등으로 부르는 것이 아니라 숫자 부분을 빼고 단지 앵귤러로만 부르기로 한 것이다. 이 책에서 앵귤러를 가리킬 때는 혼동을 피하기 위해 앵귤러 또는 앵귤러 4로 부른다. 반면, 구식 자바스크립트 앵귤러를 가리킬 때는 앵귤러JS 또는 앵귤러JS 1.x를 사용한다.

도움 받기

우리는 명확하고 정확한 의미를 전달하기 위해 최선을 다했다. 하지만 예시 코드를 실행하는 과정에서 예기치 않은 문제가 일어날 수도 있을 것이다. 일반적으로는 다음 세 가지 유형으로 문제들을 정리할 수 있다.

- 책의 버그(예 설명한 내용이 틀린 경우)
- 코드의 버그
- 여러분 코드의 버그

잘못된 내용을 설명했다거나 명확하게 개념을 이해하지 못하겠다면 주저하지 말고 이메일을 보내길 바란다. 이 책은 명확하고 정확해야 하기 때문이다. 그리고 코드에서 버그를 발견했을 때도 우리에게 알려 주기 바란다.

코드에 문제가 없는데도 여러분의 앱이 올바로 동작하지 않는다면 해결하기가 더 어려워질 수도 있다. 그럴 경우 비공식 커뮤니티 대화방에 참여하는 것도 한 가지 방법이다(뒤에 나오는 Gitter 커뮤니티 참조). 우리가 주기적으로 이 대화방에 참여하기는 하지만 수없이 많은 다른 독자들이 여러분의 문제를 더욱더 빠르게 해결해 줄 수 있을 것이다.

그래도 문제가 해결되지 않았다면, 우리는 여러분의 의견을 귀 기울이고 있으므로 적절한 응답을 받을 수 있도록 몇 가지 안내를 하고자 한다.

이메일

우리에게 이메일을 보내 기술 지원을 받을 때는 다음을 알려 주기 바란다.

- 책의 리비전 번호
- 사용하는 운영체제(예 macOS 10.13.1, 윈도우 10)
- 장과 프로젝트 이름
- 이메일을 보내는 목적
- 이미 시도해 본 내용
- 예상하는 결과
- 실제로 확인한 결과(로그 출력 포함)

기술 지원을 받을 수 있는 쉽고 간단한 방법은 문제를 간단하게 설명한 이메일이다. 이메일을 보낼 곳은 us@fullstack.io다. 우리는 항상 여러분의 의견에 귀를 기울이고 있다.

기술 지원 응답 시간

우리는 일주일에 한 번 무료 기술 지원을 진행하고 있다. 더 이른 응답이 필요하다면 우리의 프리미엄 지원도 고려해 보기 바란다.

커뮤니티 활동

Gitter 커뮤니티 https://gitter.im/ng-book/ng-book

우리는 이 책과 관련해 Gitter를 활용한 커뮤니티 대화방을 운영하고 있다. 다른 사람들과 함께 앵귤러를 배우고 싶다면 Gitter에 마련된 우리 대화방을 방문하기 바란다.

새로운 내용을 위한 의견 제시 https://fullstackio.canny.io/ng-book

우리는 끊임없이 이 책을 업데이트하며 새 블로그 포스트를 작성하고 새 재료를 만들어 내고 있다. 여러분도 새로 업데이트할 내용에 참여할 수 있다.

트위터로 업데이트 알림 받기 @fullstackio

이 책의 업데이트 소식을 트위터로 받고 싶다면 @fullstackio를 팔로우하기 바란다.

이 책의 주요 내용

본격적으로 뛰어들기에 앞서 각 장을 간략하게 소개해 이 책으로 무엇을 얻을 수 있는지 예상해 보는 것도 나쁘지 않을 것이다.

처음 몇 장에서는 일어서서 달릴 수 있는 기초 체력을 연습한다. 첫 앱을 만들고, 내장 컴포넌트를 사용하고 컴포넌트를 직접 만들 것이다.

그다음은 폼이라든가 API, 서로 다른 페이지 간 라우팅, 의존성 주입 등 중급 개념으로 나아간다.

그 이후에는 고급 개념을 다룬다. 상당한 지면을 할애해 데이터 아키텍처를 언급할 것이다. 클라이언트/서버 애플리케이션의 상태 관리는 어려운 부분이다. 우리는 RxJS 옵저버블과 리덕스라는 두 가지 대중적인 방법을 다룰 것이다. 여기서는 두 가지 방법을 적용해 같은 앱을 만들면서 두 방법을 서로 비교하고 장단점을 따져 어느 방법이 여러분 팀에 적합한지 생각해 볼 것이다.

마지막으로, 앵귤러의 가장 강력한 기능인 복잡한 고급 컴포넌트를 어떻게 만드는지 다룰 것이다. 앱을 테스트하는 방법, 앵귤러 1 앱을 앵귤러 4+ 앱으로 변환하는 방법도 다룰 것이다. 그리고 앵귤러와 네이티브스크립트를 사용해 네이티브 모바일 앱을 만드는 과정으로 대단원의 막을 내리고자 한다.

이 책을 통해 여러분은 각종 구닥다리 블로그 포스트를 읽는 것보다 더욱더 빠르게 실제 앵귤러 앱을 만들 수 있게 될 것이다. 자, 집중력을 발휘할 시간이다. 이제 곧 앵귤러 전문가의 길이 멀지 않았다. 건투를 빈다!

베타리더 후기

김용현(MicrosoftMVP)

웹 서비스는 ECMA 6의 등장과 Node.js의 활용으로 다른 분야보다 빠르게 확장되고 있는 분야입니다. 앵귤러는 이러한 흐름의 최선봉에 서 있습니다. 이 책은 익숙하지 않은 독자를 위해 기본적인 요소를 쉽게 설명하고, 앵귤러 1 등 과거 버전과의 비교를 통해 현재 지식을 기반으로 새로운 기술을 자연스럽게 습득하도록 유도하며, 고급 기능 활용과 레퍼런스 안내를 통해 개발자의 코드를 더욱 풍부하게 해 줄 것입니다.

노태환(로아팩토리)

앵귤러에만 집중하는 것이 아니라 앵귤러를 어떻게 잘 사용할 수 있을지에 초점을 맞춘 책입니다. TypeScript, Reactive Programming(with rxjs), Redux 등 모던 웹을 구성하고 있는 대부분의 기술이 집약되어 있어서 필요한 부분만 선택적으로 학습할 수도 있습니다.

이상현(SI 개발자)

페이지를 보고 참 두껍다고 생각했는데 내용을 보니 오히려 요약본에 가깝습니다. 더 두꺼워질 책을 깔끔한 설명과 실습 가능한 예제로 알차게 구성한 것 같습니다. 물론, 내용이 결코 가볍지 않아 볼수록 공부할 내용이 많다는 것도 알게 해 줍니다. 아주 맘에 드는 책이고, 이 책을 통해 대두되고 있는 웹 관련 기술을 충분히 맛볼 수 있지 않을까 생각해 봅니다.

이아름

앵귤러가 대규모로 업데이트되면서 한동안 손도 안 댔었는데, 이 책을 보면서 차근차근 해보니 다시 감이 돌아온 기분이 들었습니다. 깔끔한 편집도 마음에 들었습니다.

이요셉(취미코더)

마스터북답습니다. 무려 600페이지에 앵귤러 내용을 꽉꽉 채워 넣었고, 앵귤러-리덕스 조합에 앵귤러JS, 앵귤러 4 컨버전, 네이티브스크립트까지 소개합니다. 쉽지 않지만 도전할 만한 책입니다.

이정훈(SK주식회사)

IT 세계에서 무언가를 배우려면 직접 코딩해 보는 것이 가장 좋다고 생각합니다. 이 책은 앵귤러 앱을 단계적으로 만들면서 배울 수 있게 구성되어 있습니다. 이론을 유심히 읽고, 실습 내용을 직접 코딩해 보면 좋겠습니다. 손으로 직접 작성하다 보면 말로는 이해되지 않는 부분이 어느 순간 이해될 것입니다. 앵귤러를 배우기 쉽게 구성된 책을 찾는다면 이 책을 추천합니다.

제이펍 베타리더스팀

제이펍은 책에 대한 애정과 기술에 대한 열정이 뜨거운 베타리더들로 하여금
출간되는 모든 서적에 사전 검증을 시행하고 있습니다.

CHAPTER

1

첫 앵귤러 웹 애플리케이션 만들기

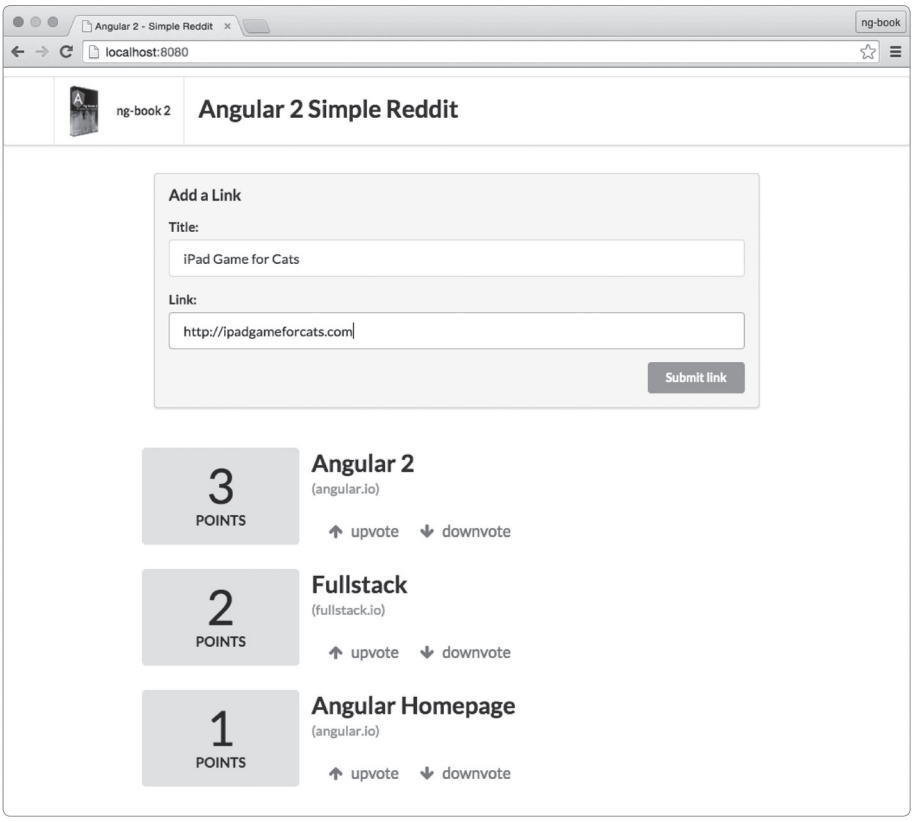
1.1 단순한 레딧 클론

이 장에서는 사용자가 글을 게시하고(제목과 URL 포함) 다른 게시물에 투표하는 애플리케이션을 만들 것이다. 이 애플리케이션을 레딧(reddit)[1]이나 프로덕트 헌트(product hunt)[2] 같은 사이트의 출발이라고도 볼 수 있을 것이다. 이 앱은 단순하지만, 다음 내용을 비롯한 앵귤러의 핵심을 대부분 다루고 있다.

- 커스텀 컴포넌트 빌드하기
- 양식(폼)을 통해 사용자 입력 받기
- 객체 리스트를 뷰로 렌더링하기
- 사용자 클릭을 가로채 처리하기
- 앱을 서버에 배포하기

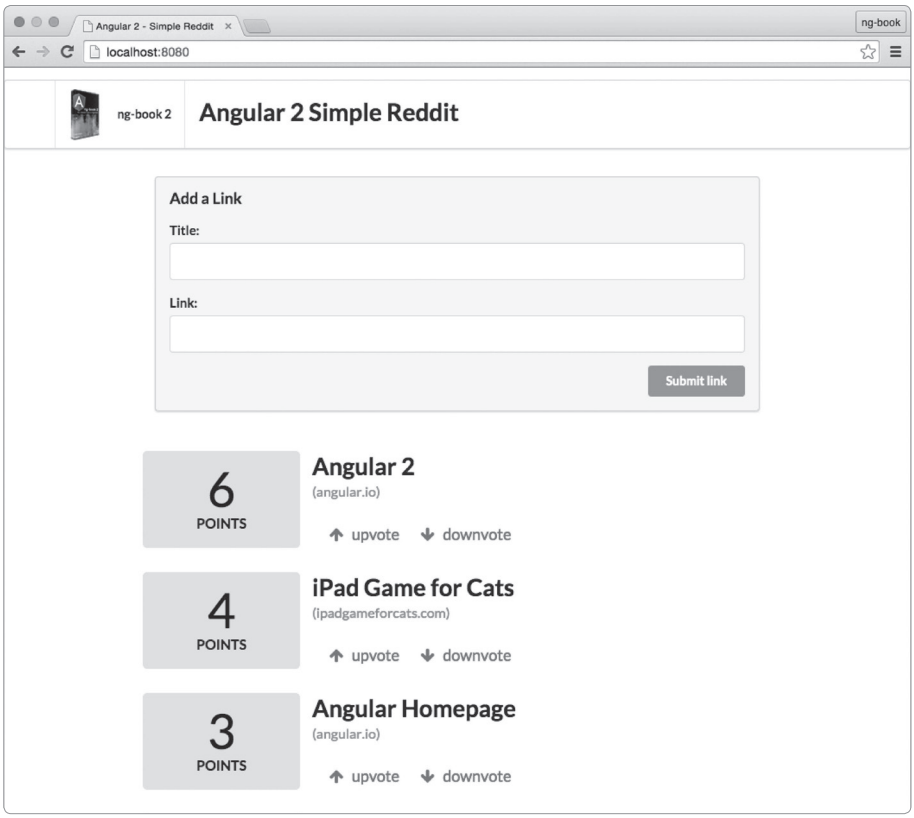
이 장을 모두 읽으면 빈 폴더를 받아 기본적인 앵귤러 애플리케이션을 만들고, 이를 현장에 배포하는 과정을 속속들이 이해하게 될 것이다. 그리고 자신만의 앵귤러 앱을 빌드하는 데 탄탄한 토대가 되어줄 것이다. 다음은 완성된 앱의 모습이다.

1 www.reddit.com
2 www.producthunt.com



완성된 애플리케이션

우선, 한 사용자가 새 링크를 제출하면 다른 사용자들은 이 게시물을 찬성하거나 반대한다. 게시물의 찬성과 반대가 쌓여 점수가 되며, 유용하다고 생각한 링크에는 투표도 할 수 있다.



새 글이 표시된 앱

이번 프로젝트에서 그리고 이 책 전반에 걸쳐 타입스크립트(TypeScript)를 사용할 것이다. 타입을 추가할 수 있는 타입스크립트는 자바스크립트 ES6의 슈퍼세트이다. 타입스크립트에 관해서는 이 장이 아니라 다음 장에서 깊게 다룰 것이다. 따라서 당분간은 문법이 다소 낯설더라도 개의치 말길 바란다. ES5('일반적인' 자바스크립트)나 ES6(ES2015)가 낯설지 않다면 별다른 문제 없이 이 장의 내용을 진행할 수 있다.

1.2 시작하기

1.2.1 Node.js와 npm

앵귤러를 시작하려면 우선 Node.js부터 설치해야 한다. Node.js 설치 방법은 여러 가지가 있으므로 구체적인 과정은 Node.js 웹사이트[3]를 참고하기 바란다. 다만 Node 6.9.0 이상 버전을 설치해야 한다.

> 맥 사용자는 홈브류(Homebrew) 같은 패키지 관리자를 사용하지 말고 Node.js 웹사이트에서 직접 Node.js를 설치하는 것이 가장 확실하다. 홈브류를 통한 Node.js 설치는 몇 가지 알려진 문제가 있다.

흔히 줄여서 npm이라고 표현하는 노드 패키지 관리자(Node Package Manager)가 Node.js의 일부분으로 설치된다. npm을 우리 개발 환경에서 사용할 수 있는지 확인하려면 터미널에서 다음을 입력한다.

```
$ npm -v
```

버전 번호가 출력되지 않고 오류만 표시된다면 npm이 포함된 Node.js 인스톨러를 다운로드했는지 확인해야 한다. npm 버전은 최소한 3.0.0이어야 한다.

1.2.2 타입스크립트

Node.js가 설치되면 타입스크립트를 설치한다. 최소한 2.1 버전을 설치해야 하며, 설치 명령은 다음과 같다.

```
1 $ npm install -g typescript
```

> **타입스크립트를 반드시 사용해야 하는가?** 반드시는 아니지만 대체로는 그렇다. 앵귤러에는 ES5 API가 포함되어 있지만, 앵귤러는 타입스크립트로 작성되며, 일반적으로는 모든 사람이 타입스크립트를 사용하기 때문이다. 이 책에서도 앵귤러를 한결 더 쉽게 다루기 위해 타입스크립트를 사용할 것이다. 다만, 타입스크립트가 엄격한 필요조건은 아니라는 점을 밝힌다.

3 https://nodejs.org/ko/download/

1.2.3 브라우저

앵귤러 앱의 배포를 위해서 구글의 크롬 웹 브라우저[4]를 강력하게 권장한다. 이 책에서는 크롬 개발자 툴킷을 사용하여 앱을 개발하고 디버깅할 것이므로 지금 바로 다운로드하는 것이 좋다.

1.2.4 윈도우 사용자를 위한 특별 과정

이 책 전반에 걸쳐 우리는 터미널에서 유닉스/맥 명령을 사용할 것이다. 이들 명령은 ls나 cd처럼 크로스 플랫폼이다. 하지만 경우에 따라서는 유닉스/맥 전용 명령을 사용할 수도 있고, 유닉스/맥 전용 플래그(ls -lp 등)를 포함할 수도 있다. 따라서 유닉스/맥 명령에 상응하는 다른 셸 명령을 직접 찾아야 할 일도 있을 것이다. 다행히 터미널에서 진행하는 작업을 최소로 줄였으므로 그런 번거로운 일을 자주 만나지는 않을 것이다.

윈도우 사용자도 우리의 터미널 예시들에 유닉스/맥 명령이 사용된다는 점을 알고 있어야 한다.

1.3 앵귤러 CLI

앵귤러는 사용자가 커맨드라인에서 프로젝트를 생성하고 관리할 수 있는 유틸리티를 제공하고 있다. 이 유틸리티를 사용하면 프로젝트 생성이나 새 컨트롤러 추가 등의 작업을 자동화할 수 있다. 앞으로 만들 애플리케이션에서 공통되는 패턴을 만들고 관리할 때 앵귤러 CLI가 큰 도움이 될 것이다.

앵귤러 CLI를 설치하려면 다음 명령을 실행한다.

```
1 $ npm install -g @angular/cli
```

설치가 끝나면 커맨드라인에서 ng 명령을 사용하여 앵귤러 CLI를 실행할 수 있다. 실제로 ng 명령을 사용하면 이런저런 출력이 화면에 쏟아지는데, 그중에서 다음 부분을 눈여겨보기 바란다.

4 https://www.google.com/chrome/

```
$ ng --version
```

모든 것이 올바로 설치되었다면 현재 버전이 터미널에 출력될 것이다. 축하한다!

macOS나 리눅스에서는 다음 행이 출력되는 경우가 있다.

```
Could not start watchman; falling back to NodeWatcher for file system events.
```

watchman이라는 도구가 설치되지 않았다는 뜻이다. watchman은 파일시스템의 파일들을 감시할 때 앵귤러 CLI를 지원하는 도구다. macOS에서는 홈브류를 사용하여 다음 명령으로 설치할 수 있다.

```
$ brew install watchman
```

macOS에서 홈브류 실행 시 오류가 발생하면 아직 홈브류가 설치된 상태가 아닐 수도 있다. 홈브류 페이지[5]를 참고하여 홈브류를 설치한다.

리눅스 사용자는 EmberCLI[6]에서 watchman 설치 정보를 확인하기 바란다. 윈도우에서는 따로 설치할 것이 없으며, 앵귤러 CLI는 네이티브 Node.js 와처를 사용한다.

앵귤러 CLI의 도움말은 다음 명령으로 확인할 수 있다.

```
$ ng --help
```

모든 옵션을 한꺼번에 이해하려고 애쓰지 않아도 된다. 이 장에서 중요한 몇 가지를 추려 설명할 것이다.

이제 앵귤러 CLI와 관련 파일이 모두 설치되었다. 본격적으로 첫 애플리케이션을 만들어 보자.

1.3.1 예시 프로젝트

터미널을 열고 ng new 명령을 실행하여 프로젝트를 새로 만든다.

```
$ ng new angular-hello-world
```

5 http://brew.sh/
6 https://ember-cli.com/user-guide/#watchman

이 명령을 실행하면 다음과 같이 출력된다.

```
  create .editorconfig
  create README.md
  create src/app/app.component.css
  create src/app/app.component.html
  create src/app/app.component.spec.ts
  create src/app/app.component.ts
  create src/app/app.module.ts
  create src/assets/.gitkeep
  create src/environments/environment.prod.ts
  create src/environments/environment.ts
  create src/favicon.ico
  create src/index.html
  create src/main.ts
  create src/polyfills.ts
  create src/styles.css
  create src/test.ts
  create src/tsconfig.json
  create .angular-cli.json
  create e2e/app.e2e-spec.ts
  create e2e/app.po.ts
  create e2e/tsconfig.json
  create .gitignore
  create karma.conf.js
  create package.json
  create protractor.conf.js
  create tslint.json
Successfully initialized git.
Installing packages fortooling via npm.
Installed packages for tooling via npm.
```

프로젝트가 생성하는 파일들은 설치된 @angular/cli의 버전에 따라 조금씩 달라질 수 있다.

잠시 기다리면 npm 의존성 파일들이 설치되며, 설치가 끝나면 다음과 같은 성공 메시지가 출력된다.

```
Project 'angular-hello-world' successfully created.
```

많은 파일이 생성되었다! 지금 당장 이 파일들을 속속들이 알려고 할 까닭은 없다. 앞으로 차근차근 살펴보면서 하나하나가 어떤 목적으로 사용되는지 설명할 것이다.

ng 명령이 만들어 준 angular-hello-world 디렉터리 안으로 들어가 무엇이 있는지 살펴보기로 하자.

```
$ cd angular-hello-world
$ tree -F -L 1
.
├── README.md                 // README
├── .angular-cli.json         // angular-cli 설정 파일
├── e2e/                      // 엔드투엔드 테스트
├── karma.conf.js             // 단위 테스트 설정
├── node_modules/             // 설치된 의존성 파일들
├── package.json              // npm 설정
├── protractor.conf.js        // e2e 테스트 설정
├── src/                      // 애플리케이션 소스 위치
└── tslint.json               // 린터 설정 파일
```

tree 명령은 온전히 선택 사항이다. macOS에서는 brew install tree 명령으로 설치할 수 있다.

지금 눈여겨보아야 할 폴더는 src인데, 이곳에 우리의 애플리케이션이 자리를 잡게 된다. 그 안을 들여다보자.

```
$ cd src
$ tree -F
.
|-- app/
|   |-- app.component.css
|   |-- app.component.html
|   |-- app.component.spec.ts
|   |-- app.component.ts
|   `-- app.module.ts
|-- assets/
|-- environments/
|   |-- environment.prod.ts
|   `-- environment.ts
|-- favicon.ico
|-- index.html
|-- main.ts
|-- polyfills.ts
|-- styles.css
|-- test.ts
`-- tsconfig.json
```

즐겨 사용하는 텍스트 에디터에서 index.html을 연다. index.html의 내용은 다음과 같다.

code/first-app/angular-hello-world/src/index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>AngularHelloWorld</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

한꺼번에 살펴보기에는 많으므로 몇 부분으로 나누어 보자.

code/first-app/angular-hello-world/src/index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>AngularHelloWorld</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
```

HTML 파일이 낯설지 않은 독자는 이곳에 별 눈길을 주지 않을 텐데, html 문서의 핵심 구조와 페이지의 문자세트, 타이틀, 기본 href 등의 메타데이터가 선언된 곳이다.

그리고 템플릿 몸체(body)는 다음과 같다.

code/first-app/angular-hello-world/src/index.html

```
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

app-root 태그는 우리의 애플리케이션이 렌더링되는 곳이다. Loading...은 일종의 자리 맡기용 텍스트다. 이곳에 우리의 '앱 코드가 로드되기 전'에 표시될 텍스트를 입력한다. 예를 들어,

로딩을 나타내는 '스피너(spinner)' img 태그를 이 자리에 두면 자바스크립트나 앵귤러 앱이 로딩 중임을 알릴 수 있다.

하지만 app-root 태그가 무엇이고 어디서 온 것일까? app-root는 우리의 앵귤러 앱에서 정의된 **컴포넌트**다. 앵귤러에서는 **자체 HTML 태그를 정의**하고 맞춤형 새 기능을 부여할 수 있다. app-root 태그는 페이지에서 앱의 '진입점'에 해당한다.

앱을 있는 그대로 실행해 본다. 곧, 컴포넌트가 어떻게 정의되었는지 살펴볼 것이다.

1.4 애플리케이션 코드 작성하기

1.4.1 애플리케이션 실행하기

구체적으로 코드를 수정하기 전에 앱을 브라우저에서 로드해 보자. 앵귤러 CLI는 HTTP 서버를 내장하고 있으므로 곧바로 우리 앱을 실행할 수 있다. 터미널로 돌아와 앱의 루트로 디렉터리를 변경한다.

```
$ cd angular-hello-world
$ ng serve
** NG Live Development Server is running on http://localhost:4200/ **
// ...
// 각종 메시지...
// ...
Webpack: Compiled successfully.
```

앱은 localhost 4200번 포트에서 실행되고 있다. 브라우저에서 http://localhost:4200에 접속한다.

다음과 비슷한 메시지가 출력되었다면 4200번 포트를 이미 다른 서비스가 사용하고 있다는 뜻이다.

```
Port 4200 is already in use. Use '--port' to specify a different port
```

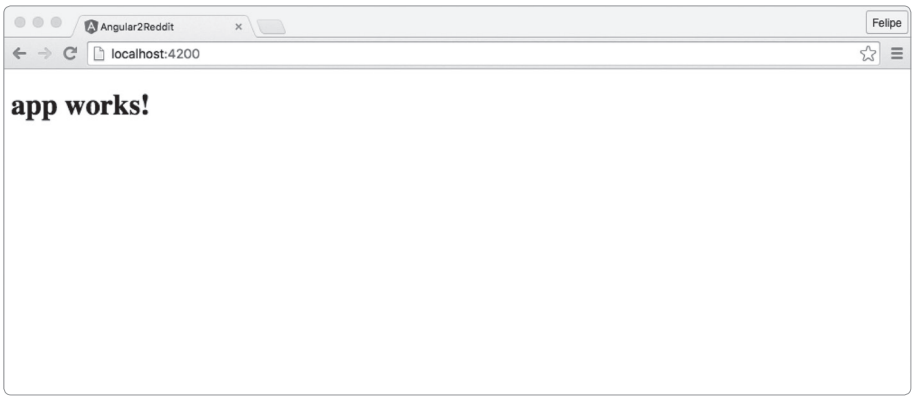
할 수 있는 일은 다음 두 가지다.

1. 그 서비스를 종료한다.
2. ng serve를 실행할 때 다음처럼 --port 플래그를 사용한다.

```
ng serve --port 9001
```

이 명령은 브라우저에서 열 URL을 http://localhost:9001처럼 변경한다.

주의할 점이 하나 더 있다. 일부 시스템에서는 localhost 도메인이 동작하지 않을 수도 있다. 127.0.0.1 형식의 숫자를 본 적이 있을 텐데, ng serve를 실행할 때는 서버가 어떤 URL에서 실행되는지 알아야 한다. 따라서 시스템에 출력되는 메시지를 주의 깊게 읽고 정확한 개발 환경의 URL을 파악해야 한다.



애플리케이션 실행하기

애플리케이션이 문제 없이 실행되었다. 실행 방법도 알았으니 본격적으로 코드를 작성해 보자.

■ Component 만들기

앵귤러를 떠받치는 가장 큰 구조적 개념이 바로 **컴포넌트(component)**다. 앵귤러 앱을 만들 때 인터랙티브 애플리케이션에 필요한 HTML 마크업들을 작성하는데, 사실 브라우저가 이해하는 마크업이 상당히 제한적이다. <select>나 <form>, <video> 등의 내장 마크업 태그들은 모두 브라우저 개발자가 정의해 놓은 기능들이다.

그런데 우리가 직접 브라우저에 새 태그를 학습시킬 수도 있을까? 이를테면 날씨를 알려 주는 <weather> 태그를 브라우저에 이해시킬 수 있을까? 또는 로그인 패널을 만드는 <login> 태그를 넣을 수 있을까? 이런 생각이 바로 컴포넌트로 구체화되었다. 맞춤형 기능을 제공하는 새로운 태그를 브라우저에 이해시키는 방법이 바로 컴포넌트다.

> 앵귤러JS 1.x 배경지식을 갖춘 사용자를 위해 한마디 덧붙이자면, **컴포넌트는 지시자(directive)의 새로운 버전이다.**

자, 첫 컴포넌트를 만들어 보자. 컴포넌트를 완성하면 다음처럼 app-hello-world 태그를 HTML 문서에서 사용할 수 있다.

```
<app-hello-world></app-hello-world>
```

앵귤러 CLI를 사용하여 새 컴포넌트를 만들려면 generate 명령이 필요하다. hello-world 컴포넌트를 만들려면 다음 명령을 실행해야 한다.

```
$ ng generate component hello-world
installing component
  create src/app/hello-world/hello-world.component.css
  create src/app/hello-world/hello-world.component.html
  create src/app/hello-world/hello-world.component.spec.ts
  create src/app/hello-world/hello-world.component.ts
```

새 컴포넌트는 어떻게 정의할까? 컴포넌트는 기본적으로 다음 두 부분으로 이루어진다.

1. Component 데코레이터
2. 컴포넌트 정의 클래스

컴포넌트 코드부터 하나씩 살펴보기로 하자. 우리의 첫 타입스크립트 파일인 src/app/hello-world/hello-world.component.ts를 연다.

code/first-app/angular-hello-world/src/app/hello-world/hello-world.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
export class HelloWorldComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

처음에는 이 스니펫(조각 코드)이 눈에 잘 안 들어오겠지만, 마음을 가다듬고 찬찬히 살펴보기로 하자.

> 타입스크립트 파일의 확장명은 .js가 아니라 .ts이다. 그런데 문제는 브라우저가 이 타입스크립트 파일을 이해하지 못한다는 점이다. 이 간극을 메우려면 .ts 파일을 .js 파일로 자동 변환하여 실시간 컴파일하는 ng serve 명령이 필요하다.

■ 의존성 파일 가져오기

import 구문은 코드를 작성하는 데 사용할 모듈을 정의한다. 여기서는 Component와 OnInit을 가져왔다.

'@angular/core' 모듈에서 Component를 가져왔다. 다시 말해 '@angular/core'가 의존성 파일을 찾을 위치다. 여기서는 '@angular/core'가 Component와 OnInit이라는 두 개의 자바스크립트/타입스크립트 객체를 정의하고 내보내라고 컴파일러에 알린 것이다.

마찬가지로, 같은 모듈에서 OnInit도 가져왔다. 나중에 다시 설명하겠지만, OnInit은 컴포넌트를 초기화할 때 코드가 실행되도록 한다. 지금은 그렇게만 이해하면 충분하다.

import의 구조는 import { things } from wherever이다. { things }에서 담당하는 것을 가리켜 **디스트럭처링(destructuring)**이라고 부른다. 디스트럭처링은 ES6와 타입스크립트가 제공하는 기능이며, 다음 장에서 자세하게 설명할 것이다. import는 자바의 import나 루비의 require와도 사뭇 닮았다. 다시 말해, 다른 모듈에서 의존성 파일을 끌고와 현재 파일에서 사용할 수 있는 상태로 만드는 과정이라 할 수 있다.

■ 컴포넌트 데코레이터

의존성 파일을 가져왔으면 컴포넌트를 선언해야 한다.

code/first-app/angular-hello-world/src/app/hello-world/hello-world.component.ts

```
3 @Component({
4   selector: 'app-hello-world',
5   templateUrl: './hello-world.component.html',
6   styleUrls: ['./hello-world.component.css']
7 })
```

타입스크립트가 처음인 독자는 다음 구조가 다소 괴상해 보일지도 모르겠다.

```
@Component({
  // ...
})
```

도대체 무슨 일을 하는 것일까? 이를 가리켜 **데코레이터(decorator)**라고 부른다. 데코레이터는 코드에 추가된 일종의 메타데이터다. HelloWorld 클래스에 @Component를 사용한 것은, 쉽게 말해 HelloWorld를 Component처럼 '장식'했다는 뜻이다.

<app-hello-world> 태그를 사용하여 마크업에서 이 컴포넌트를 사용하려면 @Component를 설정하고 selector를 app-hello-world로 지정한다.

```
@Component({
  selector: 'app-hello-world'
  // ... 필요한 내용
})
```

앵귤러의 컴포넌트 셀렉터는 CSS의 셀렉터와 비슷하다(다만, 앵귤러의 컴포넌트는 셀렉터에 특별한 문법 구조를 적용하고 있다. 이에 관해서는 잠시 뒤에 다룰 것이다). 지금은 마크업에 사용할 새 태그를 정의할 때 selector가 필요하다는 점만 이해하면 된다.

여기서 selector 프로퍼티는 컴포넌트에 어느 'DOM 요소'가 사용될지 가리킨다. 이에 따라 템플릿 안에 등장하는 <app-hello-world></app-hello-world> 태그는 모두 HelloWorldComponent 클래스 및 관련 정의들을 통해 컴파일된다.

■ templateUrl로 템플릿 추가하기

컴포넌트에서는 ./hello-world.component.html의 templateUrl을 지정했다. 다시 말해, 컴포넌트와 같은 디렉터리에 있는 hello-world.component.html 파일에서 템플릿이 로드된다. 이 파일의 내용은 다음과 같다.

code/first-app/angular-hello-world/src/app/hello-world/hello-world.component.html

```
<p>
  hello-world works!
</p>
```

여기서 p 태그를 정의하고 그 사이에 텍스트를 조금 넣었다. 앵귤러가 이 컴포넌트를 로드할 때 파일에서도 읽어 컴포넌트에 템플릿으로 사용한다.

■ template 추가하기

템플릿을 정의하는 방법은 두 가지다. @Component 객체에 template 키를 사용하여 정의할 수도 있고, templateUrl을 지정하여 정의할 수도 있다. @Component에 템플릿을 추가할 때는 다음처럼 template 옵션을 전달한다.

```
@Component({
  selector: 'app-hello-world',
  template: `
    <p>
      hello-world works inline!
    </p>
  `
})
```

백틱 사이(` ... `)에 template 문자열을 정의했다. 이 방법은 ES6의 새로운(그리고 환상적인) 기능으로서, 이렇게 하면 여러 줄의 문자열을 처리할 수 있다. 백틱을 사용하면 코드 파일에 템플릿을 넣기가 한결 수월해진다.

정말로 코드 파일에 템플릿을 넣어야 할까? 상황에 따라 다르다고 할 수 있다. 오랫동안 사람들은 코드와 템플릿을 따로 관리해야 한다고 믿었다. 이 믿음이 일부 개발팀에는 작업의 수고를 덜어 주었지만, 일부 프로젝트에서는 업무 과중으로 여겨졌다. 상당히 많은 파일들 사이를 왔다 갔다 해야 했기 때문이었다.

개인적으로는 우리 템플릿이 한 페이지 분량도 안 되기 때문에 코드 안에(즉 .ts 파일 안에) 함께 유지하는 것이 더 낫다고 생각한다. 로직이나 뷰를 함께 고려할 때 이 둘의 상호 작용 과정을 이해하기가 쉬워지기 때문이다.

뷰와 코드를 섞으면 비롯되는 가장 큰 단점은 (아직까지) 많은 에디터가 내부 문자열에는 문법 하이라이트 기능을 지원하지 않는다는 사실이다. 다행히 템플릿 문자열의 문법 하이라이트 HTML을 지원하는 에디터가 꾸준히 늘고 있다.

■ styleUrls로 CSS 스타일 추가하기

다음 styleUrls 키를 살펴보자.

```
styleUrls: ['./hello-world.component.css']
```

이 코드는 hello-world.component.css 파일에 컴포넌트의 스타일로 CSS를 사용하겠다는 뜻이다. 앵귤러는 **스타일 캡슐화**(style-encapsulation)라는 개념을 지원한다. 스타일 캡슐화는 특정 컴포넌트용으로 지정된 스타일은 다른 컴포넌트에 적용되지 않는다는 개념이다. 이에 관해서는 14장 '컴포넌트 고급'의 '스타일'에서 다시 자세하게 언급할 것이다. 지금은 컴포넌트 제한적인(component-local) 스타일을 하나도 사용하지 않을 것이다. 따라서 이 키를 통째로 삭제해도 되고 그대로 놔두어도 된다.

> 이 키는 배열을 인수로 받는다는 점에서 template과 다르다. 이는 여러 스타일시트를 하나의 컴포넌트에 로드할 수 있기 때문이다.

■ 컴포넌트 로드하기

첫 번째 컴포넌트 코드를 완성했다. 그렇다면 페이지에 어떻게 로드할까? 브라우저로 애플리케이션에 다시 방문하면 아직은 그대로일 것이다. 컴포넌트를 '만들기'만 했지 '사용'하지는 않았기 때문이다.

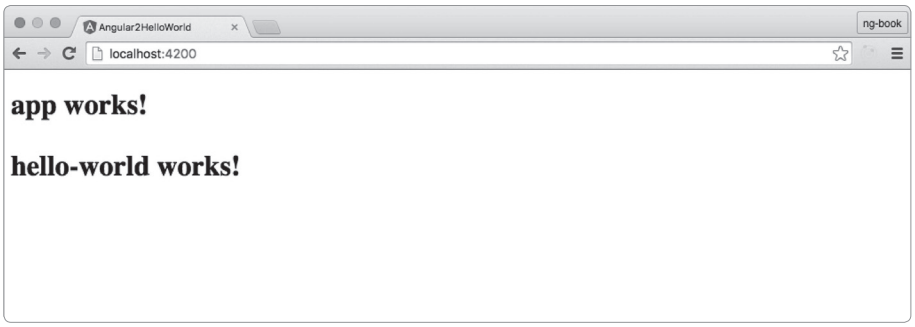
어떻게 사용해야 할까? 컴포넌트 태그를 이미 렌더링되고 있는 템플릿에 추가해야 한다. first_app/angular-hello-world/src/app/app.component.html 파일을 연다. HelloWorldComponent를 app-hello-world 셀렉터로 설정했기 때문에 템플릿에서 <app-hello-world></app-hello-world>를 사용할 수 있다. <app-hello-world> 태그를 app.component.html에 추가해 보자.

code/first-app/angular-hello-world/src/app/app.component.html

```
<h1>
  {{title}}

  <app-hello-world></app-hello-world>
</h1>
```

이제, 페이지를 새로 고침하고 결과를 살펴보자.



올바로 동작하는 모습

성공을 축하한다!

1.5 컴포넌트에 데이터 추가하기

지금은 우리의 컴포넌트가 정적인 템플릿을 렌더링한다. 그러니까 우리의 컴포넌트가 별로 흥미롭지 못하다는 뜻이다. '사용자 리스트'를 보여 주는 앱을 생각해 보자. 전체 리스트보다 개별 사용자부터 렌더링해야 할 것이다. 따라서 사용자 한 명의 이름을 보여 주는 새 컴포넌트를 만들어 보자. 새 컴포넌트를 만들려면 ng generate 명령이 필요하다.

```
ng generate component user-item
```

새로 만든 컴포넌트를 확인하려면 템플릿에 추가해야 했었다. 앞으로 변경되는 내용을 확인할 수 있도록 app-user-item 태그를 app.component.html에 추가한다. app.component.html을 다음처럼 수정한다.

code/first-app/angular-hello-world/src/app/app.component.html

```
<h1>
  {{title}}

  <app-hello-world></app-hello-world>

  <app-user-item></app-user-item>
</h1>
```

페이지를 새로 고침하면 user-item works! 텍스트가 올바로 보일 것이다.

UserItemComponent에서 특정 사용자의 이름을 보여 주려면 어떻게 해야 할까? 컴포넌트의 새 '프로퍼티'로 name을 도입하고자 한다. name 프로퍼티를 사용하면 컴포넌트를 다른 사용자에게도 재사용할 수 있다(마크업과 로직, 스타일은 그대로 유지된다). 이름을 추가하려면 UserItemComponent 클래스에 프로퍼티를 도입하여 name이라는 지역 변수를 선언해야 한다.

code/first-app/angular-hello-world/src/app/user-item/user-item.component.ts

```
8  export class UserItemComponent implements OnInit {
9    name: string; // <-- 이름 프로퍼티 추가
10
11   constructor() {
12     this.name = 'Felipe'; // 이름 설정
13   }
14
15   ngOnInit() {
16   }
17
18 }
```

여기서 우리는 두 가지를 변경했다.

1. name 프로퍼티

UserItemComponent 클래스에 '프로퍼티(property)'를 추가했다. 추가하는 문법은 ES5 자바스크립트에 비하면 비교적 새롭다. name: string;은 string 타입의 name 프로퍼티를 선언하겠다는 뜻이다.

변수에 타입을 지정하는 것은 '타입스크립트'에 이름을 지정한 것과 같다. name 프로퍼티의 타입을 string으로 지정하면 컴파일러는 name 변수가 string임을 파악한다. 만일, 이 프로퍼티에 number가 지정되면 오류가 발생된다.

이 문법은 또한 타입스크립트가 인스턴스 프로퍼티를 정의하는 방법이기도 하다. name: string;은 UserItemComponent의 모든 인스턴스에 name 프로퍼티를 지정한다.

2. 컨스트럭터

UserItemComponent 클래스에 **컨스트럭터(constructor)**, 즉 이 클래스의 새 인스턴스를 만들 때 호출되는 함수를 정의했다. 컨스트럭터에는 this.name을 사용하여 name 프로퍼티를 지정할 수 있다. 다음 코드를 살펴보자.

code/first-app/angular-hello-world/src/app/user-item/user-item.component.ts

```
11  constructor() {
12    this.name = 'Felipe'; // 이름 설정
13  }
```

새 UserItemComponent가 만들어질 때마다 그 이름이 'Felipe'로 설정된다.

1.5.1 템플릿 렌더링하기

컴포넌트에 프로퍼티가 만들어지면, 중괄호를 겹쳐 사용하는 템플릿 문법({{ }})을 적용하여 템플릿에서 그 값을 표시할 수 있다. 예를 들면 다음과 같다.

code/first-app/angular-hello-world/src/app/user-item/user-item.component.html

```
1 <p>
2   Hello {{ name }}
3 </p>
```

template에는 새 문법인 {{ name }}을 추가했다. 이중 중괄호의 이름은 **템플릿 태그(template tag)**이며, 흔히들 **콧수염 태그(mustache tag)**로도 부른다.

템플릿 태그 사이에 오는 것은 전부 '수식(expression)'으로 확장된다. 여기서는 template이 컴포넌트에 '바인딩'되었기 때문에 name이 this.name의 값, 즉 'Felipe'로 확장된다.

1.5.2 테스트하기

변경 내용을 적용하고 페이지를 다시 로드하면, Hello Felipe가 표시될 것이다.

데이터를 표시하는 애플리케이션

1.6 배열 다루기

한 명에게는 인사를 멋지게 건넸는데, 여러 명에게는 어떻게 인사할 수 있을까? 앵귤러에서는 *ngFor 문법을 사용하여 템플릿에서 객체 리스트를 반복 처리할 수 있다. 같은 마크업을 객체 컬렉션에 반복하자는 것이 기본 생각이다.

> 앵귤러JS 1.x를 사용해 본 독자들은 ng-repeat 지시자를 알고 있을 것이다. NgFor도 같은 방식으로 동작한다.

사용자 '리스트'를 렌더링할 새 컴포넌트를 만들어 보자. 우선, 새 컴포넌트를 만드는 코드는 다음과 같다.

```
ng generate component user-list
```

그리고 app.component.html 파일에서 <app-user-item> 태그를 <app-user-list>로 교체한다.

code/first-app/angular-hello-world/src/app/app.component.html

```
<h1>
  {{title}}

  <app-hello-world></app-hello-world>

  <app-user-list></app-user-list>
</h1>
```

UserItemComponent에 name 프로퍼티를 추가했던 방법 그대로 names 프로퍼티를 UserListComponent에 추가했다. 하지만 문자열 하나를 저장하는 대신 프로퍼티의 타입을 '문자열 배열'로 설정했다. 배열은 타입 다음에 [ ]로 표시하며 다음처럼 코드를 작성할 수 있다.

code/first-app/angular-hello-world/src/app/user-list/user-list.component.ts

```
export class UserListComponent implements OnInit {
  names: string[];

  constructor() {
    this.names = ['Ari', 'Carlos', 'Felipe', 'Nate'];
  }

  ngOnInit() {
  }
```

```
17
18 }
```

우선 UserListComponent 클래스의 새 string[ ] 프로퍼티를 주목해야 한다. 이 문법 구조는 names의 타입이 string들의 Array라는 것을 나타낸다. Array<string>이라는 문법 구조를 사용할 수도 있다. 여기서는 컨스트럭터를 변경하여 this.names의 값을 ['Ari', 'Carlos', 'Felipe', 'Nate']로 설정했다.

이제 템플릿을 업데이트해 명단을 렌더링할 수 있다. 업데이트에는 *ngFor를 사용할 것이며, 다음 두 과정으로 나눠 진행한다.

- 항목 리스트를 반복 처리한다.
- 항목마다 새 태그를 생성한다.

다음은 새 템플릿의 모습이다.

code/first-app/angular-hello-world/src/app/user-list/user-list.component.html

```
1 <ul>
2   <li *ngFor="let name of names">Hello {{ name }}</li>
3 </ul>
```

ul 하나와 li 하나만 있던 템플릿을 *ngFor="let name of names"라는 새 속성으로 업데이트했다. * 문자와 let 문법 때문에 주눅이 들 수도 있겠지만 하나씩 꼼꼼하게 따져 보기로 하자.

*ngFor는 이 속성에 NgFor 지시자를 사용하겠다는 뜻이다. NgFor는 for 루프의 닮은꼴이다. 컬렉션 항목 하나하나마다 새 DOM 요소를 만든다는 것이다.

값은 "let name of names"이다. 여기서 names는 HelloWorld 객체에 지정된 이름 배열이다. let name은 **참조(reference)**로 불린다. "let name of names"는 names의 각 요소를 반복하며 name이라는 '지역(local)' 변수에 지정한다는 뜻이다.

NgFor 지시자는 names 배열의 항목마다 하나의 li 태그를 렌더링하고, 반복되는 현재 항목을 담기 위해 name이라는 지역 변수를 선언한다. 이 새 변수가 Hello {{ name }} 스니펫 안에서 교체된다.

name이라는 참조 변수를 호출할 필요는 없었다. 사실 다음처럼 작성할 수도 있다.

```
<li *ngFor="let foobar of names">Hello {{ foobar }}</li>
```

하지만 거꾸로는 어떻게 될까? 막간 퀴즈 한 가지: 다음처럼 작성하면 어떻게 될까?

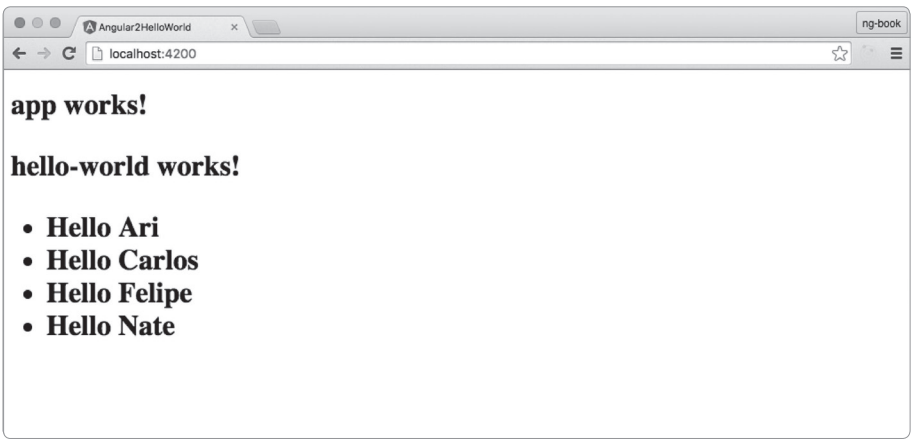
```
<li *ngFor="let name of foobar">Hello {{ name }}</li>
```

foobar는 컴포넌트의 프로퍼티가 아니기 때문에 오류가 발생한다.

NgFor는 ngFor가 호출되는 요소를 반복한다. 다시 말해, ul 태그가 아니라 li 태그에 들어간다. 리스트 자체(ul)가 아니라 리스트 요소(li)를 반복해야 하기 때문이다. NgFor와 ngFor는 다르다. NgFor는 해당 논리를 구현하는 클래스이고, ngFor는 사용할 속성에 해당하는 '셀렉터'다.

더 세부적인 지식을 원하는 독자들은 앵귤러 코어 팀의 컴포넌트 작성 과정이 고스란히 담긴 소스를 직접 살펴볼 수도 있을 것이다. 이를테면 NgFor 지시자의 소스는 https://github.com/angular/angular/blob/master/packages/common/src/directives/ng_for_of.ts에서 찾아볼 수 있다.

이제 페이지를 다시 로드하면 배열의 각 문자열에 li가 하나씩 있음을 확인할 수 있다.



데이터가 표시된 애플리케이션

1.7 UserItemComponent 사용하기

앞에서는 UserItemComponent를 만들었다. UserListComponent 안에서 각 이름을 렌더링하는 방법 대신, 우리는 UserItemComponent를 '자식 컴포넌트(child component)'로 사용해야 한다. 다시 말해, Hello 텍스트와 이름을 직접 렌더링하는 대신 UserItemComponent에서 리스트의 항목마다 템플릿(그리고 그 기능)을 지정해야 한다.

이를 위해 다음 세 과정을 거쳐야 한다.

1. (템플릿에서) UserItemComponent를 렌더링할 수 있도록 UserListComponent를 설정한다.
2. name 변수를 '입력'으로 받을 수 있도록 UserItemComponent를 설정한다.
3. UserItemComponent에 이름을 전달할 수 있도록 UserListComponent 템플릿을 설정한다.

이 세 단계를 하나하나 살펴보자.

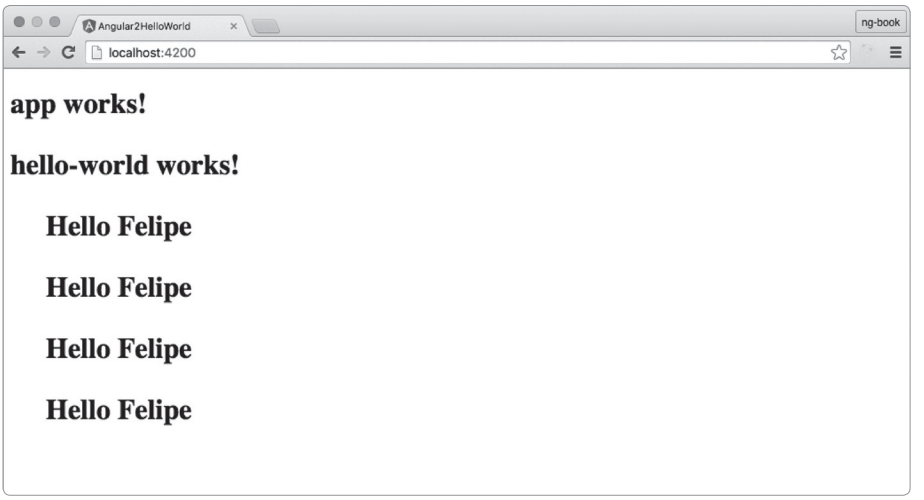
1.7.1 UserItemComponent 렌더링하기

UserItemComponent는 app-user-item 셀렉터를 지정한다. 이 태그를 템플릿에 추가해 보자.

code/first-app/angular-hello-world/src/app/user-list/user-list.component.html

```
<ul>
  <li *ngFor="let name of names">
    <app-user-item></app-user-item>
  </li>
</ul>
```

Hello 텍스트와 app-user-item 태그의 name을 서로 바꿨다. 브라우저를 다시 로드하면 화면은 다음과 같이 표시된다.



데이터가 표시된 애플리케이션

반복은 되는데 무언가 문제가 있다. 이름이 전부 'Felipe'가 아닌가? '데이터를 자식 컴포넌트에 전달'할 방법이 필요해 보인다.

1.7.2 입력 받기

UserItemComponent의 경우, constructor에서 this.name = 'Felipe';를 설정했었다. 지금은 이 컴포넌트를 변경하여 프로퍼티의 값을 받아야 한다. 다음은 UserItemComponent의 변경 내용이다.

code/first-app/angular-hello-world/src/app/user-item/user-item.component.ts

```
import {
  Component,
  OnInit,
  Input     // <--- 추가
} from '@angular/core';

@Component({
  selector: 'app-user-item',
  templateUrl: './user-item.component.html',
  styleUrls: ['./user-item.component.css']
})
export class UserItemComponent implements OnInit {
  @Input() name: string; // <-- Input 추가

  constructor() {
    // name 설정 제거
```

```
17    }
18
19    ngOnInit() {
20    }
21
22  }
```

@Input의 '데코레이터'를 가질 수 있도록 name 프로퍼티를 변경했다. Input과 Output은 다음 장에서 깊이 있게 언급할 것이다. 지금은 이 문법 구조를 적용하면 '부모 템플릿에서' 값을 전달할 수 있다는 점만 기억하기 바란다.

Input을 사용하려면 import에서 상수 리스트에 추가해야 한다. 그리고 name의 기본값을 사용할 생각이 아니므로 constructor에서 이를 제거한다. 이제 name Input이 생겼다. 사용은 어떻게 해야 할까?

1.7.3 입력값 전달하기

컴포넌트에 값을 전달하려면 템플릿에 '대괄호([ ])' 문법 구조를 적용한다. 다음은 업데이트된 템플릿이다.

code/first-app/angular-hello-world/src/app/user-list/user-list.component.html

```
1  <ul>
2    <li *ngFor="let name of names">
3      <app-user-item [name]="name"></app-user-item>
4    </li>
5  </ul>
```

app-user-item 태그에 [name]="name"이라는 새 속성을 추가했다. 앵귤러에서는 [foo]처럼 속성을 대괄호로 묶어 추가하면 foo라는 '입력'에 값을 전달한다는 뜻이다.

여기서 오른쪽에 있는 name이 ngFor의 let name ... 구문에서 나온 것이라는 점을 눈여겨보아야 한다. 다시 말해, 다음처럼 생각할 수도 있다.

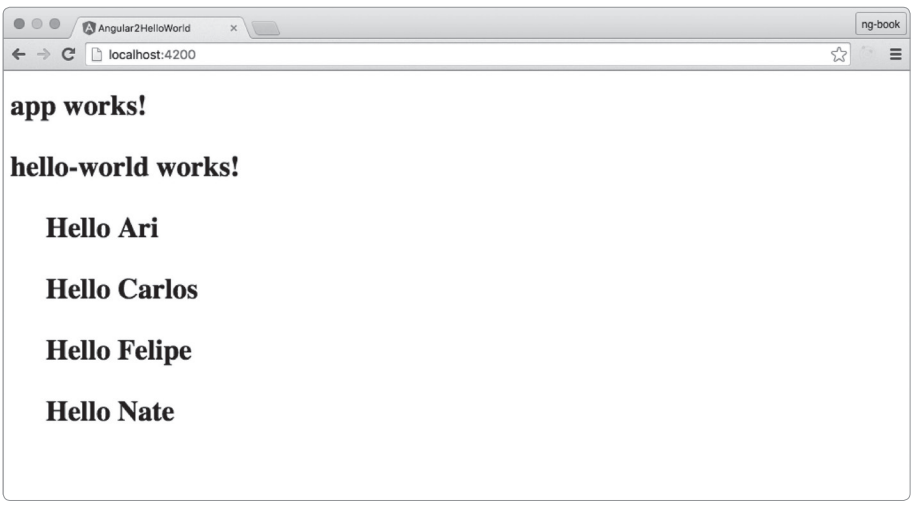
```
1    <li *ngFor="let individualUserName of names">
2      <app-user-item [name]="individualUserName"></app-user-item>
3    </li>
```

[name] 부분은 UserItemComponent의 Input을 가리킨다. "individualUserName"이라는 리터럴 문자열 그대로를 전달하지 않는 것이다. 전달하는 것은 individualUserName의 '값(value)'이다. 다시 말해, names의 요소의 값을 매번 전달하는 것이다.

입력과 출력에 관한 세부 내용은 다음 장으로 미룰 것이다. 지금은 다음 내용만 알고 있으면 된다.

1. names 반복 처리하기
2. names의 각 요소에 새 UserItemComponent 만들기
3. 이름의 값을 UserItemComponent의 name Input 프로퍼티에 전달하기

이제 이름 리스트가 올바로 동작한다!



이름들이 올바로 표시된 애플리케이션

축하한다! 컴포넌트로 첫 앵귤러 앱을 완성했다! 물론, 이 앱은 단순하기 그지없다. 이보다는 훨씬 더 정교한 앱을 만들고 싶은 생각이 스멀스멀 피어오르겠지만 조급해 할 이유가 없다. 앞으로 이 책을 통해 전문가 수준의 앵귤러 앱을 어떻게 만드는지 설명을 이어갈 것이다. 사실 이 장에서는 투표 앱(레딧이나 프로덕트 헌트 등)을 만들 것이다. 이 앱의 특징은 사용자 인터랙션과 더 많은 컴포넌트다! 하지만 그 전에 앵귤러 앱이 어떻게 부트스트랩되는지 살펴보기로 하자.

1.8 부트스트랩 속성 과정

앱에는 저마다 1차 진입점이 있다. 우리의 앱은 웹팩(Webpack)이라는 도구를 바탕으로 만들어진 앵귤러 CLI를 사용하여 빌드했다. 우리의 앱은 다음 명령으로 실행한다.

```
ng serve
```

ng는 .angular-cli.json 파일에서 앱의 진입점을 찾는다. ng가 컴포넌트를 어떻게 찾는지 추적해 보자. 다음 과정처럼 생각해 볼 수 있다.

- .angular-cli.json이 "main" 파일을 지정한다. 여기서는 main.ts이다.
- main.ts가 앱의 진입점이며 앱을 '부트스트랩(bootstrap)'한다.
- 부트스트랩 프로세스가 **앵귤러 모듈**을 부트한다. 모듈은 잠시 뒤에 다룰 것이다.
- AppModule을 사용하여 앱을 부트스트랩한다. AppModule은 src/app/app.module.ts에 지정돼 있다.
- AppModule은 어떤 '컴포넌트'를 최상위 컴포넌트로 사용할지 지정한다. 여기서는 AppComponent이다.
- AppComponent는 템플릿에 <app-user-list> 태그를 가지며, 이 태그가 사용자 리스트를 렌더링한다.

지금은 NgModule이라는 앵귤러 모듈 시스템에 초점을 맞출 것이다. 앵귤러는 '모듈(module)'이라는 강력한 개념을 제공한다. 앵귤러 앱을 부트할 때 컴포넌트까지 직접 부트하지는 않지만, 그 대신 로드할 컴포넌트를 가리키는 NgModule을 만든다. 다음 코드를 살펴보자.

code/first-app/angular-hello-world/src/app/app.module.ts

```
@NgModule({
  declarations: [
    AppComponent,
    HelloWorldComponent,
    UserItemComponent,
    UserListComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
```

```
24   bootstrap: [AppComponent]
25 })
26 export class AppModule {  }
```

우선 @NgModule 데코레이터가 보인다. 이 @NgModule(...) 코드는 다른 데코레이터처럼 메타데이터를 바로 다음에 있는 클래스(AppModule)에 추가한다.

@NgModule 데코레이터에는 키가 모두 네 개 있다. declarations, imports, providers, bootstrap 이다.

1.8.1 declarations

declarations는 이 모듈에 정의된 컴포넌트를 지정한다. 이는 앵귤러에서 중요한 기본 개념이기도 하다. 정리하자면 다음과 같다.

컴포넌트를 템플릿에서 사용하려면, 먼저 NgModule에서 정의부터 해야 한다. NgModule은 일종의 '패키지'이며 declarations는 모듈이 어떤 컴포넌트들을 '소유'했는지 나타낸다.

앞에서 살펴본 바로는 ng generate를 사용하면 컴포넌트가 자동으로 declarations 리스트에 추가되었다! ng 도구로 새 컴포넌트를 만들면, 이 컴포넌트는 현재 NgModule의 소유가 된다는 것이다.

1.8.2 imports

imports는 모듈이 어떤 '의존성 파일'들을 가지고 있는지 기술한다. 지금은 브라우저 앱을 만들고 있으므로 BrowserModule을 가져온 것이다. 모듈이 다른 모듈에 의존한다면 여기에 기술해야 한다.

import vs. imports? 이런 의문이 들 수 있다. "파일 상단에서 클래스를 import하는 것과 모듈을 imports에 두는 것이 뭐가 다르지?" 핵심만 말하자면, 템플릿에서 사용하겠다거나 '의존성 주입(dependency injection)'과 함께 사용할 것들은 NgModule의 imports에 두어야 한다. 의존성 주입이 무엇인지는 곧 다룰 것이다.

1.8.3 providers

providers는 의존성 주입에 사용된다. 어떤 서비스를 앱에서 사용할 수 있도록 주입하려면 여기에 추가해야 한다.

세부 내용은 6장을 참고하기 바란다.

1.8.4 bootstrap

bootstrap은 모듈이 앱의 부트스트랩에 사용될 때 AppComponent 컴포넌트를 최상위 컴포넌트로 로드해야 한다는 것을 나타낸다.

1.9 애플리케이션 확장하기

지금까지 기본적인 애플리케이션 제작 과정을 다루었다. 이제 레딧 클론을 본격적으로 빌드해 보자. 코딩에 앞서 앱을 찬찬히 살펴보고 논리적인 단위 컴포넌트로 나눠 보자.

데이터가 표시된 애플리케이션

앱에서는 두 개의 컴포넌트를 만들 것이다.

1. 애플리케이션 전체에 해당하는 컴포넌트. 새 글을 제출하는 데 사용될 폼(그림에서 옅은 점선 부분)이 포함된다.
2. 각 글(진한 실선 부분)

애플리케이션이 커지면 글 제출 **폼(form)**도 컴포넌트로 처리하는 것이 일반적이다. 하지만 폼이 컴포넌트가 되면 데이터 전달 과정이 더 복잡해진다. 따라서 이 장에서는 구조를 단순화하여 컴포넌트는 두 개만 구현하기로 하겠다. 지금은 두 개의 컴포넌트만 만들지만, 이 책 뒷부분에서 좀 더 정교한 데이터 아키텍처에 관해 깊이 있게 다룰 것이다.

우선 앞에서 새 애플리케이션을 만들기 위해 실행했던 동일한 **ng new** 명령을 실행하고, 만들려는 앱의 이름을 전달하여 새 애플리케이션을 만들어 보자(여기서는 angular-reddit이라는 애플리케이션을 만들 것이다).

```
ng new angular-reddit
```

angular-reddit의 완성 버전은 예시 코드 다운로드로도 제공하고 있다. 이를 통해 더욱 상세한 구조를 알 수 있을 것이다.

1.9.1 CSS 추가하기

우선 CSS 스타일을 추가하여 앱의 스타일을 정리할 필요가 있다.

앱을 처음부터 새로 빌드한다면 first_app/angular-reddit 폴더에 있는 완성된 몇 가지 파일을 복사해야 한다. 구체적으로는 다음 파일들을 애플리케이션 폴더에 복사한다.

- src/index.html
- src/styles.css
- src/app/vendor
- src/assets/images

이번 프로젝트에서는 Semantic UI[7]을 사용하여 스타일을 구현한다. Semantic UI는 저브 재단(Zurb Foundation)[8]이나 트위터 부트스트랩(Twitter Bootstrap)[9]과 유사한 CSS 프레임워크다. 여기서는 위에서 언급한 파일들을 복사만 하면 되도록 샘플 코드 다운로드에 함께 포함했다.

7 http://semantic-ui.com/
8 http://foundation.zurb.com
9 http://getbootstrap.com

1.9.2 애플리케이션 컴포넌트

이제 다음 일을 할 새 컴포넌트를 빌드해 보자.

1. 글의 현재 리스트를 저장한다.
2. 새 글 제출 폼을 포함한다.

메인 애플리케이션 컴포넌트는 src/app/app.component.ts 파일에서 찾을 수 있다. 이 파일을 다시 연다. 앞에서 본 초기 내용물을 다시 확인할 수 있을 것이다.

code/first-app/angular-reddit/src/app/app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works!';
}
```

title 프로퍼티는 AppComponent에 자동으로 생성된다. title을 사용하지 않으므로 이 행을 삭제한다. 그 밑으로도 '제목'(title)이 있는 새 링크를 제출할 것인데, 혹시 앵귤러 CLI가 자동으로 생성한 AppComponent 제목과 혼동할지도 모르겠다. 폼 아래에 제출하는 새 링크에 '제목'을 추가하면 이 제목은 별도의 폼 필드다.

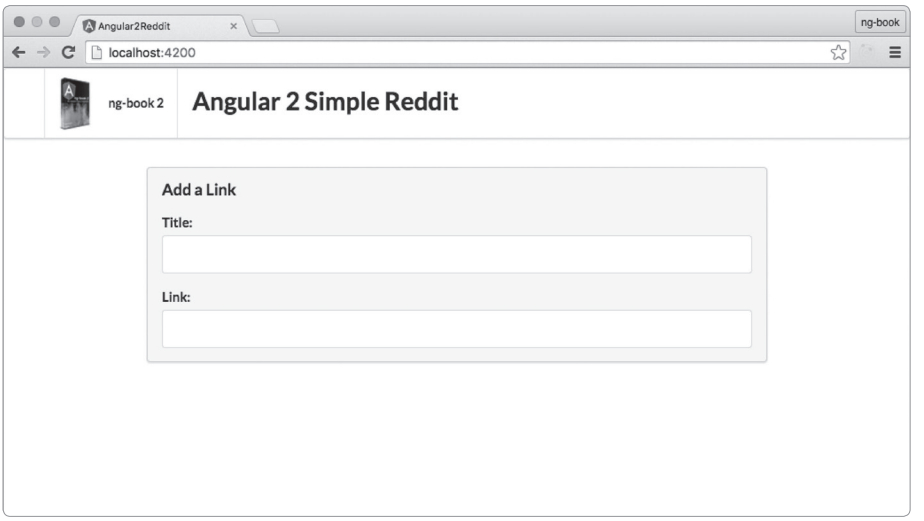
링크 추가 폼을 포함할 수 있도록 템플릿을 조금 바꾸기로 하자. Semantic UI 패키지의 스타일 몇 가지를 사용하면 폼을 더 멋지게 꾸밀 수 있다.

code/first-app/angular-reddit/src/app/app.component.html

```
<form class="ui large form segment">
  <h3 class="ui header">Add a Link</h3>

  <div class="field">
    <label for="title">Title:</label>
    <input name="title">
  </div>
  <div class="field">
    <label for="link">Link:</label>
    <input name="link">
  </div>
</form>
```

글의 title과 link URL, 이렇게 두 개의 input 태그를 정의하는 template을 만들 것이다. 브라우저를 로드하면 다음과 같이 렌더링된 폼이 나타날 것이다.



데이터가 표시된 애플리케이션

1.9.3 인터랙션 추가하기

폼에 입력 태그를 두었지만 데이터를 제출할 방법은 아직 마련하지 않았다. 폼에 제출 버튼을 추가하여 인터랙션을 구현해 보자.

폼이 제출되면 링크를 만들고 추가할 함수를 호출해야 한다. <button /> 요소에 인터랙션 이벤트를 추가하면 이 함수를 호출할 수 있다. 이벤트 이름을 괄호, 즉 ( )로 묶으면 이벤트에 응답할 수 있다. 이를테면 <button /> onClick 이벤트에 함수 호출을 추가하려면 다음처럼 전달한다.

```
<button (click)="addArticle()"
        class="ui positive right floated button">
  Submit link
</button>
```

이제 버튼이 클릭되면 addArticle( )이라는 함수가 호출된다. 이 함수는 다음과 같이 AppComponent 클래스에 정의해야 한다.

code/first-app/angular-reddit/src/app/app.component.ts

```
8  export class AppComponent {
9    addArticle(title: HTMLInputElement, link: HTMLInputElement): boolean {
10     console.log(`Adding article title: ${title.value} and link: ${link.value}`);
11     return false;
12   }
13 }
```

addArticle() 함수를 AppComponent에 추가하고 (click) 이벤트를 <button /> 요소에 추가하면, 이 함수는 버튼이 클릭될 때 호출된다. addArticle() 함수는 두 개의 인수를 받는다. 하나는 title이고, 또 하나는 link이다. 그리고 이 두 인수를 addArticle() 호출로 전달하도록 템플릿 버튼을 변경해야 한다.

이를 위해 폼의 input 요소에 특별한 문법 구조를 적용하여 **템플릿 변수**(template variable)라는 것을 만든다. 다음은 템플릿의 모습이다.

code/first-app/angular-reddit/src/app/app.component.html

```
1  <form class="ui large form segment">
2    <h3 class="ui header">Add a Link</h3>
3
4    <div class="field">
5      <label for="title">Title:</label>
6      <input name="title" #newtitle> <!-- 변경 -->
7    </div>
8    <div class="field">
9      <label for="link">Link:</label>
10     <input name="link" #newlink> <!-- 변경 -->
11   </div>
12
13   <!-- 버튼 추가 -->
14   <button (click)="addArticle(newtitle, newlink)"
15           class="ui positive right floated button">
16     Submit link
17   </button>
18
19 </form>
```

input 태그에 #(해시)를 사용하여 태그들을 '지역 변수'에 지정한 부분이 중요하다. #title과 #link를 적당한 <input /> 요소에 추가하면 이들을 변수로 addArticle() 함수에 전달할 수 있다.

정리하자면, 우리는 모두 네 군데를 변경했다.

1. 사용자에게 클릭할 곳을 나타내는 button 태그를 마크업에 만들었다.
2. 버튼이 클릭되면 해야 할 일을 정의하는 addArticle 함수를 만들었다.
3. 버튼이 클릭되면 addArticle 함수를 호출하라는 내용의 (click) 속성을 button에 추가했다.
4. #newtitle과 #newlink 속성을 <input> 태그에 추가했다.

이제 4번부터 거꾸로 하나씩 살펴보자.

■ input을 값에 바인딩하기

첫 번째 입력 태그는 다음과 같다.

```
<input name="title" #newtitle>
```

이 마크업은 <input>을 newtitle 변수에 '바인딩'한다. #newtitle 문법을 가리켜 **resolve**라 부른다. 이 문법을 적용하면 newtitle 변수를 뷰 안에서도 사용할 수 있다.

newtitle은 input DOM 요소를 나타내는 **객체(object)**다(구체적으로 말하자면 타입이 HTMLInputElement이다). newtitle은 객체이기 때문에 newtitle.value를 사용하여 입력 태그의 값을 다룰 수 있다.

마찬가지로, #newlink를 다른 <input> 태그에 추가했다. 따라서 같은 방법으로 값을 다룰 수 있다.

■ 액션을 이벤트에 바인딩하기

button 태그에는 버튼이 클릭되면 어떤 일이 일어나야 하는지 정의하기 위해 (click) 속성을 추가했다. 그리고 (click) 이벤트가 일어나면 newtitle과 newlink라는 두 개의 인수로 addArticle을 호출했다. 이 함수와 두 개의 인수는 어디서 온 것일까?

1. addArticle은 컴포넌트 정의 클래스인 AppComponent의 함수다.
2. newtitle은 title이라는 이름의 <input> 태그의 해결(#newtitle)에서 만들어졌다.
3. newlink는 link라는 이름의 <input> 태그의 해결(#newlink)에서 만들어졌다.

종합하면 다음과 같다.

```
<button (click)="addArticle(newtitle, newlink)"
         class="ui positive right floated button">
  Submit link
</button>
```

class="ui positive right floated button" 마크업은 Semantic UI에서 만들어졌으며 버튼에 색을 입힌다.

■ 액션 로직 정의하기

AppComponent에서는 새로운 함수인 addArticle을 정의했다. 이 함수의 인수는 title과 link이다. 다시 말하지만, title과 link가 '직접 입력하는 값이 아니라' HTMLInputElement 타입의 **객체**라는 점을 이해해야 한다. input으로부터 값을 가져오려면 title.value를 호출해야 한다. 지금은 인수들을 console.log로 처리한다.

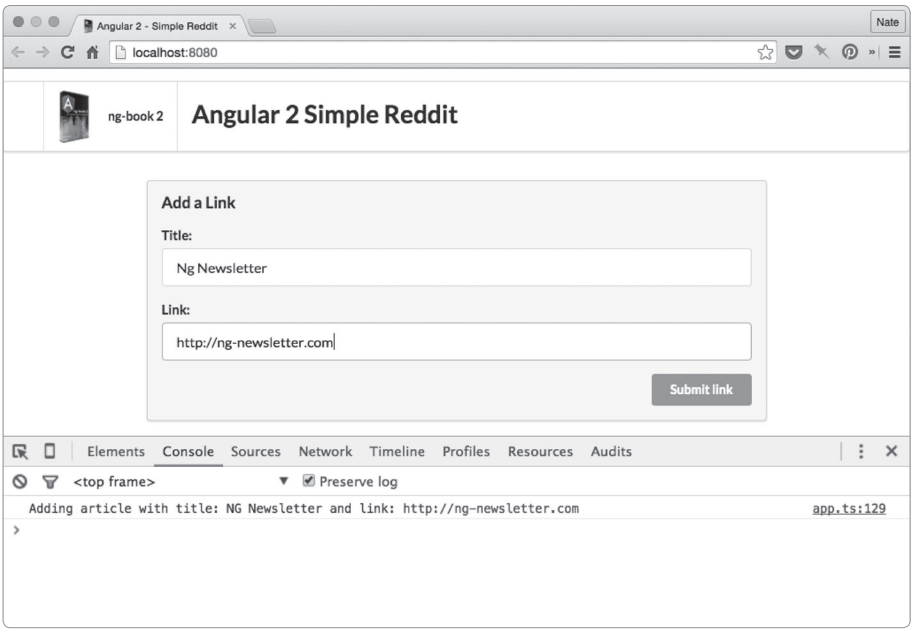
code/first-app/angular-reddit/src/app/app.component.ts

```
  addArticle(title: HTMLInputElement, link: HTMLInputElement): boolean {
    console.log(`Adding article title: ${title.value} and link: ${link.value}`);
    return false;
  }
```

백틱 문자열을 다시 사용했다. ES6의 뛰어난 기능 가운데 하나인 백틱 문자열은 템플릿 변수를 확장한다. 여기서는 문자열에 ${title.value}를 두었으며, 이렇게 하면 문자열에서 title.value의 값으로 교체된다.

■ 테스트하기

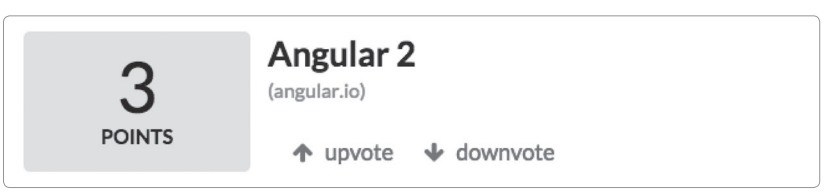
이제 제출 버튼을 클릭하면 메시지가 콘솔에 출력될 것이다.



버튼 클릭하기

1.9.4 ArticleComponent 추가하기

이제 새 글을 제출할 폼이 완성되었다. 하지만 새 글을 어느 곳에도 나타내지 않았다. 제출된 모든 글은 페이지에서 리스트 형태로 표시되기 때문에 새 컴포넌트를 만들 좋은 재료라 할 수 있다. 제출된 개별 글들을 나타내기 위해 새 컴포넌트를 만들어 보자.



레딧 게시물

ng 도구를 사용하여 새 컴포넌트를 만들어 보자.

```
ng generate component article
```

세 부분으로 나눠 새 컴포넌트를 정의해야 한다.

1. 템플릿에서 ArticleComponent 뷰를 정의한다.
2. @Component와 클래스의 어노테이션을 통해 ArticleComponent 프로퍼티를 정의한다.
3. 컴포넌트 로직을 담은 컴포넌트 정의 클래스(ArticleComponent)를 정의한다.

하나씩 세부적으로 살펴보자.

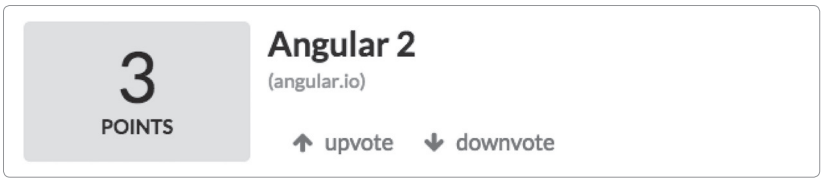
■ ArticleComponent template 만들기

다음과 같이 article.component.html 파일을 사용하여 템플릿을 정의한다.

code/first-app/angular-reddit/src/app/article/article.component.html

```
<div class="four wide column cente aligned votes">
  <div class="ui statistic">
    <div class="value">
      {{ votes }}
    </div>
    <div class="label">
      Points
    </div>
  </div>
</div>
<div class="twelve wide column">
  <a class="ui large header" href="{{ link }}">
    {{ title }}
  </a>
  <ul class="ui big horizontal list voters">
    <li class="item">
      <a href (click)="voteUp()">
        <i class="arrow up icon"></i>
          upvote
        </a>
    </li>
    <li class="item">
      <a href (click)="voteDown()">
        <i class="arrow down icon"></i>
        downvote
      </a>
    </li>
  </ul>
</div>
```

마크업이 상당히 많은데 하나씩 살펴보기로 하자.



한 줄짜리 레딧 게시물 리스트

반면, 열은 두 개다.

1. 왼쪽에 있는 투표 수
2. 오른쪽에 있는 게시물 정보

이 두 열에 CSS 클래스인 four wide column과 twelve wide column을 각각 지정했다(둘 다 Semantic UI의 CSS이다).

votes와 title은 템플릿 확장 문자열인 {{ votes }}와 {{ title }}로 표시했다. 해당 값은 ArticleComponent 클래스의 votes 및 title 프로퍼티의 값이며, 정의는 잠시 뒤로 미루겠다.

a tag: href="{{ link }}"의 href처럼 **속성 값(attribute value)**에 템플릿 문자열을 사용할 수 있다는 점에 주목해야 한다. 여기서 href의 값은 컴포넌트 클래스의 link 값에 따라 동적으로 정해진다.

'좋아요/싫어요(upvote/downvot)' 링크에도 액션을 지정해야 한다. 여기서는 (click)을 사용하여 voteUp( )/voteDown( )을 각 버튼에 바인딩했다. 업보트(좋아요) 버튼이 클릭되면 ArticleComponent 클래스의 voteUp( ) 함수가 호출된다(다운보트(싫어요) 버튼이 클릭되면 voteDown( ) 함수가 호출되는 것도 마찬가지다).

■ ArticleComponent 만들기

code/first-app/angular-reddit/src/app/article/article.component.ts

```
7  @Component({
8    selector: 'app-article',
9    templateUrl: './article.component.html',
10   styleUrls: ['./article.component.css'],
11 })
```

우선 @Component로 새 컴포넌트를 정의했다. selector는 이 컴포넌트의 페이지 내 위치가 <app-article> 태그로 나타난다는 뜻이다(즉, 셀렉터는 태그의 이름이다). 따라서 이 컴포넌트를

사용하려면 마크업에 다음 태그를 두어야 한다.

```
<app-article>
</app-article>
```

이 태그는 페이지가 렌더링될 때도 뷰에 그대로 남는다.

■ ArticleComponent 정의 클래스 만들기

마지막으로, ArticleComponent 정의 클래스를 만들어야 한다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
export class ArticleComponent implements OnInit {
  @HostBinding('attr.class') cssClass = 'row';
  votes: number;
  title: string;
  link: string;

  constructor() {
    this.title = 'Angular 2';
    this.link = 'http://angular.io';
    this.votes = 10;
  }

  voteUp() {
    this.votes += 1;
  }

  voteDown() {
    this.votes -= 1;
  }

  ngOnInit() {
  }

}
```

여기서는 ArticleComponent에 프로퍼티 네 개를 만들었다.

1. cssClass — 이 컴포넌트의 '호스트'에 적용할 CSS class
2. votes — 다운보트(싫어요)를 뺀 업보트(좋아요)의 총합을 나타내는 number
3. title — 게시물의 제목을 담은 string
4. link — 게시물의 URL을 담은 string

app-article마다 행이 있어야 한다. 우리가 사용하고 있는 Semantic UI에는 행을 나타내기 위한 row라는 CSS 클래스[10]가 제공된다.

앵귤러에서 컴포넌트 **호스트(host)**는 컴포넌트가 연결되는 요소를 말한다. 호스트 요소의 프로퍼티는 @HostBinding( ) 데코레이터를 사용하여 설정할 수 있다. 여기서는 호스트 요소 클래스의 값이 cssClass 프로퍼티와 동기화되도록 했다.

HostBinding은 @angular/core에서 가져온다. 이를테면 다음처럼 HostBinding을 추가할 수 있다.

```
import { Component, OnInit, HostBinding } from '@angular/core';
```

@HostBinding( )을 **호스트 요소**(app-article 태그)로 사용하여 class 속성을 "row"로 설정했다.

@HostBinding( )을 사용하면 컴포넌트 '안에서' app-article 마크업을 캡슐화할 수 있어 여러모로 유용하다. 다시 말해, app-article 태그를 사용할 필요도 없고, 부모 뷰의 마크업에서 class="row"가 필요하지도 않다. @HostBinding 데코레이터를 사용하면 컴포넌트 '안에서' 호스트 요소를 설정할 수 있다.

constructor( )에서 기본 속성 몇 가지를 설정했다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
  constructor() {
    this.title = 'Angular 2';
    this.link = 'http://angular.io';
    this.votes = 10;
  }
```

그리고 투표를 위한 함수 둘을 정의했다. 하나는 '좋아요'에 해당하는 voteUp이고 다른 하나는 '싫어요'에 해당하는 voteDown이다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
  voteUp() {
    this.votes += 1;
  }

```

10 http://semantic-ui.com/collections/grid.html 단축URL goo.gl/huCCZq

```
  voteDown() {
    this.votes -= 1;
  }
```

voteUp에서는 this.votes가 1씩 증가한다. 마찬가지로 voteDown에서도 1씩 감소한다.

■ app-article 컴포넌트 사용하기

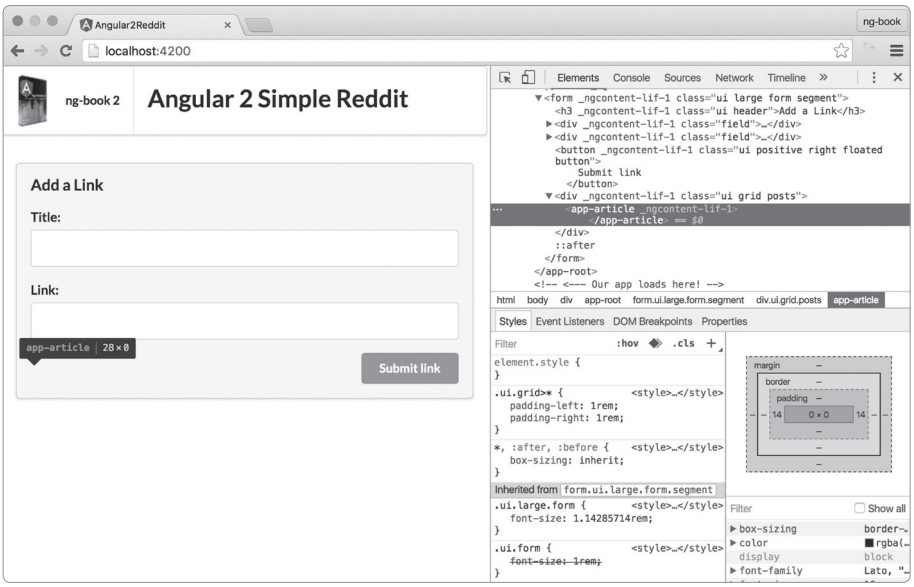
이 컴포넌트를 사용하여 데이터를 나타내려면 마크업 어딘가에는 <app-article></app-article> 태그를 추가해야 한다. 여기서는 AppComponent로 새 컴포넌트를 렌더링해야 한다. 따라서 그에 맞게 코드를 업데이트해 보자. <app-article> 태그를 </form> 태그 바로 다음에 있는 AppComponent의 템플릿에 추가한다.

```
  <button (click)="addArticle(newtitle, newlink)"
          class="ui positive right floated button">
    Submit link
  </button>
</form>

<div class="ui grid posts">
  <app-article>
  </app-article>
</div>
```

앵귤러 CLI를 사용하여 ArticleComponent를 생성하면(ng generate component를 통해) 기본적으로는 app-article 태그를 앵귤러에 '알렸어야' 했다(세부 내용은 잠시 뒤에 다룬다). 하지만 컴포넌트를 '직접' 만들었기 때문에 지금 브라우저를 다시 로드해 보면 <app-article> 태그가 컴파일되지 않았을 것이다. 이런 맙소사!

이런 문제가 생기면 가장 먼저 브라우저의 개발자 콘솔부터 열어야 한다. 마크업(아래 그림 참고)에서 <app-article> 태그가 페이지에 있기는 하지만 마크업으로 컴파일되지는 않았다. 왜 컴파일되지 않았을까?



DOM의 확장되지 않은 태그

이 문제는 AppComponent 컴포넌트가 ArticleComponent 컴포넌트를 아직 알지 못하기 때문에 일어났다.

> **앵귤러 1 참고:** 앵귤러 1 사용자는 앱이 아직 app-article 컴포넌트를 모른다는 사실에 놀라워할지도 모르겠다. 앵귤러 1의 지시자는 앱 전체에서 일관성을 가져야 하기 때문이다. 하지만 앵귤러에서는 어느 컴포넌트를 사용할지 (그에 따른 셀렉터를) 명시적으로 지정해야 한다.
>
> 한편으로는 이 때문에 몇 가지를 더 설정해야 하지만, 다르게 생각하자면 스케일이 유연한 앱을 만들 때는 훨씬 더 편해지는 것도 사실이다. 셀렉터를 네임스페이스 전체에서 공유할 필요가 없어졌기 때문이다.

AppComponent에 새 ArticleComponent 컴포넌트를 알리려면 ArticleComponent를 NgModule의 declarations 리스트에 추가해야 한다.

> ArticleComponent를 declarations에 추가했다. ArticleComponent가 AppModule의 일부분이기 때문이다. 하지만 ArticleComponent가 '다른' 모듈의 일부분이라면 imports로 가져와야 한다.
>
> NgModule에 관해서는 나중에 다시 자세하게 다룰 것이다. 지금은 새 컴포넌트를 만들려면 NgModule의 declarations에 추가해야 한다는 것을 아는 것으로 충분하다.

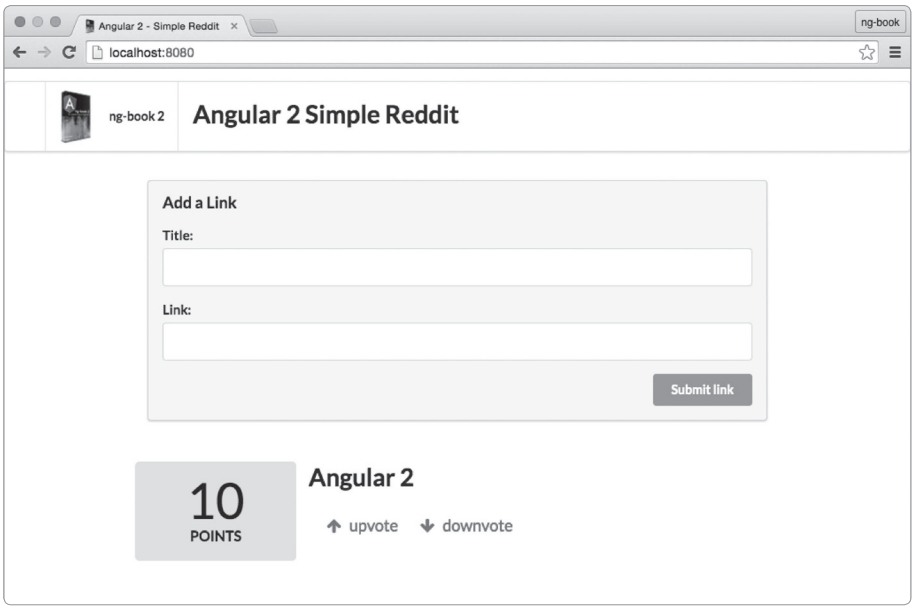
code/first-app/angular-reddit/src/app/app.module.ts

```
6  import { AppComponent } from './app.component';
7  import { ArticleComponent } from './article/article.component';
8
9  @NgModule({
10   declarations: [
11     AppComponent,
12     ArticleComponent // <-- 추가
13   ],
```

여기서 우리가 한 일은 다음과 같다.

1. ArticleComponent 가져오기
2. declarations 리스트에 ArticleComponent 추가하기

ArticleComponent를 NgModule의 declarations에 추가하고 브라우저를 다시 로드하면 게시물이 올바로 렌더링된 것을 확인할 수 있을 것이다.



렌더링된 ArticleComponent 컴포넌트

하지만 업보트나 다운보트 링크를 클릭하면 게시물 리스트가 업데이트되지 않고 페이지가 다시 로드된다. 자바스크립트는 기본적으로 click 이벤트를 모든 부모 컴포넌트로 전파한다. 이 때문에 브라우저는 빈 링크를 따라 가려고 하고, 그 결과 브라우저에 페이지가 다시 로드된다.

이 문제를 해결하려면 클릭 이벤트 핸들러가 false를 리턴하도록 해야 한다. 그래야 브라우저가 페이지를 새로 고치지 않을 것이다. voteUp()과 voteDown() 함수가 false라는 불리언 값을 리턴하도록(클릭 이벤트를 부모 컴포넌트로 전파하지 않도록) 코드를 업데이트해 보자.

```
voteDown(): boolean {
  this.votes -= 1;
  return false;
}
// `voteUp()`도 마찬가지
```

이제 링크들을 클릭하면 페이지 새로 고침 없이 투표 개수가 제대로 증가하거나 감소할 것이다.

1.10 여러 행 렌더링하기

지금은 페이지에 게시물 하나만 나타내었다. <app-article> 태그를 추가로 두는 것 이외에는 달리 렌더링할 방법이 없었던 탓이다. <app-article> 태그를 추가로 더 둔다고 해도 게시물이 전부 같은 내용을 가지게 되므로 온전한 방법은 아니다.

1.10.1 Article 클래스 만들기

앵귤러 코드를 효율적으로 작성할 수 있는 좋은 방법이 한 가지 있다. 데이터 구조를 컴포넌트 코드와 분리하는 것이다. 그러기 위해서는 우선 하나의 게시물을 나타내는 데이터 구조를 만들어야 한다. Article 클래스를 정의하기 위해 article.model.ts라는 새 파일을 추가한다.

code/first-app/angular-reddit/src/app/article/article.model.ts

```
export class Article {
  title: string;
  link: string;
  votes: number;

  constructor(title: string, link: string, votes?: number) {
    this.title = title;
    this.link = link;
    this.votes = votes || 0;
  }
}
```

Article이라는 새 클래스를 만들었다. 다만, 이 클래스는 평범한 클래스이지 앵귤러 컴포넌트는 아니다. 모델-뷰-컨트롤러 패턴에서 보자면 이는 **모델(Model)**이라 할 수 있다.

각 게시물에는 title과 link, 전체 votes 수가 있다. 그리고 새 게시물에는 title과 link가 필요하다. votes 파라미터는 옵션이며(이름 끝에 ?를 붙임) 기본값은 0이다.

이제 ArticleComponent 코드를 새 Article 클래스로 업데이트해 보자. ArticleComponent 컴포넌트에 직접 프로퍼티들을 저장하지 말고, Article 클래스의 인스턴스에 저장한다.

우선 이 클래스부터 가져온다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
6 import { Article } from './article.model';
```

사용은 다음과 같이 하면 된다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
13 export class ArticleComponent implements OnInit {
14   @HostBinding('attr.class') cssClass = 'row';
15   article: Article;
16
17   constructor() {
18     this.article = new Article(
19       'Angular 2',
20       'http://angular.io',
21       10);
22   }
23
24   voteUp(): boolean {
25     this.article.votes += 1;
26     return false;
27   }
28
29   voteDown(): boolean {
30     this.article.votes -= 1;
31     return false;
32   }
33
34   ngOnInit() {
35   }
36
37 }
```

무엇을 변경했는지 자세히 들여다보자. title과 link, votes 프로퍼티를 컴포넌트에 직접 저장하지 않고, article을 가리키는 참조만 저장했다. 이렇게 하면 article의 타입을 새 Article 클래스로 정의할 수 있어 깔끔해진다. voteUp(과 voteDown)의 경우, 컴포넌트가 아니라 article에서 votes를 증가시켜야 한다.

하지만 이런 리팩토링을 구현하자면 한 가지 더 변경해야 할 것이 있다. 템플릿 변수를 올바른 위치에서 가져오기 위해 뷰를 업데이트해야 한다. 따라서 템플릿 태그를 변경하여 article에서 읽어 와야 한다. 다시 말해, 전에는 {{ votes }}이었던 곳을 {{ article.votes }}로 변경해야 한다. title과 link도 마찬가지다.

code/first-app/angular-reddit/src/app/article/article.component.html

```
<div class="four wide column center aligned votes">
  <div class="ui statistic">
    <div class="value">
      {{ article.votes }}
    </div>
    <div class="label">
      Points
    </div>
  </div>
</div>
<div class="twelve wide column">
  <a class="ui large header" href="{{ article.link }}">
    {{ article.title }}
  </a>
  <ul class="ui big horizontal list voters">
    <li class="item">
      <a href (click)="voteUp()">
        <i class="arrow up icon"></i>
          upvote
        </a>
    </li>
    <li class="item">
      <a href (click)="voteDown()">
        <i class="arrow down icon"></i>
        downvote
      </a>
    </li>
  </ul>
</div>
```

브라우저를 다시 로드하면 모든 것이 올바로 동작하는 모습을 확인할 수 있을 것이다. 지금이 한결 나아졌지만, 코드를 들여다보면 아쉬운 점이 보인다. voteUp과 voteDown 메서드가

Article 클래스의 캡슐화에 방해가 되는데, 게시물이 내부 프로퍼티를 직접 변경하기 때문이다.

> voteUp과 voteDown은 객체들 간 구조나 프로퍼티 정보의 소통을 최소화한다는 이른바 디미터의 법칙(Law of Demeter)[11]에 위배된다.

정리하자면, ArticleComponent 컴포넌트가 Article 클래스의 내부를 지나치게 많이 알고 있다. 이를 해결하려면 Article 클래스의 voteUp과 voteDown 메서드를 추가해야 한다(그리고 domain 함수도 추가해야 한다. 이 함수에 대해서는 잠시 뒤에 다룰 것이다).

code/first-app/angular-reddit/src/app/article/article.model.ts

```
export class Article {
  title: string;
  link: string;
  votes: number;

  constructor(title: string, link: string, votes?: number) {
    this.title = title;
    this.link = link;
    this.votes = votes || 0;
  }

  voteUp(): void {
    this.votes += 1;
  }

  voteDown(): void {
    this.votes -= 1;
  }

  // domain()은 URL에서 도메인을 추출하는 유틸리티 함수다.
  // 잠시 뒤에 다룰 것이다.
  domain(): string {
    try {
      // 예: http://foo.com/path/to/bar
      const domainAndPath: string = this.link.split('//')[1];
      // 예: foo.com/path/to/bar
      return domainAndPath.split('/')[0];
    } catch (err) {
      return null;
    }
  }
}
```

11 http://en.wikipedia.org/wiki/Law_of_Demeter

그리고 두 메서드를 호출하기 위해 ArticleComponent를 변경한다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
13 export class ArticleComponent implements OnInit {
14   @HostBinding('attr.class') cssClass = 'row';
15   article: Article;
16
17   constructor() {
18     this.article = new Article(
19       'Angular 2',
20       'http://angular.io',
21       10);
22   }
23
24   voteUp(): boolean {
25     this.article.voteUp();
26     return false;
27   }
28
29   voteDown(): boolean {
30     this.article.voteDown();
31     return false;
32   }
33
34   ngOnInit() {
35   }
36
37 }
```

voteUp 함수를 왜 모델과 컴포넌트, 두 곳에 두어야 할까? voteUp( )과 voteDown( )을 두 클래스에 둔 이유는 각 함수가 서로 조금씩 다르기 때문이다. ArticleComponent의 voteUp( )은 '컴포넌트 뷰'와 관련이 있는 반면, Article 모델의 voteUp( )은 모델에서 어떤 '변형(mutation)'이 일어나는지 정의한다.

다시 말해, Article 클래스는 투표가 이뤄질 때 어떤 기능이 모델에서 일어나는지 캡슐화할 수 있다. '실제' 앱에서는 웹 서버에 API 요청을 하는 등 Article 모델의 내부 구조가 훨씬 더 복잡하다. 그리고 컴포넌트 컨트롤러에서 이렇게 복잡한 모델 전용 코드를 작성하고 싶지는 않을 것이다.

마찬가지로, ArticleComponent에서도 false를 리턴한다. 이벤트를 전파하지 말라고 하는 것인데, 이는 뷰 전용 로직의 일부이며, Article 모델의 voteUp( ) 함수가 뷰 전용 API를 알지 못하도록 해야 한다는 것이다. 다시 말해, Article 모델에서는 투표와 구체적인 뷰를 나누어야 한다.

브라우저를 다시 로드해도 화면은 이전과 달라진 것이 없다. 내부적으로 코드가 더 명확해지고 단순해졌을 뿐이다.

ArticleComponent 컴포넌트의 정의를 확인해 보자. 정말 짧아졌다! 컴포넌트에서 상당한 로직을 걷어내 모델로 이전했기 때문이다. 이와 관련한 MVC 가이드라인은 Fat Models, Skinny Controllers[12]에서 찾아볼 수 있다. 요컨대 상당수 로직을 모델로 이전하여 컴포넌트의 일을 최소로 만들자는 것이다.

1.10.2 여러 Article 저장하기

여러 Article을 리스트로 나타낼 수 있도록 코드를 작성해 보자. 우선, AppComponent에서 게시물 컬렉션을 가지도록 한다.

code/first-app/angular-reddit/src/app/app.component.ts

```
import { Component } from '@angular/core';
import { Article } from './article/article.model'; // <-- 가져오기

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  articles: Article[]; // <-- 컴포넌트 프로퍼티

  constructor() {
    this.articles = [
      new Article('Angular 2', 'http://angular.io', 3),
      new Article('Fullstack', 'http://fullstack.io', 2),
      new Article('Angular Homepage', 'http://angular.io', 1),
    ];
  }
```

AppComponent에는 다음 행이 있다.

```
10 articles: Article[];
```

Article[ ];이 조금 낯설지도 모르겠다. articles는 Articles의 Array라는 뜻이다. Array <Article>로 작성해도 같으며, 이런 패턴을 가리켜 **제네릭(generics)**이라고 부른다. 제네릭은 자바나 C# 등의 언어에 소개된 개념으로서, 컬렉션(Array)에 타입이 있다는 뜻이다. 즉, Array는 Article 타입의 객체를 담는 컬렉션이라는 것이다.

12 http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model 단축URL goo.gl/Qye8

Article 클래스에 액세스하려면, 우선 가져오기부터 해야 한다. 컨스트럭터에서 this.articles를 설정하여 이 Array를 추가한다.

code/first-app/angular-reddit/src/app/app.component.ts

```
12   constructor() {
13     this.articles = [
14       new Article('Angular 2', 'http://angular.io', 3),
15       new Article('Fullstack', 'http://fullstack.io', 2),
16       new Article('Angular Homepage', 'http://angular.io', 1),
17     ];
18   }
```

1.10.3 Input으로 ArticleComponent 설정하기

Article 모델의 리스트를 만들었다. 이를 ArticleComponent 컴포넌트로 어떻게 전달할 수 있을까? 다시 한 번 Input들을 사용했다. 앞에서는 다음처럼 ArticleComponent 클래스를 정의했다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
13 export class ArticleComponent implements OnInit {
14   @HostBinding('attr.class') cssClass = 'row';
15   article: Article;
16
17   constructor() {
18     this.article = new Article(
19       'Angular 2',
20       'http://angular.io',
21       10);
22 }
```

여기에는 한 가지 문제가 있다. 특정 Article을 컨스트럭터에 직접 하드코딩했다는 것이다. 컴포넌트를 만드는 이유에는 캡슐화도 있지만 재사용도 있다.

지금은 표시하려는 Article을 설정해야 한다. 이를테면 게시물이 article1, article2, 이렇게 둘이라면, 다음과 같이 Article을 컴포넌트의 '파라미터'로 전달하여 app-article 컴포넌트를 재사용할 수 있다.

```
1 <app-article [article]="article1"></app-article>
2 <app-article [article]="article2"></app-article>
```

앵귤러에서는 다음처럼 Component의 프로퍼티에 Input 데코레이터를 사용하여 이 과정을 수행할 수 있다.

```
class ArticleComponent {
  @Input() article: Article;
// ...
```

myArticle 변수에 Article이 담기면, 이를 뷰의 ArticleComponent로 전달할 수 있다. 한 가지 기억할 것이 있다. [variableName]처럼 변수 이름을 대괄호로 묶으면 요소 형태로 변수를 전달할 수 있다. 이를테면 다음과 같다.

```
<app-article [article]="myArticle"></app-article>
```

문법 구조를 살펴보자. [article]처럼 입력의 이름을 대괄호로 묶었다. 이 입력으로 전달하려는 것이 바로 속성의 값이다. 지금 매우 중요한 내용을 언급했는데, ArticleComponent 인스턴스의 this.article이 myArticle로 설정된다. 따라서 myArticle 변수는 '파라미터'로(즉, 입력으로) 컴포넌트에 전달된다.

다음은 @Input을 사용한 ArticleComponent 컴포넌트의 모습이다.

code/first-app/angular-reddit/src/app/article/article.component.ts

```
import {
  Component,
  OnInit,
  Input,    // <-- 추가,
  HostBinding
} from '@angular/core';
import { Article } from './article.model'; // <-- 추가

@Component({
  selector: 'app-article',
  templateUrl: './article.component.html',
  styleUrls: ['./article.component.css']
})
export class ArticleComponent implements OnInit {
  @HostBinding('attr.class') cssClass = 'row';.
  @Input() article: Article;

  constructor() {
    // 게시물은 Input 행에 위치한다.
```

```
    // 따라서 따로 할 일은 없다.
  }

  voteUp(): boolean {
    this.article.voteUp();
    return false;
  }

  voteDown(): boolean {
    this.article.voteDown();
    return false;
  }

  ngOnInit() {
  }

}
```

import를 꼭 기억해야 한다! @angular/core에서 Input 클래스를 가져왔다. 그리고 AppComponent에서처럼 Article 모델도 가져왔다.

1.10.4 Articles 리스트 렌더링하기

앞에서 우리는 articles 배열을 저장하기 위해 AppComponent를 설정했다. 이제 모든 articles를 '렌더링'하기 위해 AppComponent를 설정해 보자. 그러려면 <app-article> 태그만 사용하지 말고 NgFor 지시자를 사용하여 articles 리스트를 반복하며 app-article을 렌더링해야 한다. 다음을 AppComponent @Component의 template에 추가한다. 추가할 위치는 닫는 <form> 태그 바로 아래다.

```
    Submit link
  </button>
</form>

<!-- 추가 시작 -->
<div class="ui grid posts">
  <app-article
    *ngFor="let article of articles"
    [article]="article">
  </app-article>
</div>
<!-- 추가 끝 -->
```

앞에서는 NgFor 지시자를 사용하여 이름 리스트를 불릿 리스트로 렌더링했다. 이 문법 구조는 여러 컴포넌트를 렌더링할 때도 적용할 수 있다.

*ngFor="let article of articles"에서는 articles 리스트를 반복하며 article이라는 지역 변수를 만든다(리스트의 항목을 반복한다).

article 입력을 컴포넌트에 지정하기 위해 [inputName]="inputValue" 구문을 사용했다. 여기서는 ngFor가 설정한 article 지역 변수의 값에 article 입력을 설정했다는 뜻이다.

앞의 코드 스니펫에서 article 변수를 여러 번 사용했다. NgFor가 만든 임시 변수의 이름을 foobar로 변경하면 (잠재적으로) 더 명확해진다.

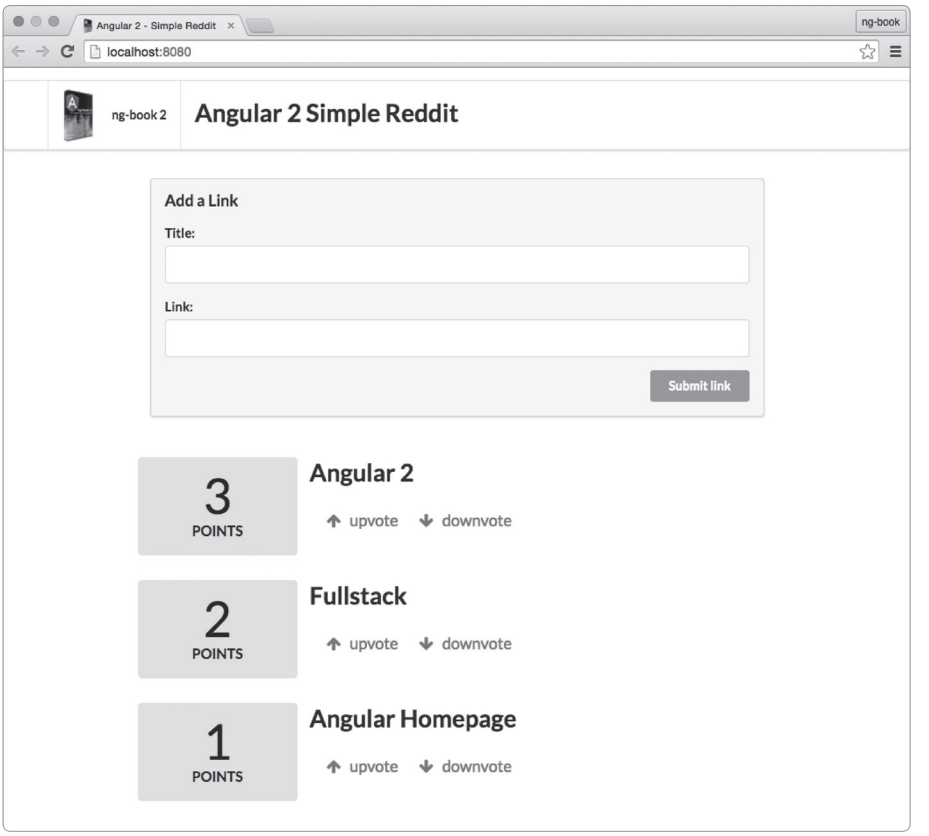
```
<app-article
  *ngFor="let foobar of  articles"
  [article]="foobar">
</app-article>
```

따라서 변수는 모두 세 개가 필요하다.

1. articles — Article의 Array. RedditApp 컴포넌트에서 정의되었다.
2. foobar — articles(Article)의 단일 요소. NgFor가 정의했다.
3. article — 필드 이름. ArticleComponent의 inputs에서 정의되었다.

기본적으로 NgFor는 임시 변수인 foobar를 생성하고, 우리는 이를 app-article로 전달했다.

브라우저를 다시 로드하면 모든 게시물이 렌더링된다.



여러 게시물이 렌더링되었다.

1.11 새 Article 추가하기

이제 버튼을 누르면 실제로 새 게시물이 추가되도록 addArticle을 변경해야 한다. addArticle 메서드를 다음처럼 변경한다.

code/first-app/angular-reddit/src/app/app.component.ts

```
20  addArticle(title: HTMLInputElement, link: HTMLInputElement): boolean {
21    console.log(`Adding article title: ${title.value} and link: ${link.value}`);
22    this.articles.push(new Article(title.value, link.value, 0));
23    title.value = '';
24    link.value = '';
25    return false;
26  }
```

이 코드에서는 다음 일들을 한다.

1. 제출된 제목과 URL로 새 Article 인스턴스를 만든다.
2. 이 인스턴스를 Article 배열에 추가한다.
3. input 필드의 값을 지운다.

> input 필드의 값을 어떻게 지울까? 자, 기억하는지 모르겠지만 title과 link는 'HTMLInputElement 객체'였다. 다시 말해, 그 프로퍼티들을 설정할 수 있다. value 프로퍼티를 변경하면 페이지의 input 태그도 변경된다.

새 게시물을 입력 필드에 추가하고 '링크 제출하기(Submit Link)'를 클릭하면, 새 게시물이 추가된 것을 확인할 수 있다.

1.12 마무리 손질

1.12.1 Article 도메인 표시하기

이제 마무리할 때가 되었다. 사용자가 링크를 클릭했을 때 어느 도메인으로 리디렉트될지 알려 주는 일종의 힌트를 링크 옆에 추가해 보자. 다음과 같이 Article 클래스에 domain 메서드를 추가한다.

code/first-app/angular-reddit/src/app/article/article.model.ts

```
22   domain(): string {
23     try {
24       // 예: http://foo.com/path/to/bar
25       const domainAndPath: string = this.link.split('//')[1];
26       // 예: foo.com/path/to/bar
27       return domainAndPath.split('/')[0];
28     } catch (err) {
29       return null;
30    }
31   }
```

이 함수를 ArticleComponent의 템플릿에 호출한다.

```
1 <div  class="twelve  wide column">
2   <a class="ui large header" href="{{ article.link }}">
3     {{ article.title }}
4   </a>
5   <!-- 이곳 -->
6   <div class="meta">({{ article.domain() }})</div>
```

```
  <ul class="ui big horizontal list voters">
    <li class="item">
      <a href (click)="voteUp()">
```

브라우저를 다시 로드하면 각 URL의 도메인 이름을 볼 수 있다.

1.12.2 점수별로 재정렬하기

막상 게시물을 클릭하고 투표해 보면 무언가 이상하다는 느낌을 받을 것이다. 게시물들이 점수에 따라 재정렬되지 않는다! 가장 높은 점수를 받은 게시물이 최상단에 보이고, 그보다 낮은 순위의 게시물이 아래로 내려가는 것은 상식에 가깝다.

articles는 AppComponent 클래스의 Array에 저장되지만, 이 Array는 정렬되지 않은 상태다. AppComponent에 sortedArticles라는 새 메서드를 만들어 이 문제를 해결해 보자.

code/first-app/angular-reddit/src/app/app.component.ts

```
  sortedArticles(): Article[] {
    return this.articles.sort((a: Article, b: Article) => b.votes - a.votes);
  }
```

ES6 화살표 함수 이 코드 스니펫에는 ES6의 '화살표' 함수(=>)가 사용되었다. 자세한 내용은 MDN web docs의 arrow functions[13]에서 찾아볼 수 있다.

내장 함수인 sort( )는 MDN web docs의 Array.prototype.sort( )[14]에서 자세한 내용을 찾아볼 수 있다.

ngFor에서는 sortedArticles( )를 반복 처리했다(articles를 직접 처리하지 않았다).

```
<div class="ui grid posts">
  <app-article
    *ngFor="let article of sortedArticles()"
    [article]="article">
  </app-article>
</div>
```

13 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions 단축URL goo.gl/m1coLv

14 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort 단축URL goo.gl/PHlDE

1.13 배포

온전하게 실행되는 앱이 완성되었다. 이제 우리의 친구들이 사용할 수 있도록 인터넷에 올릴 순서만 남았다.

> ⚠ 제품 단계 앱의 배포와 성능은 뒷부분에서 다룰 중급 주제다. 지금은 배포가 무엇인지 간단하게나마 훑고 지나가기로 하겠다.

앱의 **배포(deployment)**는 코드를 서버로 제공하는 행위다. 서버로 제공해야 다른 사람들이 코드에 액세스할 수 있다. 뭉뚱그려 말하면 배포 시 다음과 같은 일들을 해야 한다.

- 모든 타입스크립트 코드를 자바스크립트로 '컴파일'한다(브라우저가 읽을 수 있도록).
- 모든 자바스크립트 코드 파일을 한두 개의 파일로 '묶는다(bundle)'.
- 자바스크립트와 HTML, CSS, 이미지들을 서버로 '업로드'한다.

결과적으로, 앵귤러 앱은 '자바스크립트 코드를 로드'하는 HTML 파일이다. 따라서 인터넷에 존재하는 어떤 컴퓨터에 코드를 업로드해야 한다. 하지만 그보다 먼저 앵귤러 앱을 **빌드(build)**해야 한다.

1.13.1 앱을 제품으로 빌드하기

지금까지 앱을 만드느라 사용한 앵귤러 CLI 도구는 앱을 제품으로 빌드할 때도 사용할 수 있다. 사실 다음과 같은 명령 한 줄만 입력하면 된다. first_app/angular-reddit에서 다음을 입력한다.

```
ng build --target=production --base-href /
```

ng 도구는 이 명령을 받아 앱을 production 환경에 알맞도록 빌드한다. 이와 함께 --base-href는 슬래시 하나로 설정되었다. base-href는 앱이 사용할 '루트' URL이다. 만일, 앱을 서버의 /ng-book-demo/라는 서브폴더에 배포하겠다면 --base-href에 '/ng-book-demo/'를 지정해야 한다. 명령이 실행 완료되기까지 잠시 기다리면 로컬 디스크에서 dist 폴더가 생성된다.

```
$ ls dist/
136B   assets/
5.3K   favicon.ico
```

```
 27K   flags.9c74e172f87984c48ddf.png
306K   icons.2980083682e94d33a66e.svg
119K   icons.706450d7bba6374ca02f.ttf
 55K   icons.97493d3f11c0a3bd5cbd.woff2
 70K   icons.d9ee23d59d0e0e727b51.woff
 59K   icons.f7c2b4b747b1a225eb8d.eot
1.1K   index.html
1.4K   inline.44deb5fed75ee6385e18.bundle.js
 17K   main.c683e6eda100e8873d71.bundle.js
 82K   polyfills.b81504c68200c7bfeb16.bundle.js
503K   styles.7f23e351d688b00e8a5b.bundle.css
440K   vendor.cc4297c08c0803bddc87.bundle.js
```

이 파일들은 앱을 컴파일한 결과다. 각 파일이 중간에는 문자들의 긴 열이 보인다. 이를테면 다음과 같다.

```
main.c683e6eda100e8873d71.bundle.js
```

이 문자들은 콘텐트의 해시인데, 여러분과는 일치하지 않을 것이다. 파일들은 아이콘 몇 개와 index.html, main.js, polyfills.js, vendor.js, styles.css 몇 가지 등이다. 이 모두를 서버로 업로드해야 한다.

1.13.2 서버로 업로드하기

HTML과 자바스크립트의 호스팅 방법은 셀 수 없이 많다. 여기서는 현재까지 가장 쉽다고 할 수 있는 방법인 now[15]를 적용하고자 한다.

now가 마음에 들지 않으면 얼마든지 다른 방법을 사용해도 무방하다. 몇 가지 대안을 제시하면, Heroku, AWS S3, FTP를 통한 자체 서버 업로드 등을 생각해 볼 수 있다. 중요한 것은 서버는 로컬 디스크의 dist 폴더에 있는 모든 것을 인터넷에 노출한다는 점이다.

■ now 설치하기

npm으로 now를 설치할 수 있다.

```
npm install -g now
```

15 https://zeit.co/now

다음처럼 now로 사이트를 대단히 쉽게 배포할 수 있다.

```
cd dist # change into the dist folder
now
```

now 명령이 실행되면 두어 가지 질문(이메일 주소 등)에 답을 해야 한다. 이메일 주소를 확인하고 그 안에 포함된 링크를 클릭한다. 계정이 확인되면 now는 코드들을 업로드하고 앱을 확인할 수 있는 URL을 제공한다. 이 URL에 접속하면 앱을 확인할 수 있다. 앱이 올바로 동작하면 URL을 친구와 공유해 보자.

축하한다! 드디어 첫 번째 앵귤러 앱의 빌드와 배포를 마무리했다.

1.14 전체 코드

이 장에서는 필요한 만큼 코드를 여럿으로 나누어 자세하게 파헤쳤다. 우리의 앱에 사용된 파일들 전부와 온전한 타입스크립트 코드는 이 책의 코드 다운로드에서 확인할 수 있다.

1.15 정리

우리가 해냈다! 첫 번째 앵귤러 앱이 완성된 것이다! 돌이켜보면 그다지 어렵지는 않았을 것이다. 물론 아직도 많은 것을 배워야 한다. 이를테면, 데이터 흐름 이해하기, AJAX 요청하기, 내장 지시자, 라우팅, DOM 조작하기 등이 여러분을 기다리고 있다.

하지만 지금은 첫 앵귤러 앱을 완성한 즐거움을 누리자. 앵귤러 앱을 어떻게 작성하는지 충분히 경험했으니 말이다. 이를 구체적으로 정리하면 다음과 같다.

1. 앱을 컴포넌트로 나누기
2. 뷰 만들기
3. 모델 정의하기
4. 모델 표시하기
5. 인터랙션 추가하기

앞으로 이어질 내용에서는 앵귤러로 정교한 앱을 만들기 위한 모든 과정을 다룰 것이다.