



Finatra: The Fast, Testable Scala Services Framework That Powers Twitter

Steve Cosenza

Finatra & Data API Tech Lead | @scosenza



+



=





FINAGLE

“Finagle is an extensible remote procedure call system for the JVM, used to construct high-concurrency servers.”





FINAGLE

box

STRAVA™



VICE

rdio™



foursquare™

PAGERDUTY

nest

NCBI

Sabre



EA

The New York Times





SINATRA

“Sinatra is a DSL for quickly creating web applications in Ruby with minimal effort.”

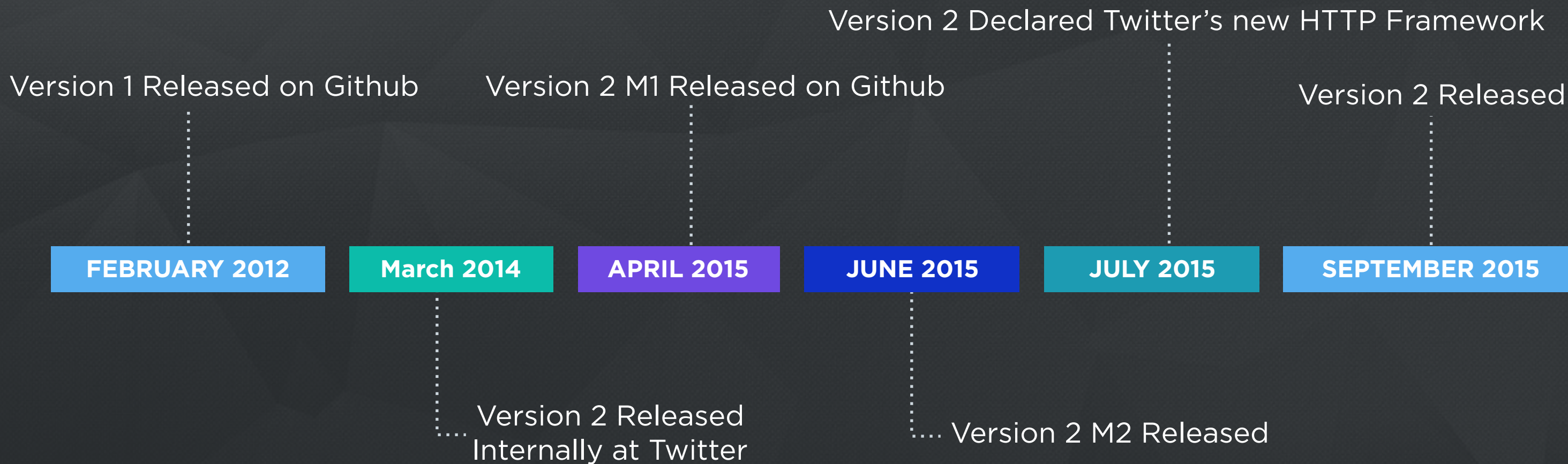




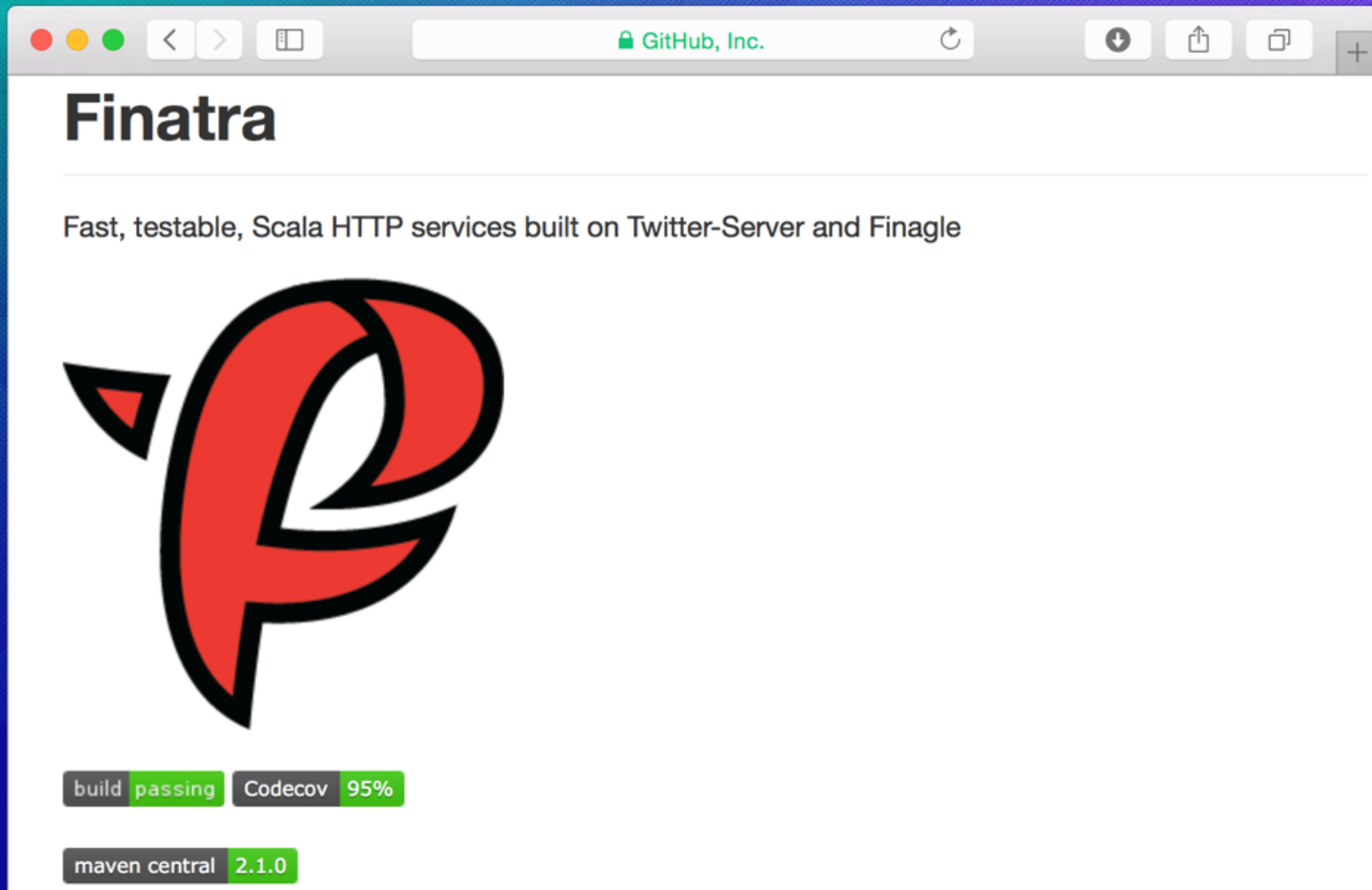
A lightweight framework for building
feature testable applications on
Twitter's open source stack.



Finatra Project History



Open Source - Apache 2 License



Feature Testing

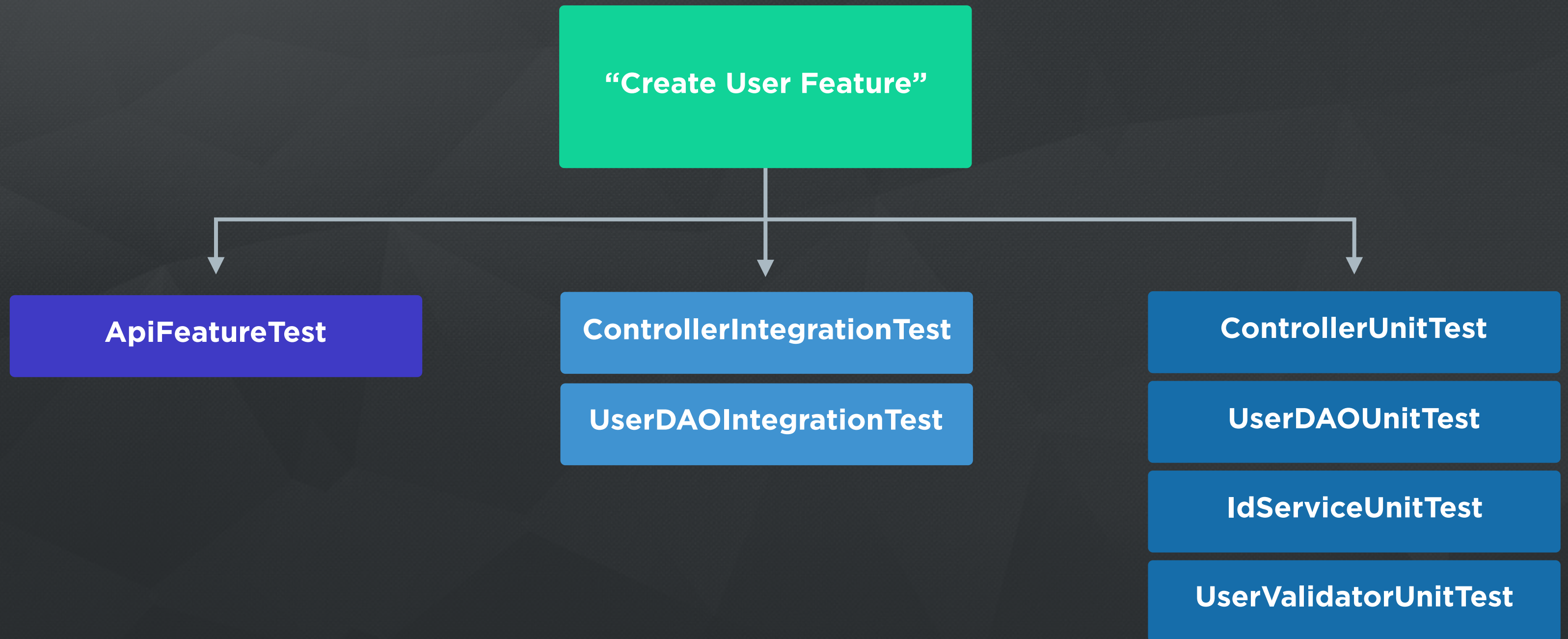


+ FEATURE TEST

A hybrid of an integration and component test enabling test-doubles and black-box/white-box assertions.



Feature Tests



“In our test philosophy... a feature test is something we must have... It’s a test that verifies that something we want to build works, and works as expected... If you don’t have a feature test, you might not even have a feature.”

–Viktor Klang
Akka Tech Lead | Typesafe



Feature Test

```
class PopularApiFeatureTest extends FeatureTest {  
  val server =  
    new EmbeddedHttpServer(new PopularApiServer)  
  
  "Handle POST request" in {  
    server.httpPost(  
      path = "/api/popular",  
      postBody = """"{ "start": "2015-09-01", ... }""",  
      andExpect = Status.Ok,  
      withJsonBody = """"{  
        "start" : "2015-09-01T00:00:00Z",  
        "end" : "2015-09-01T00:00:00Z",  
        "tweets" : [ ... ]  
      }""")  
  }  
}
```

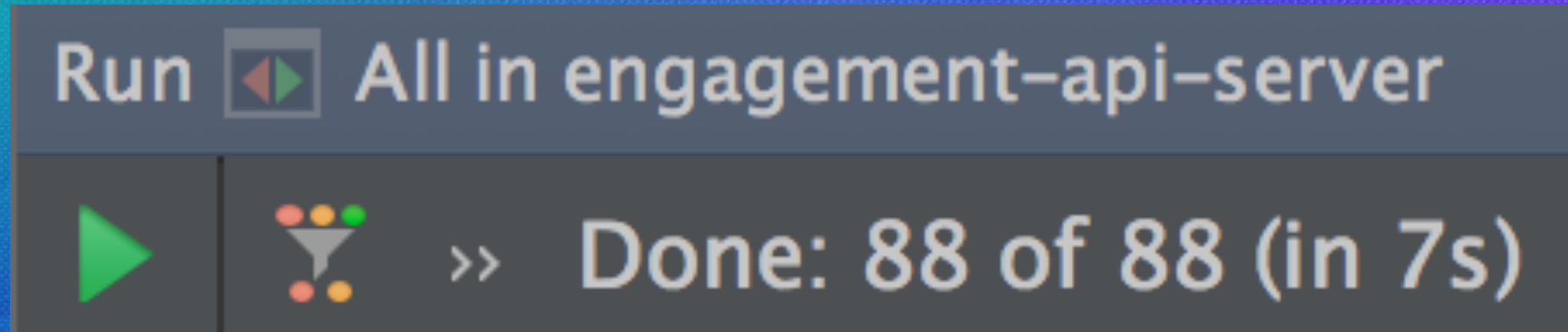


}

Qualities Of A Good Test



Real World Feature Tests



SCoverage generated at Thu Sep 24 20:40:05 PDT 2015

Lines of code:	1702	Files:	36
Statement coverage:	99.61 %		



Dependency Injection Frameworks





BOILERPLATE

The sections of code that have to be included in many places with little or no alteration



Dependency Injection Frameworks

**Eliminate
Boilerplate**

**Consistent
Configuration**

**Facilitate
Testing**



“The main advantage that [Dependency Injection] brings is not realised until you start writing tests... once you move onto integration testing, this is where Play can start to realise the advantages of DI.”

–James Roper (March 2015)
Play Framework Tech Lead | Typesafe



JSR-330



GUICE

“Lightweight dependency injection framework for Java... brought to you by Google.”



JSR-330 Dependency Injection

@Singleton

```
class PopularApiController @Inject()(
    userRepository: UserRepository,
    searchService: SearchService,
    engagementService: EngagementService)
    extends Controller {

    post("/api/popular") { request: PopularPostRequest =>
        ...
    }
}
```



Server Construction

```
class PopularApiServer extends HttpServer {  
  override val modules = Seq(  
    UserRepositoryModule,  
    SearchServiceModule,  
    EngagementServiceModule)  
  
  override def configureHttp(router: HttpRouter) {  
    router  
      .filter[LoggingMDCFilter[Request, Response]]  
      .filter[TraceIdMDCFilter[Request, Response]]  
      .add[PopularApiController]  
  }  
}
```



Feature Testing

```
class PopularApiFeatureTest extends FeatureTest with Mockito {  
    val server = new EmbeddedHttpServer(new PopularApiServer)  
  
    // Bind Mocks  
    @Bind val databaseClient = mock[DatabaseClient]  
    @Bind @SearchClient val searchClient = mock[HttpClient]  
    @Bind @EngagementsClient val engagementsClient = mock[HttpClient]  
  
    "Handle POST request" in {  
        // Setup Mocks  
        databaseClient.executeSql(...) returns Future(SqlResponse(...))  
        searchClient.execute(...) returns Future(SearchResponse(...))  
        engagementsClient.execute(...) returns Future(EngagementResponse(...))  
  
        server.httpPost(...)  
    }  
}
```



Feature Testing

```
class  
  val
```

```
// Bind Mocks
```

```
@Bind val databaseClient = mock[DatabaseClient]
```

```
@Bind @SearchClient val searchClient = mock[HttpClient]
```

```
@Bind @EngagementsClient val engagementsClient = mock[HttpClient]
```

```
"Handle POST request"
```

```
  // Setup Mocks
```

```
}
```

```
}
```



Feature Testing

```
class
  val

  // Bind Mocks
  @Bind
  @Bind @SearchClient
  @Bind @EngagementsClient

  "Handle POST request"
    // Setup Mocks
    databaseClient.executeSql(...) returns Future(SqlResponse(...))
    searchClient.execute(...) returns Future(SearchResponse(...))
    engagementsClient.execute(...) returns Future(EngagementResponse(...))

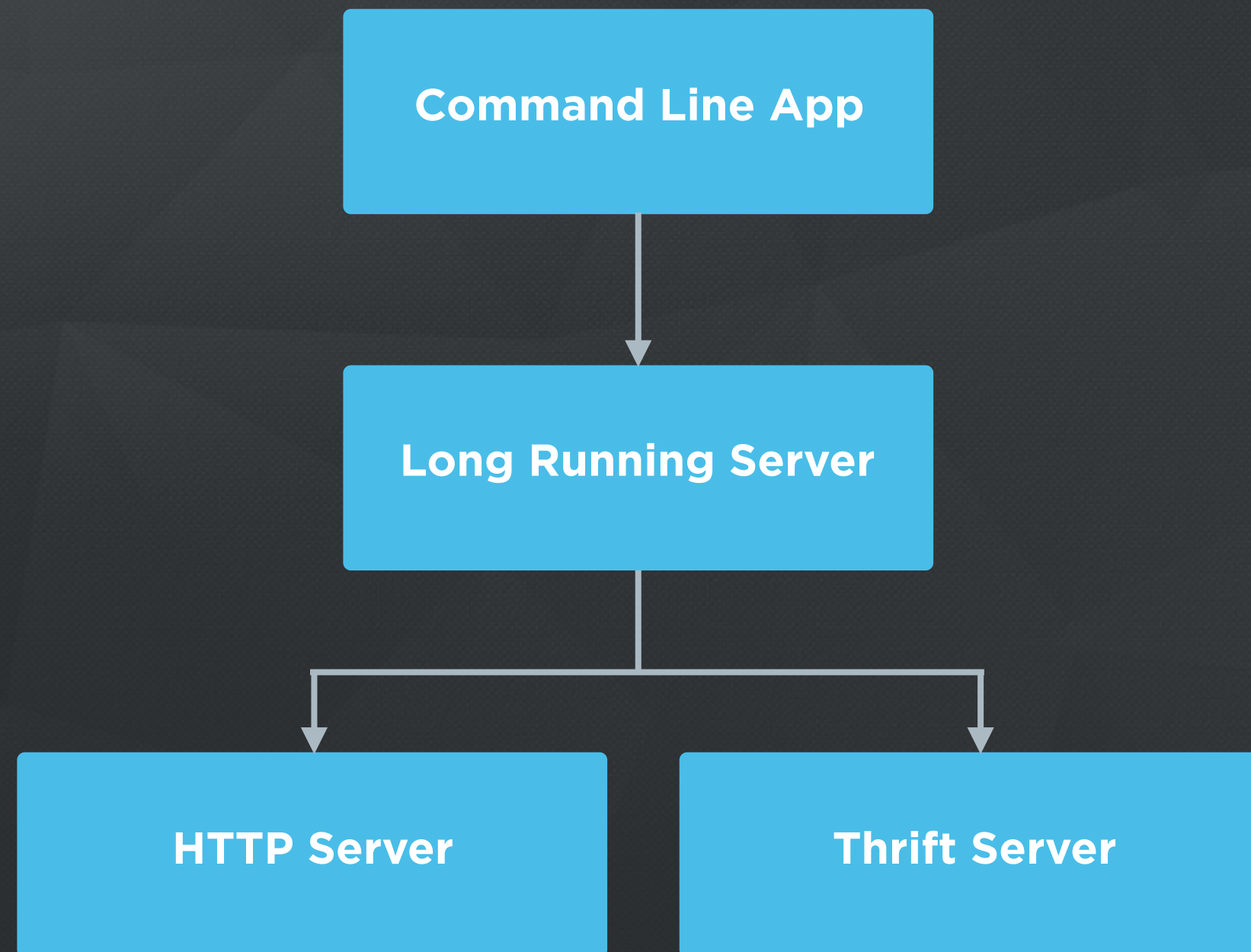
  }
}
```



Consistency



Consistency



Consistency: Thrift Server

```
class PopularApiThriftServer extends ThriftServer {  
  override val modules = Seq(  
    UserRepositoryModule,  
    SearchServiceModule,  
    EngagementServiceModule)  
  
  override def configureThrift(router: ThriftRouter) {  
    router  
      .filter[LoggingMDCFilter][ThriftRequest, ThriftResponse]  
      .filter[TraceIdMDCFilter][ThriftRequest, ThriftResponse]  
      .add[PopularApiThriftService]  
  }  
}
```



Consistency: Http And Thrift Server

```
class CombinedServer extends HttpServer with ThriftServer {  
  override val modules = Seq(...)  
  
  override def configureHttp(router: HttpRouter) {  
    router  
      .filter[LoggingMDCFilter[Request, Response]]  
      .filter[TraceIdMDCFilter[Request, Response]]  
      .add[PopularApiController]  
  }  
  
  override def configureThrift(router: ThriftRouter) {  
    router  
      .filter[LoggingMDCFilter[ThriftRequest, ThriftResponse]]  
      .filter[TraceIdMDCFilter[ThriftRequest, ThriftResponse]]  
      .add[PopularApiThriftService]  
  }  
}
```



Consistency: Thrift Feature Testing

```
class PopularApiThriftFeatureTest extends FeatureTest {  
  val server =  
    new EmbeddedThriftServer(new PopularApiThriftServer)  
  
  val client =  
    server.thriftClient[PopularApiService.FutureIface]  
  
  "Handle popularTweets" in {  
    client.popularTweets(...) should equal(...)  
  }  
}
```



Consistency: Java Http Server

```
public class PopularApiServer extends HttpServer {

    @Override
    public Collection<Module> javaModules() {
        return ImmutableList.<Module>of(
            new UserRepositoryModule(),
            new SearchServiceModule(),
            new EngagementServiceModule());
    }

    @Override
    public void configureHttp(HttpRouter httpRouter) {
        httpRouter
            .filter(CommonFilters.class)
            .add(PopularApiController.class);
    }
}
```



Consistency: Http Controllers

@Singleton

```
class ScalaUserController @Inject()(
  userService: UserService)
  extends Controller {

  get("/users/:id") { request: Request =>
    userService.lookup(request.getLongParam("id"))
  }
}
```

@Singleton

```
public class JavaUserController extends JavaController {
  @Inject UserService userService;

  @Override protected void configureRoutes() {
    get("/users/:id", request ->
      userService.lookup(request.getLongParam("id")));
  }
}
```



An End to End Example

CREATING A POPULAR TWEETS API



Popular API Requirements

POST Request

Path: /api/popular

Header: api_key

JSON Body:

```
{  
  "start": "2015-09-01",  
  "end": "2015-09-30"  
}
```

HTTP 200 Response

JSON Body:

```
{  
  "start": "2015-09-01T00:00:00Z",  
  "end": "2015-09-30T00:00:00Z",  
  "tweets": [  
    {  
      "id": "123",  
      "message": "hi",  
      "counts": {  
        "impressions": "20",  
        "engagements": "10"  
      }  
    }  
  ]  
}
```



Feature Test

```
class PopularApiFeatureTest extends FeatureTest {  
  val server = new EmbeddedHttpServer(new HttpServer {})  
  "Handle POST request" in {  
    server.httpPost(  
      path = "/api/popular",  
      headers = Map("api_key" -> "secret123"),  
      postBody = """"{ "start": "2015-09-01", ... }""",  
      andExpect = Status.Ok,  
      withJsonBody = """"  
        {  
          "start" : "2015-09-01T00:00:00Z",  
          "end" : "2015-09-01T00:00:00Z",  
          "tweets" : [ ... ]  
        }  
        """" )  
  }  
}
```



Test Failure

```
=====
HTTP POST /api/popular
[Header]Content-Length -> 204
[Header]Content-Type -> application/json;charset=utf-8
[Header]Host -> 127.0.0.1:60102
[Header]api_key -> secret123
{
  "start": "2015-09-01",
  "end": "2015-09-30"
}
```

```
=====
RoutingService Request("POST /api/popular", from /127.0.0.1:60103) not
found in registered routes:
-----
```

[Status]404 Not Found

[Header]Content-Length -> 0

EmptyBody



Request Parsing

```
{  
  "start": "2015-09-01",  
  "end": "2015-09-30"  
}
```

```
case class PopularPostRequest(  
  @PastTime start: DateTime,  
  @PastTime end: DateTime,  
  @Header @NotEmpty api_key: String)
```



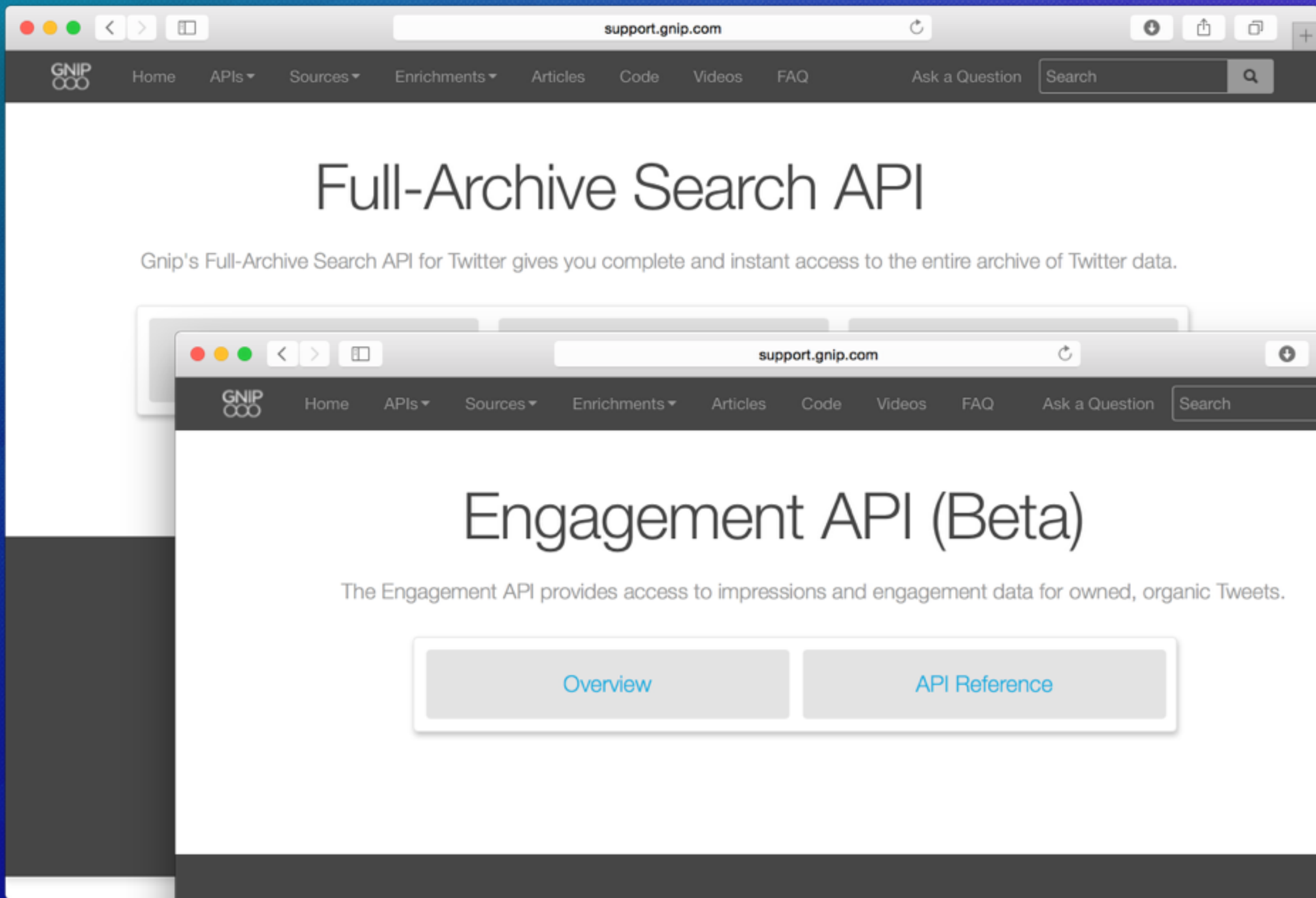
Response Generation

```
{
  "start": "2015-09-01T00:00:00Z",
  "end": "2015-09-30T00:00:00Z",
  "tweets": [
    {
      "id": "123",
      "message": "hi",
      "counts": {
        "impressions": "20",
        "engagements": "10"
      }
    }
  ]
}
```

```
case class ApiResponse(
  start: DateTime,
  end: DateTime,
  tweets: Seq[TweetResponse])
```

```
case class TweetResponse(
  id: TweetId,
  message: String,
  counts: Map[EngagementType, Long])
```





Controller

@Singleton

```
class PopularController @Inject()(
  userRepository: UserRepository,
  searchService: SearchService,
  engagementService: EngagementService)
  extends Controller {

  post("/api/popular") { request: PopularPostRequest =>
    for {
      user <- userRepository.getByApiKey(request.apiKey)
      tweets <- searchService.findTweets(user, request)
      engagements <- engagementService.lookup(user, tweets)
    } yield {
      ApiResponse.create(request, tweets, engagements)
    }
  }
}
```



Server

```
class PopularApiServer extends HttpServer {  
  
  override def modules = Seq(  
    UserRepositoryModule,  
    SearchServiceModule,  
    EngagementServiceModule)  
  
  override def configureHttp(router: HttpRouter) {  
    router  
      .filter[CommonFilters]  
      .add[PopularController]  
  }  
}
```



Feature Test

```
class PopularApiFeatureTest extends FeatureTest {  
  val server = new EmbeddedHttpServer(new PopularApiServer)  
  "Handle POST request" in {  
    ...  
  }  
}
```



But When We Run Our Tests...

```
=====
HTTP POST /api/popular
```

```
...
```

```
=====
GET /search/fullarchive/accounts/popularapi/prod.json
Authorization -> Basic Og==
```

```
internal server error
```

```
com.twitter.server.NamedResolverNotFoundException:
```

```
Resolver not found for scheme 'flag' with name 'data-api'.
```



Feature Test Mocking

```
class PopularApiFeatureTest extends FeatureTest with Mockito {  
  val server = new EmbeddedHttpServer(new PopularApiServer)  
  
  // Bind Mocks  
  @Bind val databaseClient = mock[DatabaseClient]  
  @Bind @SearchClient val searchClient = mock[HttpClient]  
  @Bind @EngagementsClient val engagementsClient = mock[HttpClient]  
  
  "Post request" in {  
    // Setup Mocks  
    databaseClient.executeSql(..) returns Future(SqlResponse(...))  
    searchClient.execute(..) returns Future(SearchResponse(...))  
    engagementsClient.execute(..) returns Future(EngageResponse(..))  
  
    server.httpPost(...)  
  }  
}
```



Test Success!

```
=====
HTTP POST /api/popular
```

```
...
{
  "start" : "2015-08-01",
  "end" : "2015-08-30"
}
```

```
=====
127.0.0.1 - - [15/Sep/2015:22:14:56 +0000] "POST /api/popular HTTP/1.1" 200 149 931 "-"
-----
```

```
[Status] Status(200)
```

```
...
{
  "start" : "2015-08-01T00:00:00Z",
  "end" : "2015-08-30T00:00:00Z",
  "tweets" : [
    {
      "id" : "123",
      "message" : "Hi",
      "counts" : {
        "impressions" : "20",
        "engagements" : "10"
      }
    }
  ]
}
```



Let's Test A Bad Request

```
"Post bad request" in {  
    server.httpPost(  
        path = "/api/popular",  
        postBody = ""  
        {  
            "start": "bad",  
            "end": "2020-08-30"  
        }  
        ""  
    ),  
    andExpect = Status.BadRequest)  
}
```



Test Success

```
=====
HTTP POST /api/popular
-----
```

```
[Status]    Status(400)
{
  "errors" : [
    "api_key: header is required",
    "start: error parsing 'bad' into an ISO 8601 datetime",
    "end: [2020-08-30T00:00:00.000Z] is not in the past"
  ]
}
```





Twitter's Production HTTP Framework
Powerful Feature Testing
Optional JSR-330 Dependency Injection
Designed for Consistency and Modularity





Questions?

Steve Cosenza @scosenza
Christopher Coco @cacoco
Finatra @finatra

