Enterprise Java Developer's

Petyo Dimitrov

October, 2017

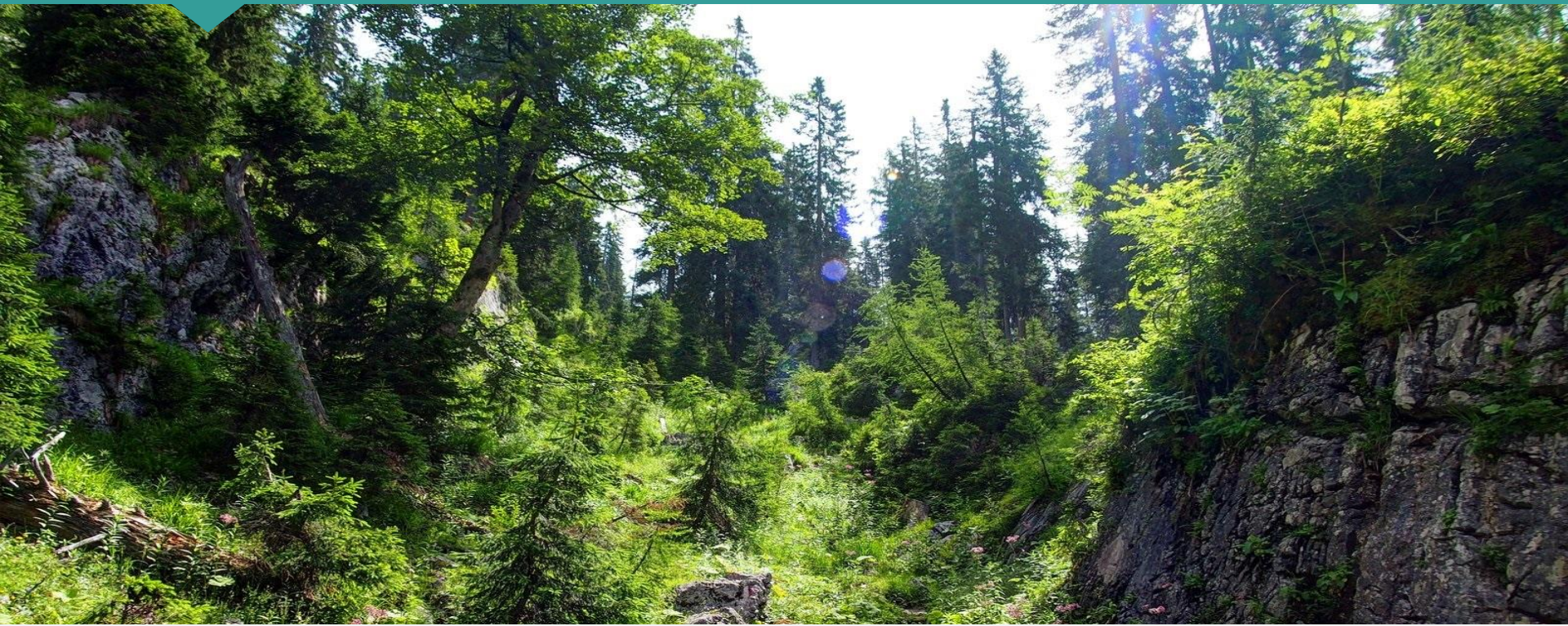
AGENDA

About me

Steps for survival

Q&A

THE ENTERPRISE JAVA WILDERNESS



3 Survival guide

STEP 1: COME PREPARED

KNOWLEDGE (1)

Solid understanding of core Java & some specifics:

- garbage collection strategies
- class loading specifics
- debugging (thread & heap dumps)

Some experience with databases and middleware

KNOWLEDGE (2)

Knowledge in OOP concepts and design patterns

- Singleton, Dependency Injection, Factory, MVC …

Core Java EE specs like Servlets, JPA & Components

Basic Linux command line skills

KNOWLEDGE IMPROVEMENT

Write code

Collaborate with experienced people and learn from them

Join an open-source project

Code reviews are a great way to learn

STEP 2: BRING GEAR

What tools should I be experienced in?

IDES & TOOLS


VisualVM
Top Consumers
Biggest Objects (Overview)
(e) 67.4 KB
(f) 58 KB
(g) 50.2 KB
(h) 42 KB
(i) 38.6 KB
(j) 35.7 KB
(k) 34.3 KB
(l) 26.5 KB
(m) 1.3 MB
(d) 123.6 KB
(c) 162.7 KB
(b) 163.9 KB
(a) 166 KB
IJ

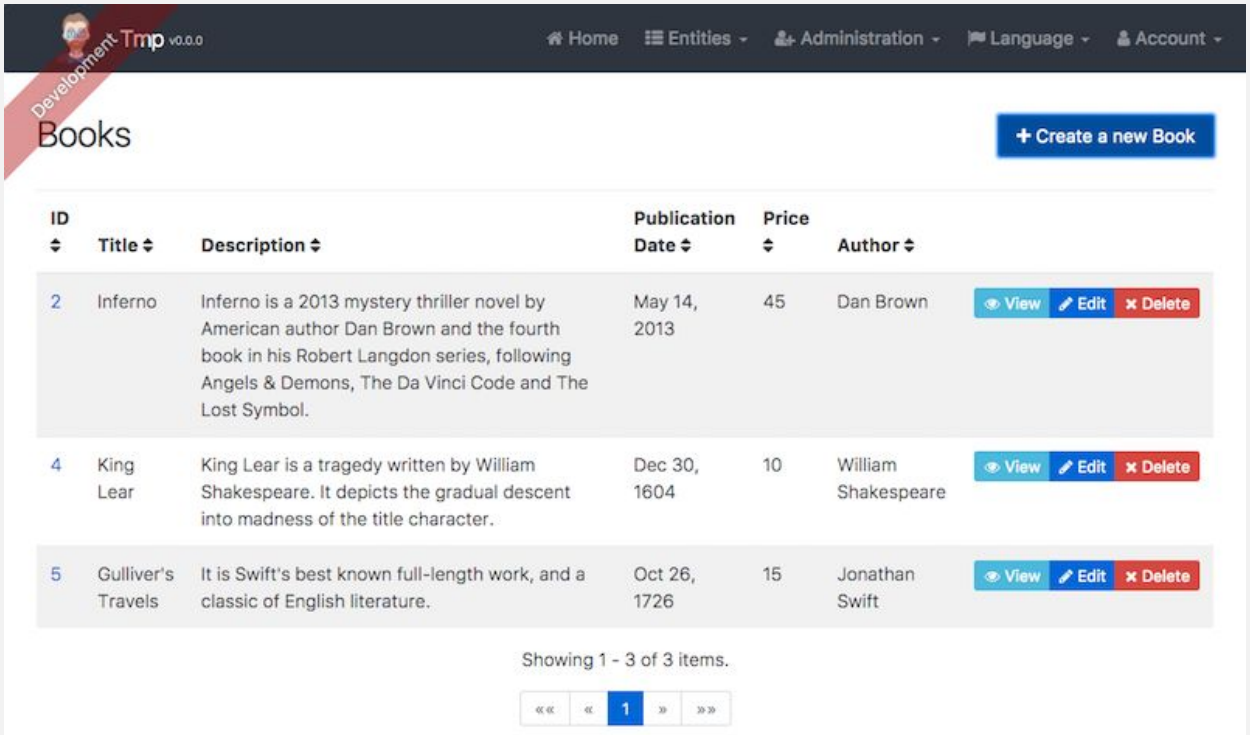
BE LAZY & AUTOMATE

Builds & Tests (via Maven, Jenkins, etc.)

Administrative tasks (via Scripts, custom tools)

Environment setup (Vagrant, Docker)

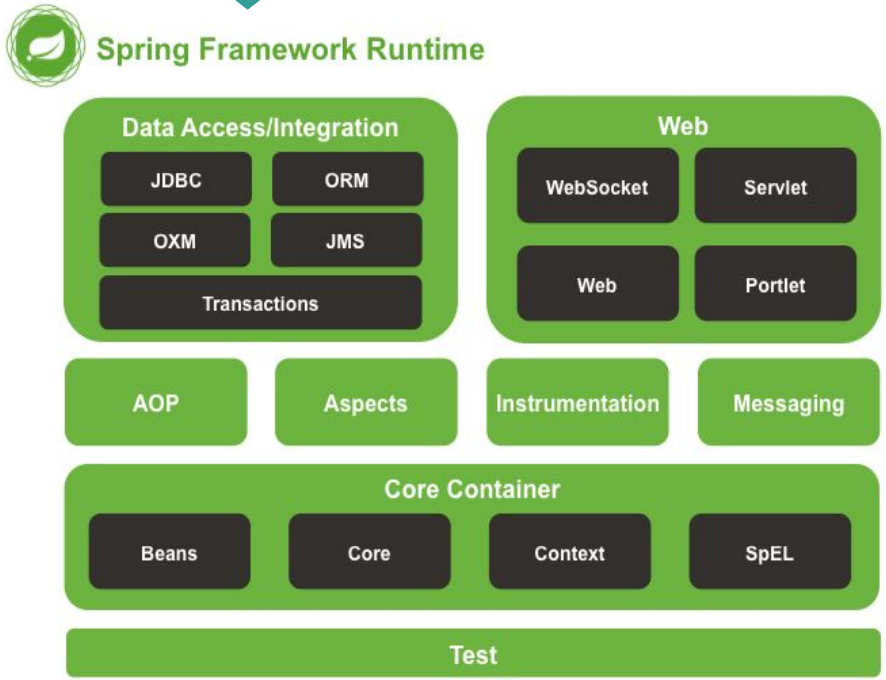
APPLICATION GENERATION

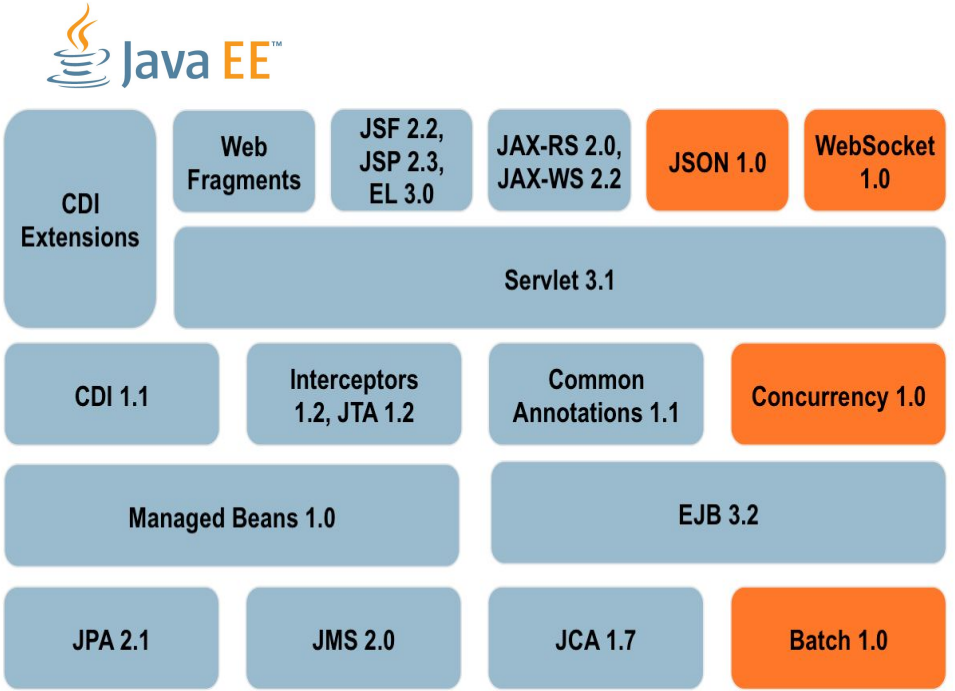


STEP 3: GET ORIENTED

Which technology stack should I choose?

SPRING VS JAVA EE

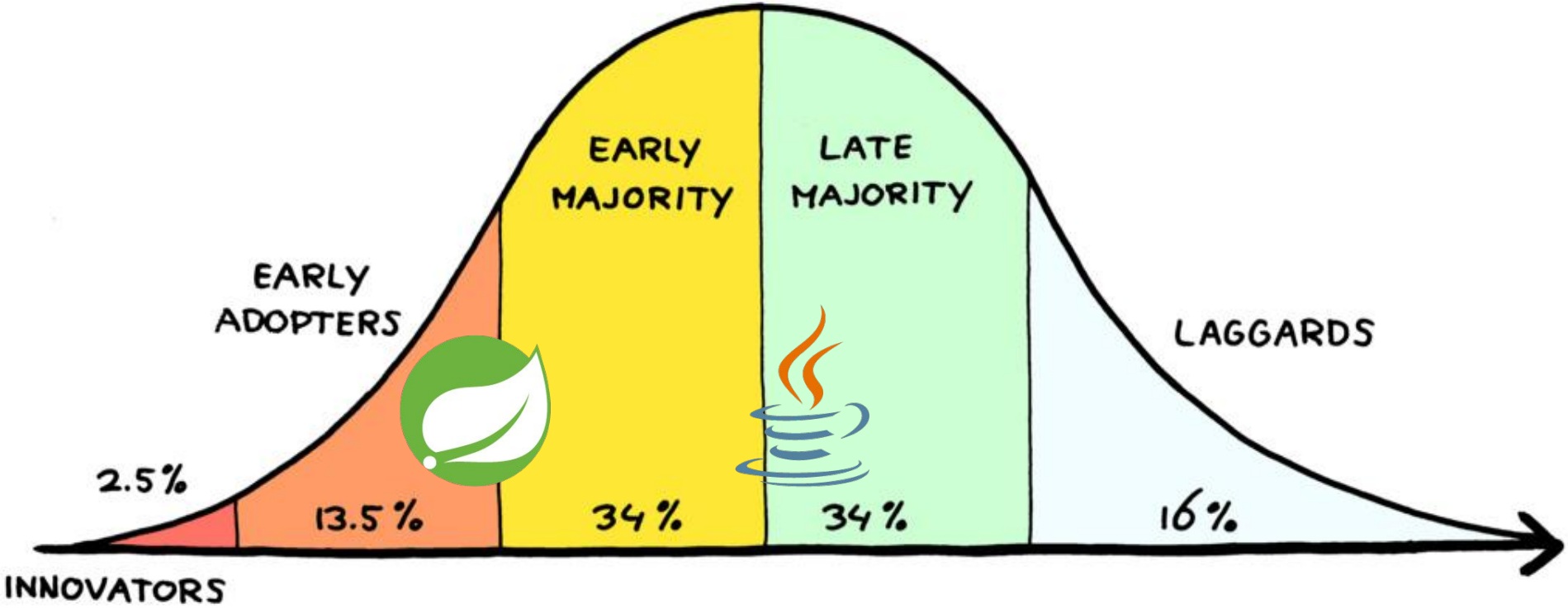




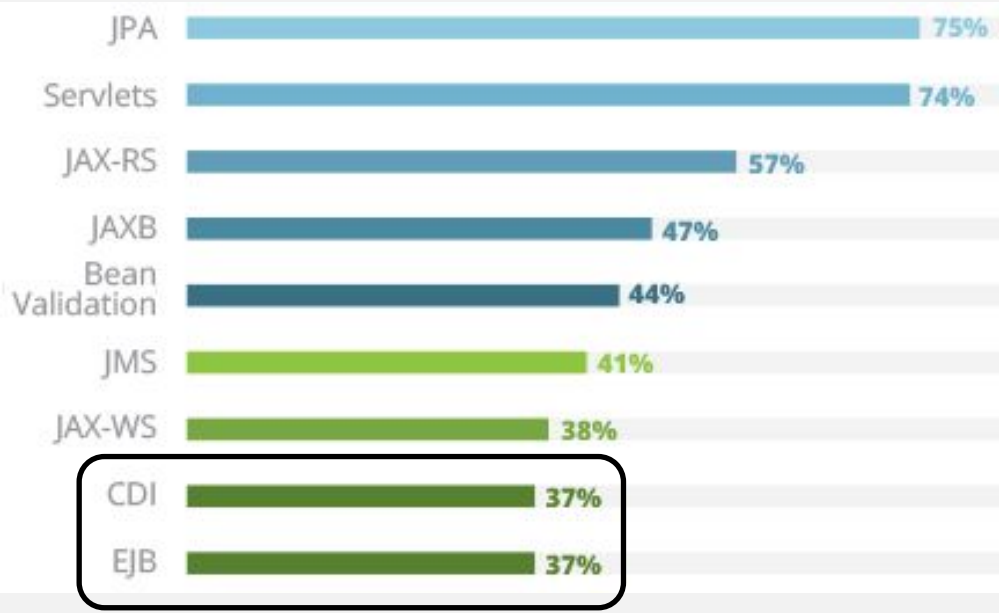
CLIENT REQUIREMENTS

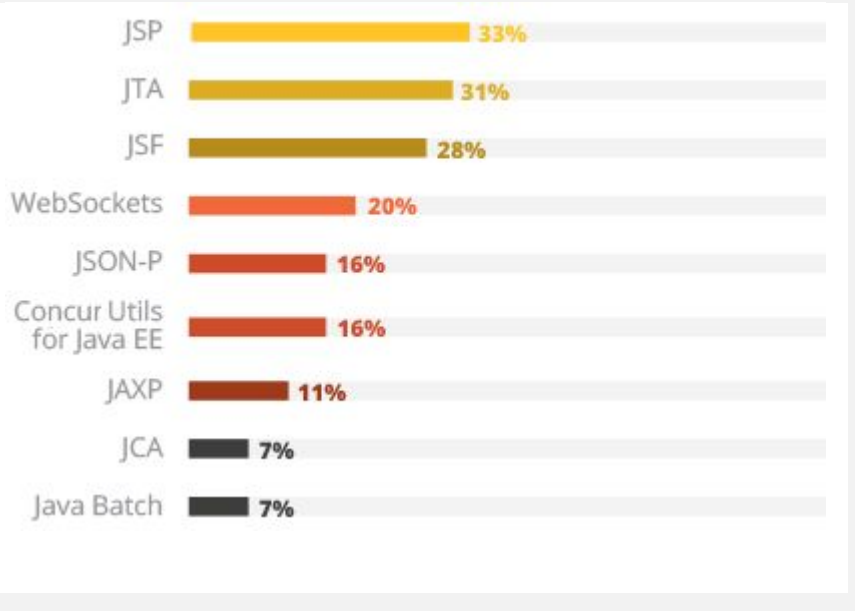
KNOWLEDGE REQUIREMENTS

PROJECT REQUIREMENTS



POPULAR JAVA EE SPECIFICATIONS





ZeroTurnaround's survey of ~1700 developers

AND NOW WHAT?


YOU CAN
SURVIVE
3 HOURS
without
SHELTER
3 DAYS
without
WATER
30 DAYS
without
FOOD

STEP 4: BUILD SHELTER

How do I setup the project?

BASIC SETUP (1)

CI:

Jenkins

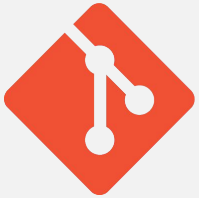
Travis CI

Build:

maven

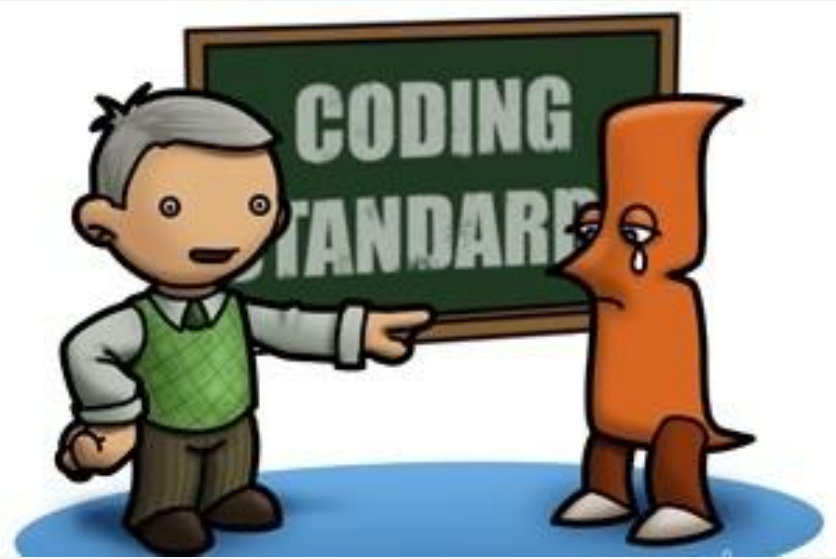
gradle

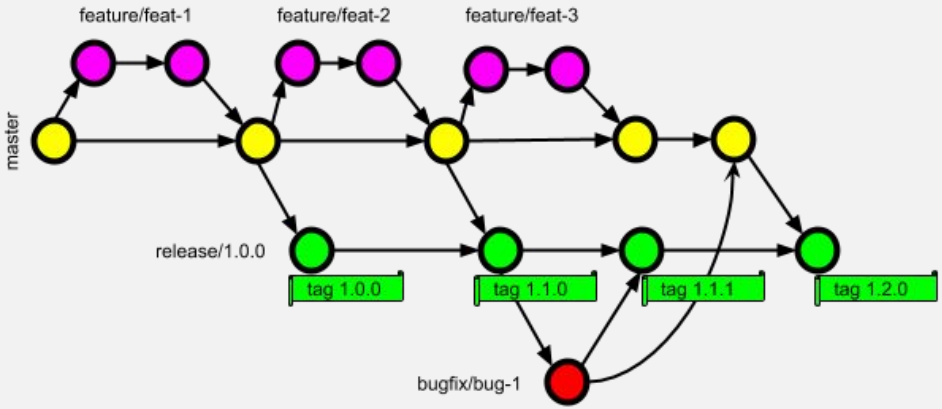
VCS:



git

BASIC SETUP (2)





ADVANCED SETUP

1. Static code analysis Sonar / IDE-based
2. DB schema management Flyway / Liquibase
3. In-memory DB for development
4. Easy to setup local environment
5. Stable staging environment
6. Continuous Delivery

UNIT TESTING!

Via:

- JUnit & Mockito / Powermock / EasyMock
- Groovy & Spock

Caveats:

- one-off short-term projects
- tests treated as second class code

UNIT TESTING ISSUES – USELESS TESTS

```
def "invite calls the service"() {
    setup:
    def form = Mock(SomeInputForm)
    def actor = Mock(Actor)

    when:
    underTest.invite(actor, form)

    then:
    1 * service.invite(actor, form)
}
```

UNIT TESTING ISSUES – BRITTLE TESTS

```
def "smart test name"() {
    setup:
    def customer = PETYO
    def device = SMART_DEVICE
    …
    when:
    def result = underTest.execute(taskData)

    then:
    1 * deviceService.findByCustomer(customer) >> serviceResultA
    0 * anotherService._

    result == expected result
}
```

STEP 5: FIND WATER

How do I implement the project?

APPLICATION DESIGN

Consider modules & package structure

Review component interfaces

Beware of excessive Dependency Injection

Principles of Domain Driven Design

SHOULD I USE AN ORM?

relational	nosql
new	legacy
object centric	data centric
CRUD queries	reporting queries

WHAT PROBLEMS CAN I EXPECT?

"Magic" powers i.e. hidden learning curve

Reduced control over DB

Loss of DB specific capabilities

Difficulty fetching necessary data

HOW TO DESIGN REST API-S?

- Follow the REST principles
  & look at the APIs of large companies
- Use proper HTTP verbs (GET, PUT, POST, PATCH…)
  - `GET /movie/1/booking`
- Use proper HTTP status codes
  - `418 I'm a teapot`

HOW TO DESIGN REST API-S? (2)

- Medium grained resources
  - up to two levels of nesting

- Security:
  - HTTPS
  - OAuth2
  - BasicAuth

HOW TO DESIGN REST API-S? (3)

- Proper URLs using plural nouns
  - `GET /movies vs GET /getAllMovies`
- Spinal-case in URLs and camelCase / snake_case for parameters
  - http://www.penisland.net/
  - `GET /order-item/1?orderNumber=2`

HOW TO DESIGN REST API-S? (4)

- Consider versioning early on:
  - only major version
  - aim to have no more than 2 versions in parallel
  - `/v1/movies`, `/v2/movies`

- Filters & sorting via URL parameters
  - `?sort=rating,budget&director=nolan`

HOW TO DESIGN REST API-S? (5)

- I18n of data:
  - via `Accept-Language: bg_BG`
- Handling of operations (i.e. non-resources)
  - `POST /email/12/send`
  - consider JSON-RPC

STEP 6: FIND FOOD

What about performance?

WHAT PROBLEMS SHOULD I EXPECT?

- Infrastructure issues (available resources, unreliability, latency)
- External system communication (synchronous calls, no timeouts, faulty integrations)
- Lack of middleware tuning (thread & connection pools, clusters)
- Garbage collection (limits, strategies)
- Bugs (synchronization issues, memory leaks)

HOW TO IMPROVE PERSISTENCE?

1. Monitor query performance
2. Review native SQL of sensitive queries
   - mark/optimize slow queries
3. Use caching offered by ORM
4. Beware of many-to-many relations & fetch types
5. Run updates/deletes in bulk (beware of cascading)
6. Paging & query projection
7. Move logic to DB

HOW TO IMPROVE FRONT END?

1. Track time for processing each REST request
2. Use gzip
3. Partial request & responses (?fields + HTTP PATCH)
4. Cache friendly results (etag, last-modified)
5. Paging

STEP 7: STAY IN ONE PLACE vs SCOUT THE AREA

QUESTIONS?

THANK YOU

petyo.dimitrov@musala.com