



# **RESTful Web Services με Jakarta EE & JAX-RS**

**Αθ. Ανδρούτσος**



# Distributed Systems (1)

Java EE – REST API

- Το πρόβλημα της επικοινωνίας ετερογενών συστημάτων μέσω δικτύου (κατανεμημένων συστημάτων) υπάρχει από παλιά και τίθεται ως εξής: Πως μπορούμε να μεταφέρουμε δεδομένα και να καλέσουμε μεθόδους μέσω Internet;
- Πως μπορεί δηλαδή ένας client να καλεί μεθόδους ενός απομακρυσμένου Server περνώντας και δεδομένα;



# Distributed Systems (2)

- Ιστορικά αυτό γινόταν με πρωτόκολλα επικοινωνίας, όπως το RPC (Remote Procedure Calls) όπου στέλνόταν από τον client το όνομα της μεθόδου και οι τυπικές παράμετροι και επιστρεφόταν τα αποτελέσματα over HTTP
- Για περισσότερο flexibility και interoperability μεταξύ ετερογενών συστημάτων υπήρχε και το middleware της CORBA και αργότερα το περιβάλλον .NET της Microsoft



# CGI Scripts

- Στη συνέχεια και με την έλευση του Web, η επικοινωνία του Client με τον Server γινόταν μέσω HTTP και του Web Server, ενώ ο Web Server μέσω ενός ειδικού script, CGI (Common Gateway Interface) που μπορεί να είναι γραμμένο σε Perl, Python, κλπ. επικοινωνούσε με τις τοπικές μεθόδους και επέστρεφε τα αποτελέσματα



# Servlets

- Αργότερα πάλι και αυτό το μοντέλο απλοποιήθηκε με τους Web Servers που είναι και Runtime Containers όπως ο Apache Tomcat οι οποίοι με τη χρήση config files (όπως web.xml) ή και annotations της γλώσσας μπορούν να αντιστοιχίσουν URIs σε μεθόδους εξυπηρέτησης των requests (π.χ. Servlets)



# Web Services

- Τα Servlets και γενικά οι Controllers εκτός από HTML αρχεία μπορούν να σερβίρουν και data, οπότε μπορούν να παίξουν το ρόλο των Web Services
- Ένα Web Service είναι ένα service που παρέχεται over HTTP το οποίο αποστέλλει πίσω στον client δεδομένα και παρέχεται τυπικά από ένα Controller (στην αρχιτεκτονική MVC) μέσω ενός URI (end-point)



# Τύποι Web Services

Java EE – REST API

- **SOAP** (Simple Object Access Protocols): πρωτόκολλο που ορίζει ένα standard τρόπο μεταφοράς δεδομένων με XML
- **REST** (Representational State Transfer): Μία προσέγγιση για την μεταφορά δεδομένων μέσω HTTP (GET, POST, PUT, DELETE). Τα RESTful Web Services χρησιμοποιούν JSON ή XML για τη μεταφορά δεδομένων



# RESTful Web Services (1)

Java EE – REST API

- Όταν ένα Web Service ακολουθεί τα conventions της μεθοδολογίας REST (Representational State Transfer), ονομάζεται RESTful Web Service
- Παρόλο που η ίδια η μεθοδολογία REST δεν ορίζει συγκεκριμένα υποκείμενα πρωτόκολλα ή πακέτα μεταφοράς, τυπικά χρησιμοποιείται το πρωτόκολλο HTTP και τα JSON ή XML packets, αντίστοιχα





# RESTful Web Services (2)

Java EE – REST API

- Στην μεθοδολογία **REST** **έμφαση** **δίνεται** **στη** **μορφή** **των** **URIs** **που** **αντιστοιχούν** **σε** **πόρους** **του** **Server** **τα** **οποία** **πρέπει** **να** **ακολουθούν** **κάποια** **conventions** **καθώς** **και** **στις** **μεθόδους** **του** **πρωτοκόλλου** **HTTP** **που** **πρέπει** **να** **χρησιμοποιούνται** (όπως **GET**, **POST**, **PUT**, **DELETE**), **όπως** **θα** **δούμε** **παρακάτω**



# RESTful Web Services

Java EE – REST API

- Επομένως, όταν ένας Controller επιστρέφει, όχι Web σελίδες, αλλά **δεδομένα**, σε μία **Representational State Transfer (REST)** μορφή, όπως για παράδειγμα **JSON** ή **XML** μέσω **HTTP** τότε αναφερόμαστε σε **Restful Web Services** ή αλλιώς **REST API**
- Ενώ όταν η υπηρεσία παρέχεται μέσω του πρωτοκόλλου **SOAP** (Simple Object Access Protocol) αναφερόμαστε σε **SOAP Web Services** που χρησιμοποιούν **XML**



# Microservices

- Μεγάλες εφαρμογές μπορούν να αναλύονται σε ένα σύνολο από μικρότερα RESTful Web Services που μπορούν να τρέχουν ανεξάρτητα με τη δική τους ΒΔ και να επικοινωνούν μεταξύ τους με JSON
- Με τον όρο *Microservices* εννοούμε ανεξάρτητα μικρά RESTful Web services που μπορούν να γίνουν build, test & deploy ανεξάρτητα



# XML / JSON Μορφή

- Αν, για παράδειγμα, ζητήσουμε από ένα RESTful Web Service τα στοιχεία ενός Καθηγητή με `id=1`, τότε αυτά επιστρέφονται είτε σε JSON ή σε XML ως εξής:

## XML

```
<teacher>
  <id>1</id>
  <name>Th. Andr.</name>
</teacher>
```

## JSON

```
{
  "id": 1,
  "name": "Th. Andr."
}
```



# XML vs JSON

- Σχετικά με τις δύο μορφές αναπαράστασης δεδομένων XML και JSON για τη μεταφορά τους μέσω δικτύου είναι προτιμότερη η μορφή JSON γιατί:
  - είναι πιο σύντομη
  - μπορεί να αναπαραστήσει πίνακες
  - Το συντακτικό είναι παρόμοιο με της JavaScript
  - Μπορεί να επεξεργαστεί ευκολότερα με JavaScript



# REST

- Τυπικά, το Representational State Transfer (REST) είναι μία αρχιτεκτονική σχεδιασμού κατανεμημένων συστημάτων με βάση ένα πρωτόκολλο επικοινωνίας που συνήθως είναι το **HTTP** ενώ η επικοινωνία γίνεται με ανεξάρτητα tokens πληροφοριών συνήθως σε μορφή **JSON** ή **XML**
- Το REST έγινε introduce από τον Roy Fielding το 2000



# Resources (1)

- Το βασικό abstraction στο REST είναι ο όρος ***Resource*** (πόρος)
- Κάθε πληροφορία που έχει ***state*** και ***identity*** είναι ένα resource
- Για παράδειγμα ένα document, image, service, entity, ή και collection από resources, είναι ένα resource



# Resources (2)

- Κάθε resource **σχετίζεται με ένα URI** που πρέπει να αναγνωρίζει με μοναδικό τρόπο το resource
- Ένα URI (Uniform Resource Identifier) όπως έχουμε αναφέρει, είναι ένα string που περιέχει δύο μέρη: το ***scheme*** που είναι το πρωτόκολλο επικοινωνίας όπως το http ή https ακολουθούμενο από το ***domain name*** και προαιρετικά ***port***, το ***path***, και πιθανά μετά ένα ***query string***





# RESTful Web Services (1)

Java EE – REST API

- Η αρχιτεκτονική του REST είναι η **client-server**. Τυπικά, ένας JavaScript client επικοινωνεί με ένα Web Server που παρέχει ένα REST API
- Όταν ζητάμε μία υπηρεσία ενός REST API δηλαδή όταν ζητάμε ένα RESTful web service, - **συνήθως με JavaScript και AJAX**-, η AJAX κλήση περιλαμβάνει το Resource URI καθώς και το HTTP Method όπως: *GET, POST, PUT, DELETE*
- Το default method στις AJAX κλήσεις είναι το GET



# RESTful Web Services (2)

- Επομένως ένα web service που συμμορφώνεται στο REST API, δηλαδή παρέχει υπηρεσίες μέσω JSON ή XML και τυπικά μέσω HTTP αναφέρεται ως **RESTful Web Service**
- Η επικοινωνία του client με τον Server είναι όπως γνωρίζουμε είναι **stateless**
- Στο REST API ο Server μπορεί την 1<sup>η</sup> φορά να κάνει το authentication (π.χ. με user credentials) αλλά δεν κρατάει sessions (session objects), απλά επιστρέφει ένα authentication token (JSON web token / JWT) και τα HTTP requests στη συνέχεια περιέχουν όλη την πληροφορία που χρειάζεται για να γίνει authenticate το request



# Media Type (1)

- Το data format του REST είναι κατά βάση το MIME type του περιεχομένου (data, payload) του response που στέλνει ο Server προς τον client
- Το media type μπορεί να είναι:
  - **JSON** (JavaScript Object Notation) – Εύκολο και για humans για read/write και για machines για parse / generate. Είναι το πιο συχνά χρησιμοποιούμενο media type στο REST (application/json)
  - **XML** (Extensible Markup Language) – Χρησιμοποιείται στο REST API εναλλακτικά του JSON (application/xml)
  - **Binary Data** – non-textual data (images, audio, video)



# Media Type (2)

- Εφόσον έχουμε το πεδίο ***Accept*** στο request header και το ***Content-Type*** στο HTTP Response header καθώς και τα media types, δεν χρειάζεται να προσδιορίζεται το media type στο URL
- Κάθε Media Type ορίζει ένα τρόπο επεξεργασίας του resource. Αν για παράδειγμα το Content-Type είναι application/json τότε ξέρουμε ότι πρέπει να γίνει parse από JavaScript



# Σχεδιασμός REST API

Java EE – REST API

- Στο **REST API** κάθε **interaction** με τον **Server** είναι **stateless**, επομένως το REST είναι απλό, lightweight και fast!
- Ένα ιδιαίτερα σημαντικό σημείο είναι ο σχεδιασμός του REST API όσο αφορά την **ονοματοδοσία των URIs**
- Στα non-REST Web Services δεν έχουμε conventions για την ονοματοδοσία των URLs. Αντίθετα στο REST API ο σχεδιασμός του API σημαίνει και την ονοματοδοσία των URIs



# REST API Design (1)

- Για την ονοματοδοσία των URIs χρησιμοποιούμε **ουσιαστικά** (nouns) για να αναγνωρίσουμε resources. Ένα resource μπορεί να είναι **μοναδικό** (singular) ή **collection**
- Αν ο REST Controller επιστρέφει ένα collection των Teachers τότε το URI θα μπορούσε να είναι **/teachers**
- Αν επιστρέφει ένα teacher, τότε το URI θα μπορούσε να είναι **/teachers/{teacherId}** ή **/teachers/{uuid}** όπου το {teacherId} ή {uuid} είναι Path Parameters ή αλλιώς Path variables



# REST API Design (2)

- Επομένως ένα REST Resource **αναγνωρίζεται από το URL Path**
- Στο παράδειγμα `/teachers/{teacherId}` το `{teacherId}` είναι ένα path parameter. Ένα get request της μορφής `/teachers/1` είναι προτιμότερο από `/get-teacher?teacherId=1`
- Το 1<sup>ο</sup> αναγνωρίζει ένα πόρο (resource-oriented path), το 2<sup>ο</sup> περιγράφει μία μέθοδο με παράμετρο



# Singular & Collections API

Java EE – REST API

- Όπως αναφέραμε, αν το resource αφορά σε ένα collection αντικειμένων (π.χ. όλοι οι teachers), τότε μπορεί να αναπαρασταθεί με το URI **/teachers**
- Αν αφορά ένα teacher μπορεί να αναπαρασταθεί με το **/teachers/{teacherId}**
- Αν έχουμε subcollections μπορούμε να αναπαραστήσουμε με (για παράδειγμα)
  - /teachers/{teacherId}/students, ή
  - /teachers/{teacherId}/students/{studentId}





# API Design

- Είναι σημαντικό κατά το σχεδιασμό του API να χρησιμοποιούμε URIs που να αντιστοιχούν το resource model με απλότητα προς τους clients
- Όταν τα URIs έχουν ορθή ονοματοδοσία, τότε το API είναι εύκολο να χρησιμοποιηθεί, διαφορετικά μπορεί να είναι δύσκολο να χρησιμοποιηθεί



# Best Practices

- Όπως αναφέραμε, χρησιμοποιούμε **ουσιαστικά** για να εκφράσουμε resources αντί για ρήματα
- Για παράδειγμα αν θέλουμε την υπηρεσία της διαχείρισης καθηγητών μπορούμε να αντιστοιχίσουμε τα παρακάτω URI <http://www.aueb.gr/school-app/api/teachers> ή <http://api.aueb.gr/school-app/teachers>
- Για να αναφερθούμε σε ένα συγκεκριμένο καθηγητή θα μπορούσε να είναι `/school-app/teachers/{teacherId}`



# No-REST, no-conventions

- Όταν η αρχιτεκτονική δεν είναι rest θα μπορούσαμε να χρησιμοποιούμε URIs της μορφής /get-teacher ή /delete-teacher
- Δηλαδή, θα μπορούσαμε να χρησιμοποιούμε CRUD names στα URIs και να χρησιμοποιούμε την HTTP GET ή POST για να καλούμε την αντίστοιχη υπηρεσία στον Server



# CRUD names Anti-pattern

- Αντίθετα, στην αρχιτεκτονική REST δεν χρησιμοποιούμε CRUD names στα URIs

```
GET /deleteUser?id=1234  
GET /deleteUser/1234  
DELETE /deleteUser/1234  
POST /users/1234/delete
```



# API Design

- Χρησιμοποιούμε **hyphens (-)** για να αυξήσουμε το readability των URIs, **όχι underscores**
- Τα underscores (δεν επιτρέπονται στα URLs, δεν αναγνωρίζονται από την Google)
- Χρησιμοποιούμε **lowercase letters** ως convention μιας και το μέρος του path είναι case sensitive
- **Δεν χρησιμοποιούμε file extensions.** Τα file types είναι μέρος του Content-Type



# REST API Design

HTTP Method	URL Design	Περιγραφή
GET	api.aueb.gr/app/students	Get all students
GET	api.aueb.gr/app/students/{id}	Get one student
POST	api.aueb.gr/app/students	Insert a student
PUT	api.aueb.gr/app/students/{id}	Update student with id = {id}
DELETE	api.aueb.gr/app/students/{id}	Delete student with id = {id}



# Status Codes (1)

HTTP Method	/students	/Students/{id}
GET	200 (OK), list of students Use pagination, sorting and filtering to navigate big lists.	200 (OK), single student 404 (Not Found), if ID not found or invalid. 400 Bad Request.
POST	201 (Created), 'Location' header with link to /students/{id} containing new ID.	404 (Not Found). 400 Bad Request.
PUT	404 (Not Found), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. 400 Bad Request.
DELETE	404 (Not Found), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid. 400 Bad Request.

- Μαζί με τα data ο Server θα πρέπει να επιστρέφει και ένα **HTTP Response status code**, ώστε ο client να μπορεί να ελέγξει αν το request εξυπηρετήθηκε επιτυχώς ή όχι



# Status Codes (2)

HTTP Method	/students	/Students/{id}
GET	200 (OK), list of students Use pagination, sorting and filtering to navigate big lists.	200 (OK), single student 404 (Not Found), if ID not found or invalid. 400 Bad Request.
POST	201 (Created), 'Location' header with link to /students/{id} containing new ID.	404 (Not Found). 400 Bad Request.
PUT	404 (Not Found), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. 400 Bad Request.
DELETE	404 (Not Found), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid. 400 Bad Request.

- Στα παραδείγματα παραπάνω, μπορούμε να έχουμε και 401 Unauthorized





# Query String για filtering

Java EE – REST API

- Στα query strings μπορούμε να έχουμε search criteria για να κάνουμε **filtering** τα αποτελέσματα
- `api.cf.aueb.gr/school-app/students?region=GR`
- `api.cf.aueb.gr/school-app/students?region=GR&sort=lastname`
- `api.cf.aueb.gr/school-app/students?region=GR&city=Athens&sort=year`



# Form names

- Αν είχαμε φόρμες για insert, update, delete θα μπορούσαμε να τις ονομάζουμε ως nested του path
- Για παράδειγμα
  - `api.aueb.gr/app/students/insert`
  - `api.aueb.gr/app/students/{id}/edit`
  - `api.aueb.gr/app/students/{id}/remove`



# Java EE - JAX-RS

- Θα δούμε στα επόμενα εφαρμογές των RESTful Web Services. Η Jakarta EE παρέχει το JAX-RS (Java API for RESTful Web Services) για τα **specs** του API των RESTful Web services
- Ορίζει δηλαδή ένα set από annotations και interfaces που μπορούν να χρησιμοποιηθούν για να δημιουργήσουμε Web Services
- Οι βασικές υλοποιήσεις του JAX-RS είναι τα [Jersey](#) και [RESTEasy](#) όπου μπορείτε να ανατρέξετε



# IntelliJ JAX-RS (1)

Java EE – REST API

New Project

Generators

- Maven Archetype
- Jakarta EE**
- Spring Initializr
- JavaFX
- Quarkus
- Micronaut
- Ktor
- Compose for Desktop
- HTML
- React
- Express
- Angular CLI
- Vue.js
- Vite

Name: jax6-test

Location: ~\IdeaProjects

Project will be created in: ~\IdeaProjects\jax6-test

☐ Create Git repository

Template: REST service JAX-RS resource

Application server: Tomcat 10.1.18 New...

Language: Java Kotlin Groovy

Build system: Maven Gradle

Group: gr.aueb.cf

Artifact: jax6-test

JDK: corretto-17 java version "17.0.6"

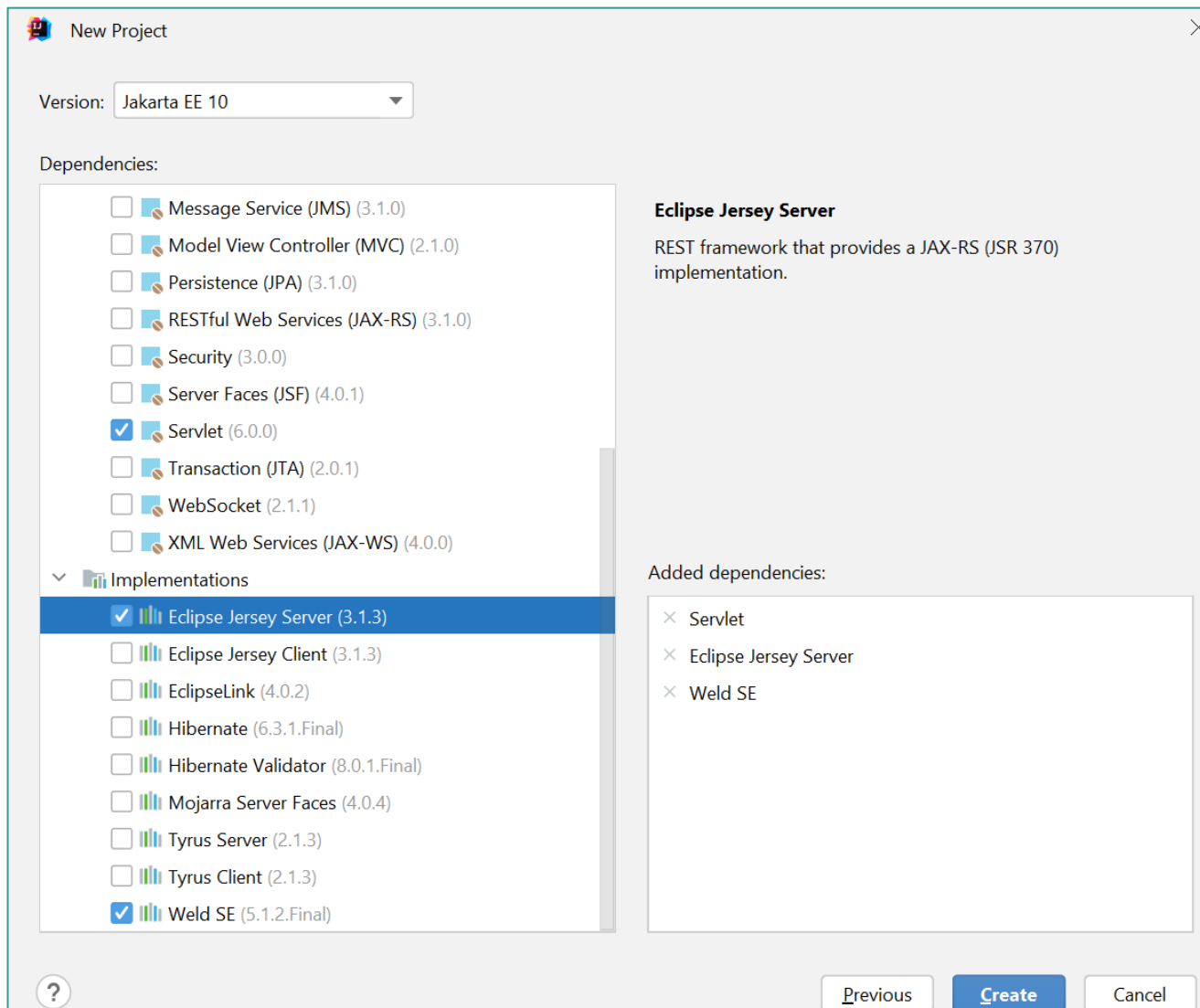
Next Cancel

- New Java Jakarta EE Project με Maven και Apache Tomcat 10.1
- Ως Template επιλέγουμε **REST service**



# IntelliJ JAX-RS (2)

Java EE – REST API



- Επιλέγουμε Jakarta EE 10 και ως dependencies έχουμε τα **Servlet, Jersey και Weld**



# POM.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven
5         <modelVersion>4.0.0</modelVersion>
6
7         <groupId>gr.aueb.cf</groupId>
8         <artifactId>jax6-test</artifactId>
9         <version>1.0-SNAPSHOT</version>
10        <name>jax6-test</name>
11        <packaging>war</packaging>
12
13        <properties>
14            <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15            <maven.compiler.target>17</maven.compiler.target>
16            <maven.compiler.source>17</maven.compiler.source>
17            <junit.version>5.10.0</junit.version>
18        </properties>
19
```

- Αλλάζουμε το compiler target και source στο 17



# Dependencies (1)

```
20 <dependencies>
21   <dependency>
22     <groupId>jakarta.servlet</groupId>
23     <artifactId>jakarta.servlet-api</artifactId>
24     <version>6.0.0</version>
25     <scope>provided</scope>
26   </dependency>
27   <dependency>
28     <groupId>org.glassfish.jersey.containers</groupId>
29     <artifactId>jersey-container-servlet</artifactId>
30     <version>3.1.3</version>
31   </dependency>
```

- Το jakarta-servlet-api είναι η βιβλιοθήκη για τα servlets και είναι provided από τον Apache Tomcat (Το JAX-RS είναι υλοποιημένο on top of servlet technology)
- Το Jersey είναι η βασική υλοποίηση Jakarta RESTful Web Services - JAX-RS (Το JAX-RS 3.1.\* είναι μέρος της Jakarta EE 10 και η αντίστοιχη υλοποίηση Jersey 3.1.3) και το εισάγουμε με το jersey-container-servlet



# Dependencies (2)

```
32 <dependency>
33   <groupId>org.glassfish.jersey.media</groupId>
34   <artifactId>jersey-media-json-jackson</artifactId>
35   <version>3.1.3</version>
36 </dependency>
37 <dependency>
38   <groupId>org.glassfish.jersey.inject</groupId>
39   <artifactId>jersey-cdi2-se</artifactId>
40   <version>3.1.3</version>
41 </dependency>
42 <dependency>
43   <groupId>org.jboss.weld.se</groupId>
44   <artifactId>weld-se-core</artifactId>
45   <version>5.1.2.Final</version>
46 </dependency>
```

- Το Jackson δίνει ένα API για mapping Java objects σε JSON Strings και το αντίθετο. Ενσωματώνεται στο Jersey με το jersey-media-json-Jackson
- Το jersey-cdi2-se αφορά dependency injection με το annotation @Inject
- Για να δουλέψει σωστά το jersey-cdi στον Tomcat χρειάζεται config που το δίνει ο weld-se-core





# Application class (1)

```
1 package gr.aueb.cf.jaxbtest;
2
3 import jakarta.ws.rs.ApplicationPath;
4 import jakarta.ws.rs.core.Application;
5
6 @ApplicationPath("/api")
7 public class HelloApplication extends Application {
8
9 }
```

- Η κλάση *HelloApplication* κάνει extends την abstract κλάση *jakarta.ws.rs.Application*, και **είναι το entry point** μίας **JAX-RS εφαρμογής** κάνοντας config τον Apache Tomcat ώστε να κάνει scan τις κλάσεις (resources, providers, filters) που συνθέτουν ένα JAX-RS app
- Πρέπει να βρίσκεται στο root package ώστε να σκανάρει τα περιεχόμενα αυτού του package και των subpackages



# Application class (2)

```
1 package gr.aueb.cf.jaxbtest;  
2  
3 import jakarta.ws.rs.ApplicationPath;  
4 import jakarta.ws.rs.core.Application;  
5  
6 @ApplicationPath("/api")  
7 public class HelloApplication extends Application {  
8  
9 }
```

- Όταν δεν έχει σώμα, σκανάρει το ίδιο package και όλα τα subpackages για JAX-RS components. Εναλλακτικά, μπορούμε να ορίσουμε συγκεκριμένες κλάσεις
- Πρέπει επίσης να ορίσουμε το **@ApplicationPath** που είναι ένα διακριτό path μέσα στο οποίο τρέχει όλο το REST API



# Application class (3)

```
1 package gr.aueb.cf.jaxbtest;  
2  
3 import jakarta.ws.rs.ApplicationPath;  
4 import jakarta.ws.rs.core.Application;  
5  
6 @ApplicationPath("/api")  
7 public class HelloApplication extends Application {  
8  
9 }
```

- Μπορεί το `@ApplicationPath` να είναι `/` αν όλο το domain εξυπηρετεί REST API διαφορετικά αν η εφαρμογή περιλαμβάνει και non-REST API το convention είναι `/api` για REST Services



# HelloRestController

Java EE – REST API

```
1 package gr.aueb.cf.jax.rest;
2
3 import jakarta.ws.rs.GET;
4 import jakarta.ws.rs.Path;
5 import jakarta.ws.rs.Produces;
6 import jakarta.ws.rs.core.MediaType;
7
8 @Path("/greetings")
9 public class HelloRestController {
10
11     @GET
12     @Path("/hello")
13     @Produces(MediaType.TEXT_PLAIN)
14     public String hello() {
15         return "Hello greetings from CF6";
16     }
17 }
```

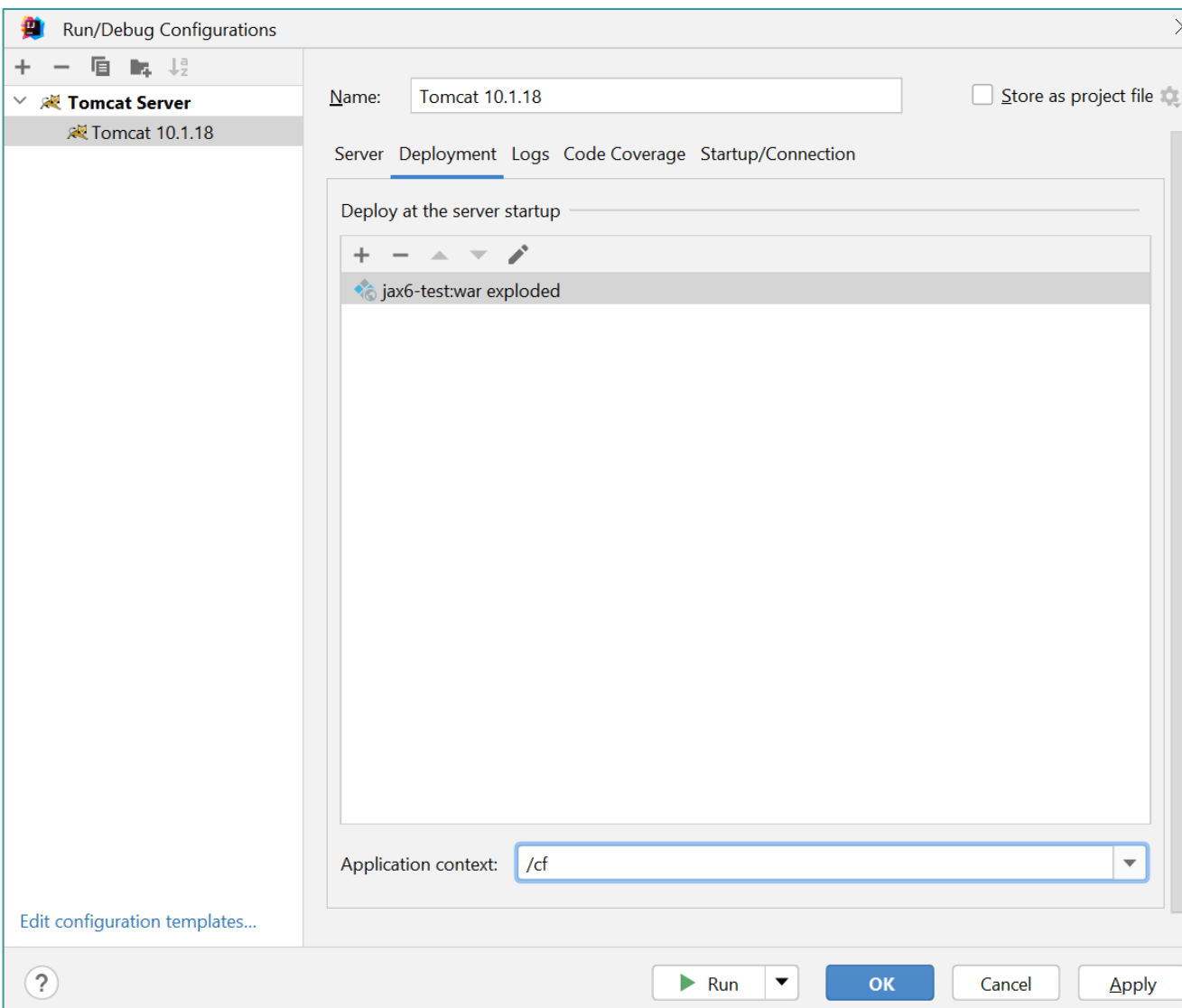
- Ορίζουμε ένα RestController (μέσα στο package rest) με **@Path("/greetings")**. Και μέσα ορίζουμε controllers (ή αλλιώς Resources)
- Ορίζουμε επίσης ένα end-point με **@Path("/hello")** στη μέθοδο hello(). Η μέθοδος είναι **@GET**. Και επιστρέφει **@Produces**, text/plain (media, ContentType). Ο επιστρεφόμενος τύπος της μεθόδου είναι String
- Το τελικό resource path (REST end-point) είναι **/api/greetings/hello**



# IntelliJ deployment config

Java EE – REST API

- Ορίζουμε το `/cf` ως το root context





# Maven war filename

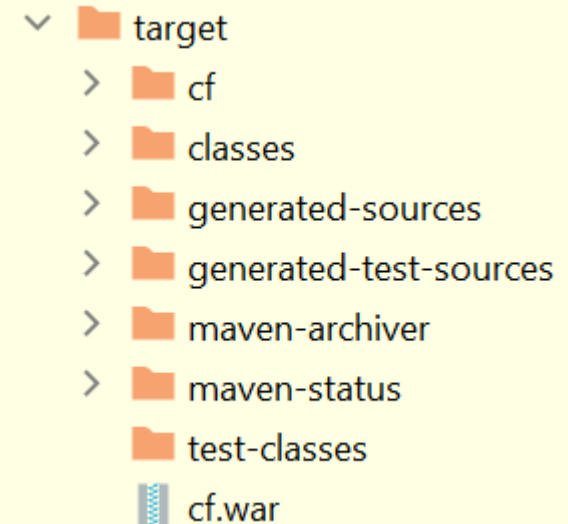
```
61      <build>
62          <finalName>cf</finalName>
63          <plugins>
64              <plugin>
65                  <groupId>org.apache.maven.plugins</groupId>
66                  <artifactId>maven-war-plugin</artifactId>
67                  <version>3.4.0</version>
68              </plugin>
69          </plugins>
70      </build>
71  </project>
```



# Maven package

Terminal: Git Bash × + ▾

a8ana@thanassis-pc MINGW64 ~/IdeaProjects/jax6-test  
\$ mvn clean package

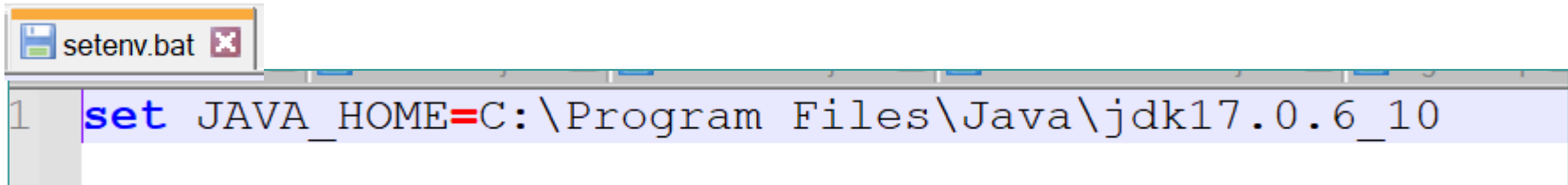


- Με `mvn clean package` δημιουργούμε ένα war με όνομα `cf.war`

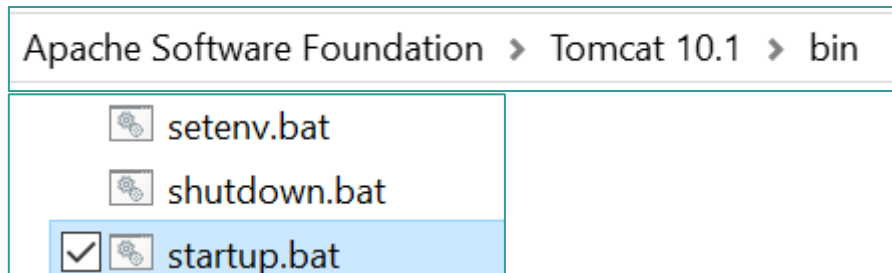


# Tomcat starts standalone

Java EE – REST API



```
1 set JAVA_HOME=C:\Program Files\Java\jdk17.0.6_10
```



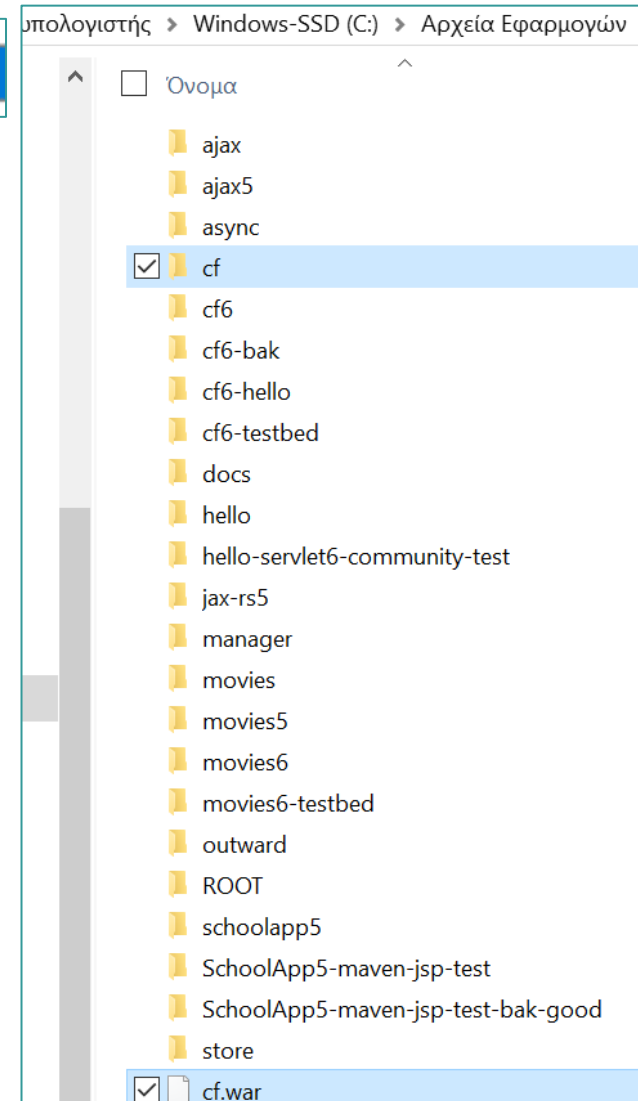
- Αφού ορίσουμε στο setenv του Tomcat το jdk17 εκκινούμε τον Tomcat με startup





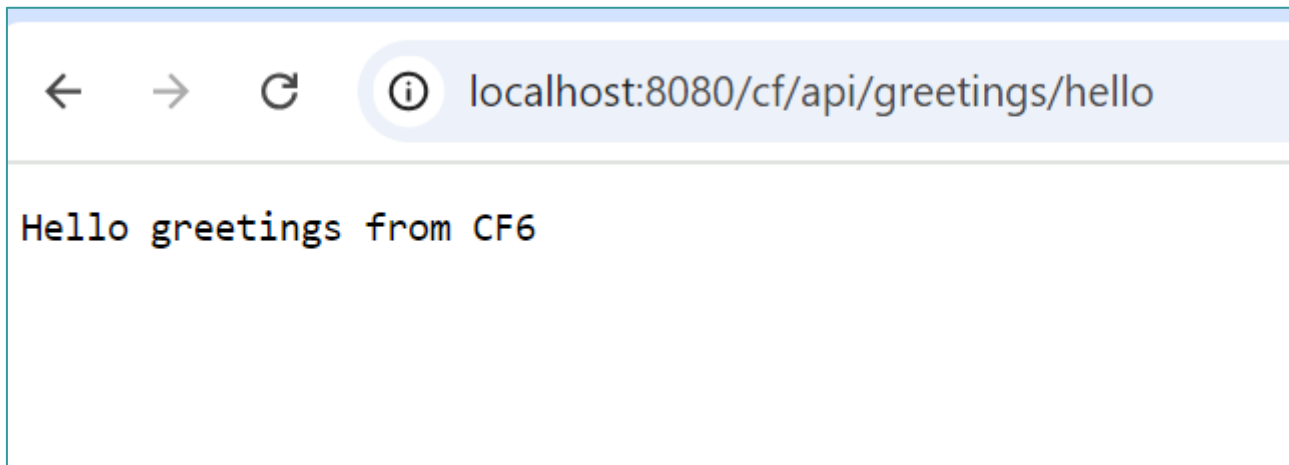
# Apache Tomcat

- Στον webapps folder εισάγουμε το .war και αυτόματα δημιουργείται ο αντίστοιχος folder





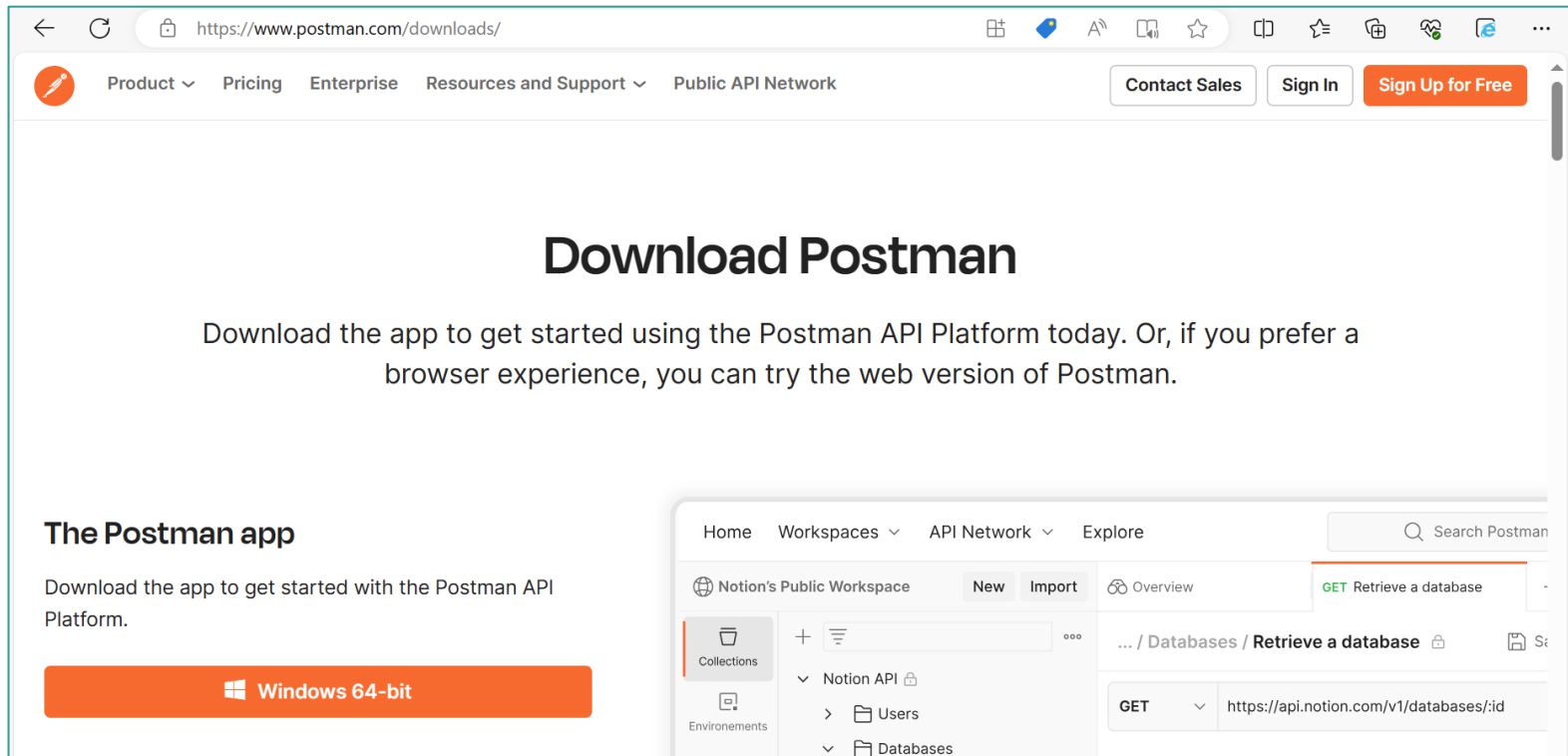
# Browser



- Το string που επιστρέψαμε εμφανίζεται και στον Browser αλλά συνήθως για να εμφανίζουμε αποτελέσματα από Web Services χρησιμοποιούμε JavaScript
- Για testing των endpoints χρησιμοποιούμε εργαλεία όπως το Postman



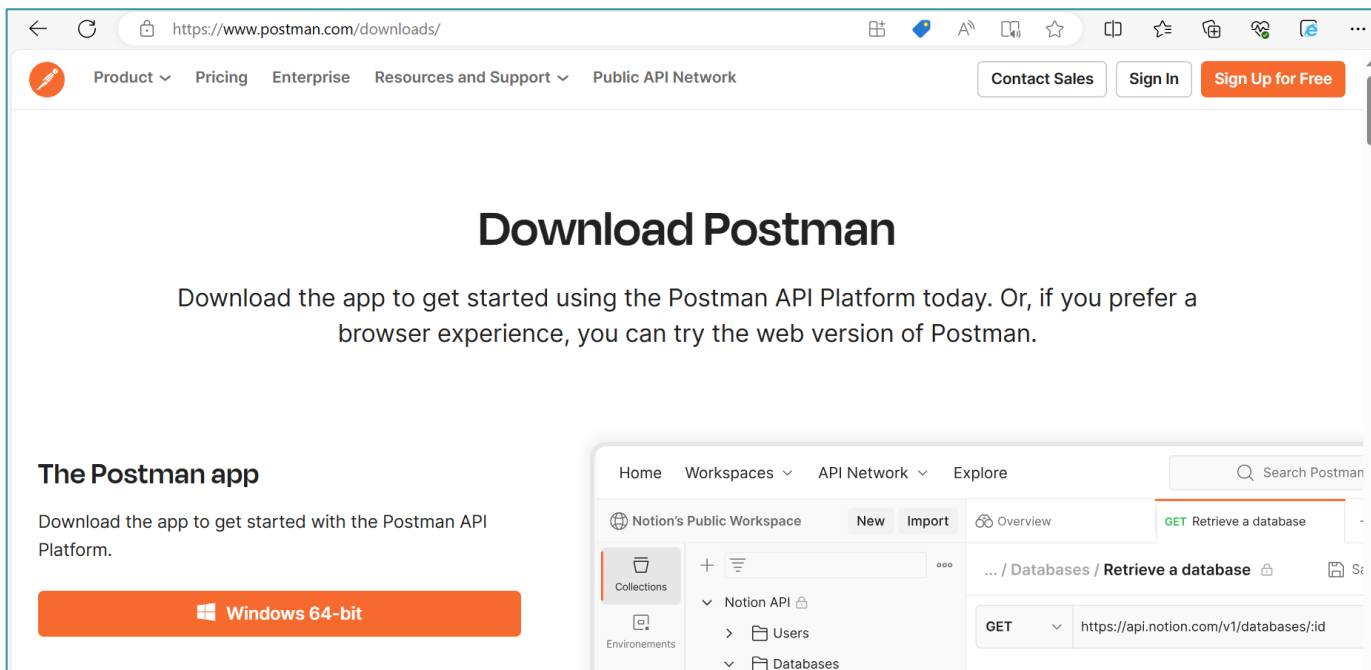
# Postman (1)



- Ο **Postman** είναι testing tool για REST APIs, τον οποίο και θα χρησιμοποιήσουμε ως client για να τεστάρουμε τα RESTful Web Service μας



# Postman (2)



- Μπορούμε να κάνουμε **download** και να εγκαταστήσουμε τοπικά τον postman
- Μπορούμε να κάνουμε **sign up** με τον google account ώστε να συγχρονίζετε το τοπικό postman με το cloud



# Postman Environments (1)

Java EE – REST API

The screenshot shows the Postman interface with the 'Environments' tab selected. The left sidebar has a red circle around the 'Environments' icon. The main panel displays the 'JAX6-Test' environment. A table lists variables, with 'base\_url' checked and having a value of 'http://localhost:8080/cf'.

Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> base_url	default	http://localhost:8080/cf	http://localhost:8080/cf
Add new variable			

- Στο Postman Environments μπορούμε να δημιουργήσουμε ένα environment. Τα environments ορίζουν context και config σε μορφή key-value που αφορούν ένα Collection από API endpoints
- Με το + δημιουργούμε ένα environment με όνομα JAX-RS-Test και θα ορίσουμε στη συνέχεια μεταβλητές, όπως το **base\_url** με value το context path: `http://localhost:8080/cf`
- Στη συνέχεια μπορούμε να **επιλέξουμε ένα environment** και να αναφερόμαστε στη μεταβλητή με **{{base\_url}}**



# Postman Environments (2)

Java EE – REST API

- Το initial value αφορά την αρχική και default τιμή της μεταβλητής και αν το environment γίνει export θα περιλαμβάνεται το default value
- Το Current value είναι για testing λόγους, αν κάποιος από τους collaborators θελήσει να αλλάξει το value τοπικά χωρίς να επηρεάσει το initial value



# Collections (1)

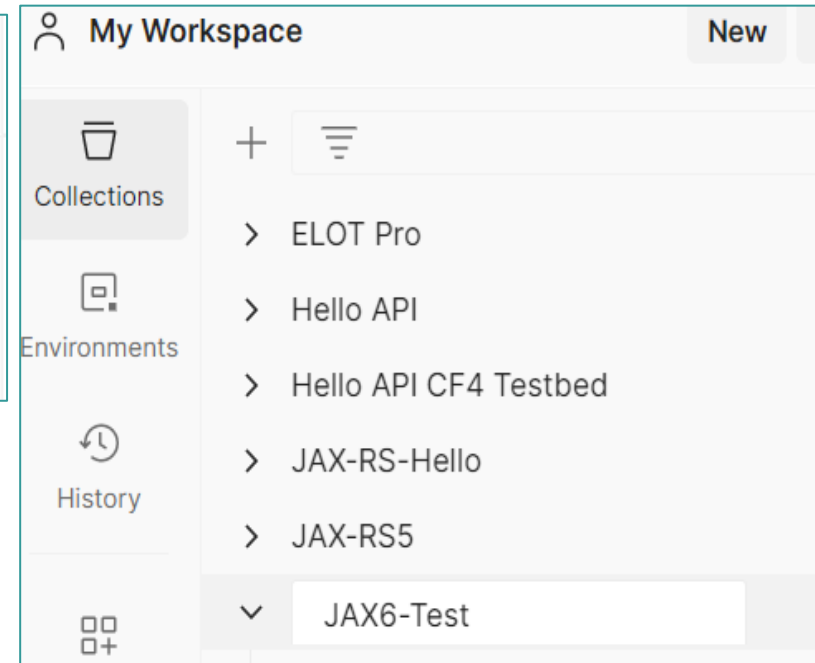
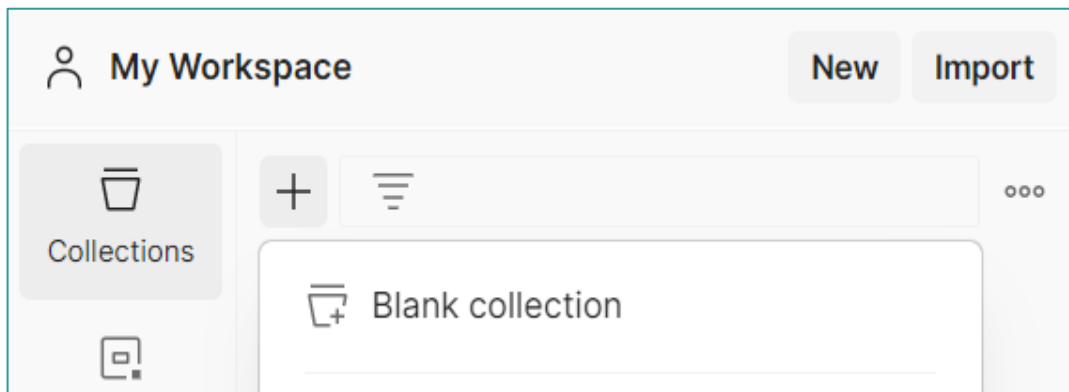
The screenshot shows the IntelliJ IDEA IDE interface. On the left, the 'My Workspace' sidebar is visible. It has a 'Collections' icon (a trash can) circled in red. Below it, the 'Environments' section lists several environments: 'Globals', 'JAX-RS-Hello', 'JAX-RS5', 'JAX6-Test' (which is selected and circled in red), and 'Schoolapp5'. The main area of the IDE shows the 'JAX6-Test' environment. At the top of this area, there is a toolbar with icons for 'Fork', 'Save', 'Share', and others. The 'JAX6-Test' tab is circled in red. Below the toolbar, there is a table of variables.

Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> base_url	default	http://localhost:8080/cf	http://localhost:8080/cf
Add new variable			

- Μπορούμε να επιλέξουμε το environment (πάνω δεξιά) και να ορίσουμε επίσης ένα Collection (πάνω αριστερά στα Collections) ώστε να οργανώσουμε τα test



# Collections (2)

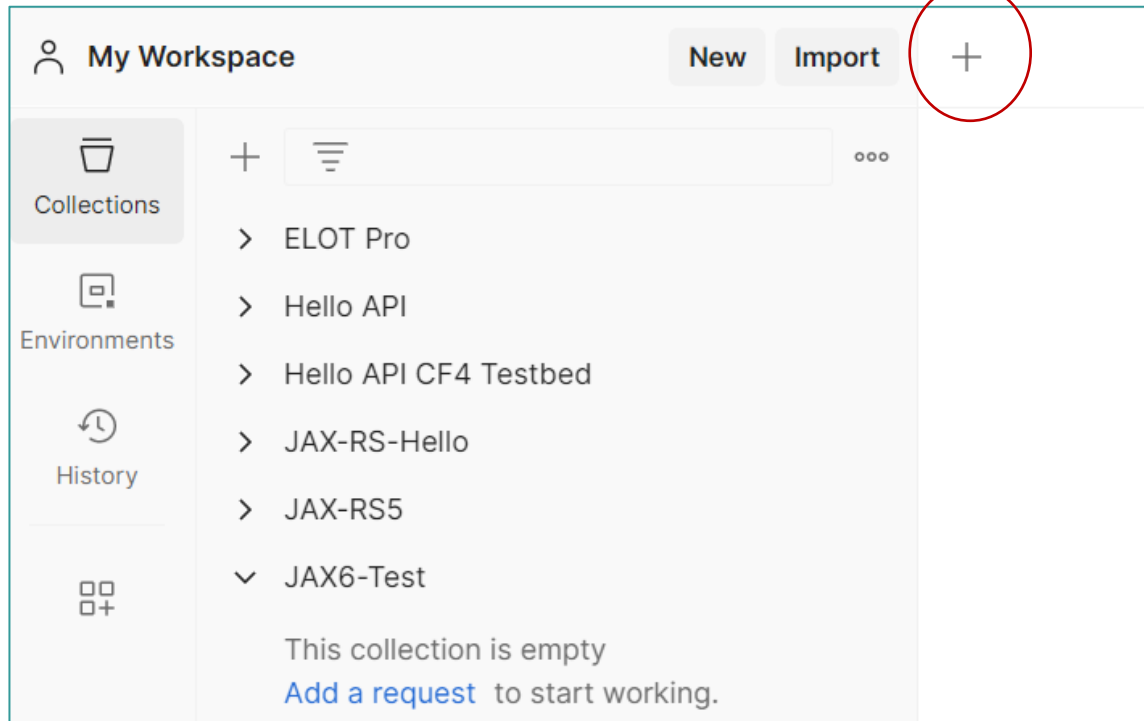


- Ορίζουμε ένα Blank collection και το κάνουμε Rename JAX6-Test





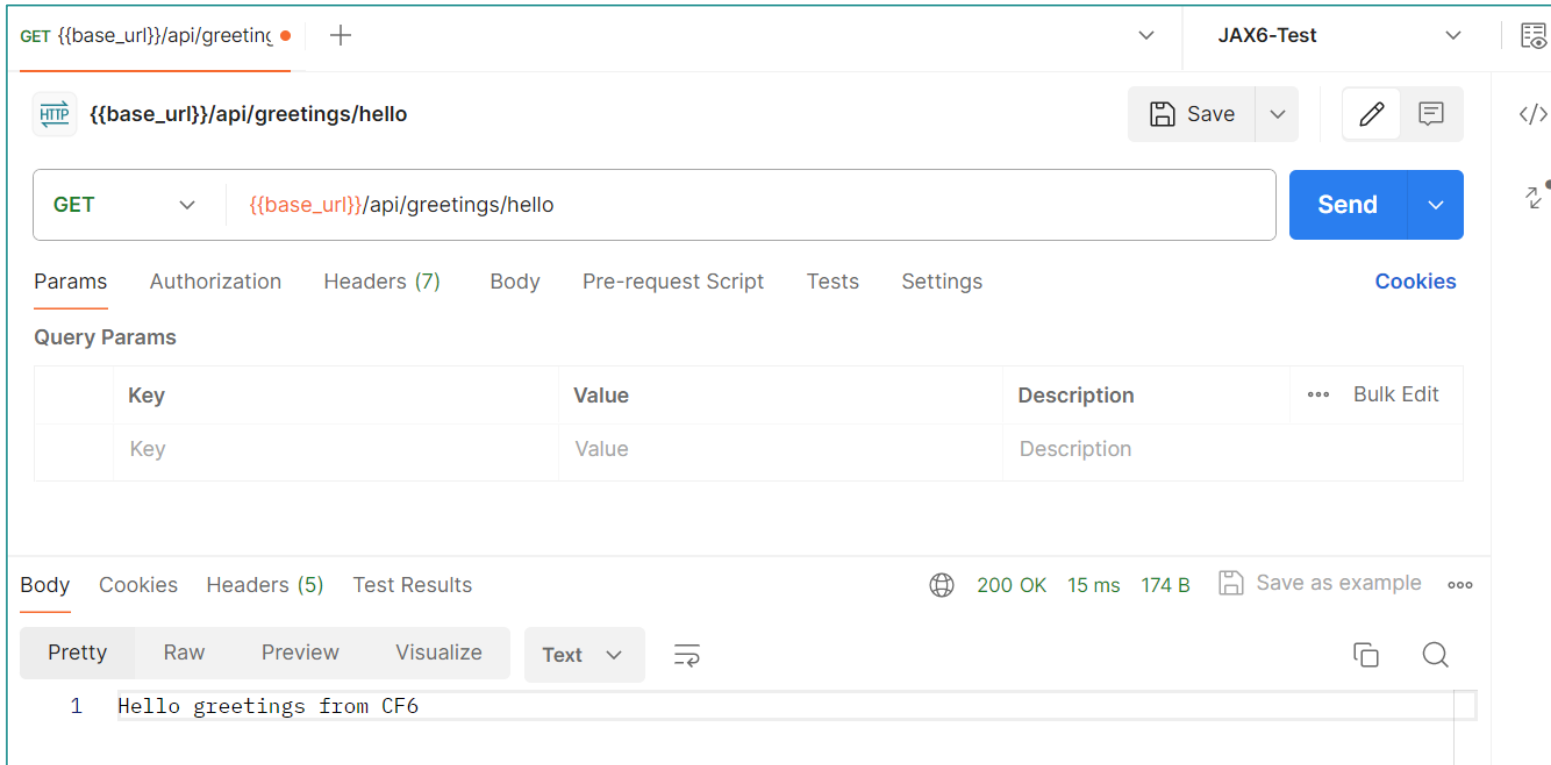
# Postman Request



- Μπορούμε να προσθέσουμε ένα request που είναι η βασική δομή του Postman για να στέλνουμε requests σε APIs



# Postman Request



- Στη γραμμή του request ορίζουμε το HTTP Method, εδώ **GET**, και το **URI**, με το `{{base_url}}/api/greetings/hello`



# Postman – API Testing

Java EE – REST API

GET {{base\_url}}/api/greeting

HTTP {{base\_url}}/api/greetings/hello

GET {{base\_url}}/api/greetings/hello

Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

200 OK 15 ms 174 B Save as example

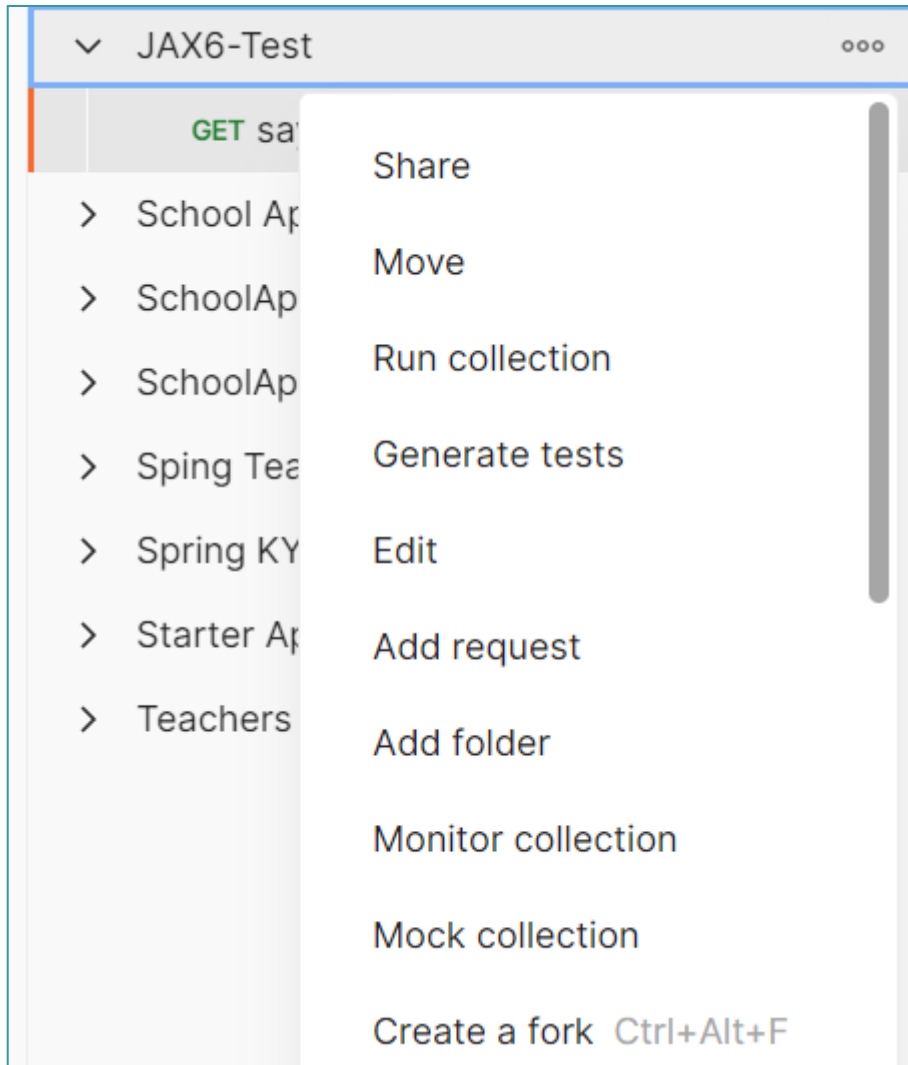
Pretty Raw Preview Visualize Text

1 Hello greetings from CF6

- Το **/api/greetings/hello** είναι ένα **REST Endpoint** το οποίο μπορούμε να τεστάρουμε με Postman. Επιλέγουμε GET καθώς και το URI του Resource και πατάμε Send. Επιστρέφεται το string 'Hello greetings from CF6'



# Tests με Postman (1)



- Στο Collections μπορούμε να προσθέσουμε requests με δεξί κλικ στο Collection (JAX6-Test) και Add request
- Μπορούμε επίσης να δημιουργήσουμε ένα folder (με ... ή δεξί κλικ στο JAX6-Test και Add Folder) και μετά να προσθέσουμε requests (Add request)



# Tests με Postman (2)

Java EE – REST API

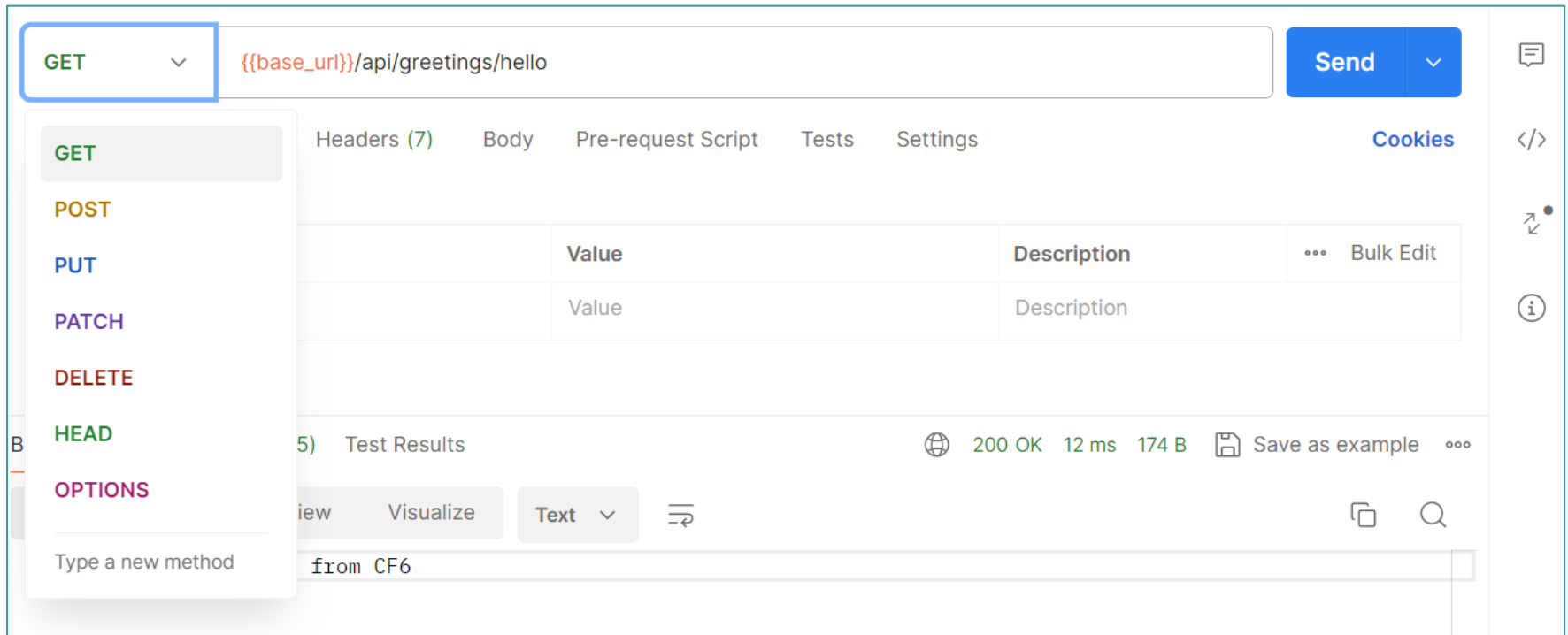
The screenshot shows the Postman interface. On the left, a collection named 'JAX6-Test' is expanded, showing a request named 'GET say hello'. The main panel displays the details of this request:

- Method: GET
- URL: `{{base_url}}/api/greetings/hello`
- Params: Authorization, Headers (7), Body
- Query Params: A table with two rows, each with a 'Key' column.

- Αποθηκεύουμε στο collection το GET Request ως say hello



# Requests



- Στο Postman, επιλέγουμε το HTTP Action του request όπως GET, POST, PUT, DELETE, κλπ. μετά το URI, και μετά πιθανόν αν υπάρχουν τα Params (Query String), Authorization, τους Headers και το Body



# TeacherReadOnlyDTO

Java EE – REST API

```
1 package gr.aueb.cf.jax.dto;
2
3 import lombok.*;
4
5 @NoArgsConstructor
6 @AllArgsConstructor
7 @Getter
8 @Setter
9 @Builder
10 public class TeacherReadOnlyDTO {
11
12     private Long id;
13     private String firstname;
14     private String lastname;
15 }
```

- Έστω ένα TeacherRead OnlyDTO Entity
- Έχουμε δημιουργήσει με Lombok constructors, getters, setters και builder



# Response

```
20  @GET
21  @Path("/hello-json")
22  @Produces(MediaType.APPLICATION_JSON)
23  public Response helloJSON() {
24      TeacherReadOnlyDTO dto = TeacherReadOnlyDTO.builder()
25          .id(1L)
26          .firstname("Αθανάσιος")
27          .lastname("Ανδρούτσος")
28          .build();
29
30      return Response.ok(dto).build();
31  }
```

- Συνήθως ο επιστρεφόμενος τύπος από ένα REST endpoint δεν είναι ένα String αλλά ένα **Response instance**. Το Response είναι μία abstract κλάση που υλοποιείται από ένα *ResponseBuilder* με τη μέθοδο build() – γραμμή 30
- Η **.ok()** δημιουργεί ένα response με status OK ενώ ως payload (data) εισάγει ένα TeacherReadOnlyDTO instance, που έχει δημιουργηθεί με το builder pattern. Μετά κάνουμε build() στο response.ok()





# ok() vs status() , entity()

```
20 @GET
21 @Path("/hello-json")
22 @Produces(MediaType.APPLICATION_JSON)
23 public Response helloJSON() {
24     TeacherReadOnlyDTO dto = TeacherReadOnlyDTO.builder()
25         .id(1L)
26         .firstname("Αθανάσιος")
27         .lastname("Ανδρούτσος")
28         .build();
29
30     // return Response.ok(dto).build();
31     return Response
32         .status(Response.Status.OK)
33         .entity(dto)
34         .build();
35 }
```

- Η **Response.ok()** είναι ισοδύναμη με την **Response.status (Status.OK).entity (entity)**
- Η παρούσα μορφή είναι περισσότερο readable



# Java / JSON Serialization

```
20  @GET
21  @Path("/hello-json")
22  @Produces(MediaType.APPLICATION_JSON)
23  public Response helloJSON() {
24      TeacherReadOnlyDTO dto = TeacherReadOnlyDTO.builder()
25          .id(1L)
26          .firstname("Αθανάσιος")
27          .lastname("Ανδρούτσος")
28          .build();
29
30      // return Response.ok(dto).build();
31      return Response
32          .status(Response.Status.OK)
33          .entity(dto)
34          .build();
35  }
```

- Αν εισάγουμε στην `ok()` ή στην `entity()` ένα Java instance και έχουμε ένα **runtime JSON mapper** όπως ο **Jackson** μπορούμε να μετατρέπουμε αυτόματα Java instances σε **JSON Strings** και το αντίθετο



# Εκτέλεση

The screenshot shows the IntelliJ IDEA IDE interface. The main editor window displays the `HelloRestController.java` file. The code includes annotations for a REST endpoint: `@GET`, `@Path("/hello-json")`, and `@Produces(MediaType.APPLICATION_JSON)`. The `helloJSON()` method returns a `Response` object, which is built using a `TeacherReadOnlyDTO` instance created by `TeacherReadOnlyDTO.builder()`. The DTO is populated with values for `id`, `firstname`, and `lastname`. The response is then built and returned with a status of `Response.Status.OK`.

The left sidebar shows the project structure for `jax6-test`. The `rest` folder contains the `HelloRestController` and `HelloApplication` classes. The `dto` folder contains the `TeacherReadOnlyDTO` class.

The bottom panel shows the `Services` tab with a `Tomcat 10.1.18` server running. The `Tomcat Catalina Log` displays the following logs:

```
19-Sep-2024 22:09:24.176 WARNING [RMI TCP Connection(2)-127.0.0.1] org.glassfish.je
19-Sep-2024 22:09:24.480 INFO [RMI TCP Connection(2)-127.0.0.1] org.jboss.weld.boot
19-Sep-2024 22:09:24.668 INFO [RMI TCP Connection(2)-127.0.0.1] org.jboss.weld.envi
19-Sep-2024 22:09:24.807 INFO [RMI TCP Connection(2)-127.0.0.1] org.jboss.weld.boot
19-Sep-2024 22:09:25.494 INFO [RMI TCP Connection(2)-127.0.0.1] org.jboss.weld.envi
[2024-09-19 10:09:25,651] Artifact jax6-test:war exploded: Artifact is deployed suc
[2024-09-19 10:09:25,651] Artifact jax6-test:war exploded: Deploy took 3,398 millis
19-Sep-2024 22:09:31.859 INFO [Catalina-utility-2] org.apache.catalina.startup.Host
19-Sep-2024 22:09:31.973 INFO [Catalina-utility-2] org.apache.catalina.startup.Host
```

- Εκτελούμε στο IntelliJ



# New Request – say hello JSON

Java EE – REST API

The screenshot displays the JAX6-Test REST client interface. On the left, the 'My Workspace' sidebar shows a tree of collections and environments. The 'JAX6-Test' collection is expanded, showing two requests: 'GET say hello' and 'GET say hello JSON'. The 'GET say hello JSON' request is selected. The main panel shows the request configuration for 'GET say hello JSON' with the URL 'http://localhost:8080/cf/api/greetings/hello-json'. The 'Send' button is visible. Below the URL bar, tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Scripts', 'Tests', and 'Settings' are shown. The 'Query Params' table is empty. The 'Body' tab is active, showing a JSON response in 'Pretty' format: 

```
1 {
2   "id": 1,
3   "firstname": "Αθανάσιος",
4   "lastname": "Ανδρούτσος"
5 }
```

 The status bar at the bottom indicates a '200 OK' response with a 40 ms duration and 231 B size.

- Δημιουργούμε ένα νέο request και το μετονομάζουμε σε say hello JSON και εκτελούμε το GET request στο endpoint **http://localhost:8080/cf/api/greetings/hello-json**



# Get με Query String (1)

```
40 @GET
41 @Path("/teachers")
42 @Produces(MediaType.APPLICATION_JSON)
43 public Response getTeachers(@QueryParam("city") String city) {
44     // List<TeacherReadOnlyDTO> = teacherService.getTeachersByCity(city)
45     // Assume teacher service returns DTO
46     List<TeacherReadOnlyDTO> teacherReadOnlyDTOs = List.of(
47         new TeacherReadOnlyDTO(1L, "Alice", "W."),
48         new TeacherReadOnlyDTO(2L, "Bob", "D.));
49
50     return Response.status(Response.Status.OK).entity(teacherReadOnlyDTOs).build();
51
52 }
53 }
```



# Get με Query String (2)

Java EE – REST API

HTTP JAX6-Test / get teachers by city Save Share

**GET** http://localhost:8080/cf/api/greetings/teachers?city=Athens Send

Params • Authorization Headers (7) Body Scripts Tests Settings Cookies

**Query Params**

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	city	Athens			
	Key	Value	Description		

Body Cookies Headers (5) Test Results 200 OK • 6 ms • 245 B • ...

Pretty Raw Preview Visualize JSON ≡ 📄 🔍

```
1  [
2    {
3      "id": 1,
4      "firstname": "Alice",
5      "lastname": "W."
6    },
7    {
8      "id": 2,
9      "firstname": "Bob",
10     "lastname": "D."
11   }
12 ]
```



# Bad Request

```
@GET
@Path("/teachers")
@Produces(MediaType.APPLICATION_JSON)
public Response getTeachers(@QueryParam("city") String city) {
    if (city == null || city.isEmpty()) {
        return Response.status(Response.Status.BAD_REQUEST).build();
    }
    // List<TeacherReadOnlyDTO> = teacherService.getTeachersByCity(city)
    // Assume teacher service returns DTO
    List<TeacherReadOnlyDTO> teacherReadOnlyDTOs = List.of(
        new TeacherReadOnlyDTO(1L, "Alice", "W."),
        new TeacherReadOnlyDTO(2L, "Bob", "D."));

    return Response.status(Response.Status.OK).entity(teacherReadOnlyDTOs).build();
}
```

- Ελέγχουμε αν το query param υπάρχει. Αν όχι επιστρέφεται 400 (Bad Request)



# Path variable (1)

```
55  @GET
56  @Path("/teachers/{id}")
57  @Produces(MediaType.APPLICATION_JSON)
58  public Response getTeacher(@PathParam("id") Long id) {
59      // assume we get a TeacherReadOnlyDTO teacherDto = teacherService.getTeacher(id)
60      TeacherReadOnlyDTO dto = new TeacherReadOnlyDTO(1L, "Αθανάσιος", "Ανδρούτσος");
61      return Response.status(Response.Status.OK).entity(dto).build();
62
63  }
```





# Path variable (2)

HTTP JAX6-Test / get a teacher with id Save Share

GET ▼ `{{base_url}}/api/greetings/teachers/1` Send ▼

Params Authorization Headers (7) Body Scripts Tests Settings Cookies </>

Query Params

	Key	Value	Description	... Bulk Edit
--	-----	-------	-------------	---------------

Body Cookies Headers (5) Test Results 200 OK • 575 ms • 231 B • 🌐 e.g. ...

Pretty Raw Preview Visualize JSON ▼ ≡ 📄 🔍

```
1 {  
2   "id": 1,  
3   "firstname": "Αθανάσιος",  
4   "lastname": "Ανδρούτσος"  
5 }
```



# Data Binding

- Στα προηγούμενα παραδείγματα ο client λάμβανε δεδομένα με GET
- Στη συνέχεια θα δούμε πως ο client στέλνει δεδομένα τα οποία λαμβάνει ο REST Controller και τα αντιστοιχεί σε δικές του μεταβλητές
- Το θέμα λοιπόν που θα μας απασχολεί είναι το **data binding δεδομένων από τον client είτε από φόρμες ή από JSON strings σε μεταβλητές και DTOs της εφαρμογής μας**
- Όταν ο Controller λαμβάνει δεδομένα για insert/update, όπως γνωρίζουμε, θα πρέπει να τα κάνει validate



# Hibernate Validator

Java EE – REST API

```
<!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>8.0.1.Final</version>
</dependency>
```

- Το Bean Validation 2.0 είναι μέρος της Java EE 8 ενώ το **Bean Validation 3.0 είναι μέρος της Jakarta EE 9 και 10.** Παρέχει annotations για να κάνουμε validate τα DTOs
- Ο **Hibernate Validator 8.0.0.Final και 8.0.1.Final** είναι το reference implementation του Bean Validation 3.0 <https://beanvalidation.org/3.0/>



# Data Binding – DTO

```
1 package gr.aueb.cf.jax.dto;
2
3 import ...
4
11
12 @NoArgsConstructor
13 @AllArgsConstructor
14 @Getter
15 @Setter
16 public class UserInsertDTO {
17     @Email(message = "Invalid e-mail")
18     private String username;
19
20     @Pattern(regexp = "^(?=.*?[a-z])(?=.*?[A-Z])(?=.*?\\d)(?=.*?[@$!%*?&]).{8,}$", message = "Invalid password")
21     private String password;
22 }
```

- Έστω το UserDTO που αντιστοιχεί σε ένα user για insert
- Έχουμε εισάγει bean validations με **Java bean Validation** που υλοποιείται από τον Hibernate Validator



# Validation Annotations List

Java EE – REST API

1. `@NotNull`: Validates that the annotated element is not null.
2. `@Size`: Validates that the size of the annotated element is within specified boundaries (min and max).
3. `@Min`: Validates that the annotated element is a number whose value is greater than or equal to the specified minimum.
4. `@Max`: Validates that the annotated element is a number whose value is less than or equal to the specified maximum.
5. `@DecimalMin`: Validates that the annotated element is a number whose value is greater than or equal to the specified minimum (for `BigDecimal` and `BigInteger` types).
6. `@DecimalMax`: Validates that the annotated element is a number whose value is less than or equal to the specified maximum (for `BigDecimal` and `BigInteger` types).
7. `@Digits`: Validates that the annotated element is a number whose value is within the specified range of integral and fractional digits.
8. `@Past`: Validates that the annotated date is in the past.
9. `@Future`: Validates that the annotated date is in the future.
10. `@Pattern`: Validates that the annotated string matches the specified regular expression pattern.
11. `@Email`: Validates that the annotated string is a valid email address.
12. `@NotBlank`: Validates that the annotated string is not null, not empty, and contains at least one non-whitespace character.
13. `@NotEmpty`: Validates that the annotated element is not null and not empty.
14. `@Positive`: Validates that the annotated number is positive.
15. `@PositiveOrZero`: Validates that the annotated number is positive or zero.
16. `@Negative`: Validates that the annotated number is negative.
17. `@NegativeOrZero`: Validates that the annotated number is negative or zero.
18. `@CreditCardNumber`: Validates that the annotated string is a valid credit card number.
19. `@URL`: Validates that the annotated string is a valid URL.

- Ο Hibernate Validator παρέχει annotations για bean validation
- Μπορεί να χρησιμοποιηθεί για syntax validation
- Για λογικά validations (π.χ. δεν υπάρχει ίδιο email στη ΒΔ) θα πρέπει να έχουμε custom validation



# JPA / Hibernate Validator

Java EE – REST API

```
1 package gr.aueb.cf.jax.rest;
2
3 import ...
4
12
13 @Path("/users")
14 public class UserRestController {
15
16     private final ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
17     private final Validator validator = factory.getValidator();
18 }
```

- Δημιουργούμε ένα validator factory instance με το Validation και στη συνέχεια ένα validator



# Insert user with JSON data

- Στις Web API εφαρμογές όπου ο Server κάνει expose REST endpoints, ο client στέλνει JSON ή XML data
- Ο Server λαμβάνει τα JSON (ή XML) data και πρέπει να τα κάνει parse με μία βιβλιοθήκη όπως Jackson (ή JSON-B) ώστε να τα κάνει map σε κάποιο DTO. Το mapping γίνεται αυτόματα αρκεί τα πεδία του DTO να είναι τα ίδια με τα keys του JSON
- Η JSON-B είναι της Jakarta και είναι πολύ απλή βιβλιοθήκη. Η Jackson έχει περισσότερες δυνατότητες
- Στη συνέχεια και αφού γίνει το insert στη ΒΔ, επιστρέφεται στον client ως Http code το 201 (Created)



# Insert user (1)

```
19 @Path("/users")
20 public class UserRestController {
21
22     private final ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
23     private final Validator validator = factory.getValidator();
24
25     @POST
26     @Path("")
27     @Consumes(MediaType.APPLICATION_JSON)
28     @Produces(MediaType.APPLICATION_JSON)
29     public Response insertUser(UserInsertDTO userInsertDTO, @Context UriInfo uriInfo) {
30         List<String> errors = new ArrayList<>();
31         Set<ConstraintViolation<UserInsertDTO>> violations = validator.validate(userInsertDTO);
32         if (!violations.isEmpty()) {
33             for (ConstraintViolation<UserInsertDTO> violation : violations) {
34                 errors.add(violation.getMessage());
35             }
36             return Response.status(Response.Status.BAD_REQUEST).entity(errors).build();
37         }
38         Long newlyCreatedId = 2L;
39         URI uri = uriInfo.getAbsolutePathBuilder().path(newlyCreatedId.toString()).build();
40         return Response.status(Response.Status.CREATED).location(uri).build();
41     }
42 }
```

- Συνολικά, αριστερά είναι το process για το insert του user
- Το path του Controller είναι /users





# @POST

```
25 @POST
26 @Path("/")
27 @Consumes(MediaType.APPLICATION_JSON)
28 @Produces(MediaType.APPLICATION_JSON)
29 public Response insertUser(UserInsertDTO userInsertDTO, @Context UriInfo uriInfo) {
```

- @POST. Με post κάνουμε insert. Τα semantics της insert στην αρχιτεκτονική REST είναι με POST. Η POST δεν είναι idempotent (αν κάνουμε πολλαπλά POST θα γίνουν πολλαπλά insert)
- @Path. Είναι κενό, ώστε το τελικό path να είναι /teachers. Στο REST το URI Design ορίζει αυτή τη μορφή path για το POST
- @Consumes. Ορίζει τη μορφή του payload των εισερχόμενων data. Αν είναι JSON, τότε γίνεται αυτόματα –με Jackson– το mapping στο DTO
- @Context. Με @Context κάνουμε inject διάφορα στοιχεία του context όπως το context του Request, UriInfo, HttpHeaders, SecurityContext κλπ



```
25 @POST
26 @Path("")
27 @Consumes(MediaType.APPLICATION_JSON)
28 @Produces(MediaType.APPLICATION_JSON)
29 public Response insertUser(UserInsertDTO userInsertDTO, @Context UriInfo uriInfo) {
```

- Παρέχει πρόσβαση σε πληροφορίες σχετικά με το URI του request, όπως το absolute path, τα query params, το base URI, κλπ. και είναι χρήσιμο για την κατασκευή URI για νέα resources ή για την πρόσβαση σε πληροφορίες σχετικά με το τρέχον request URI



# POSTMAN

## Java EE – REST API

HTTP JAX6-Test / add a user Save Share

**POST** ▼ `{{base_url}}/api/users` Send ▼

Params Authorization Headers (9) **Body** ● Scripts Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {  
2   "username": "thanasis@aueb.gr",  
3   "password": "Ath@naSi0s!"  
4 }
```

Body Cookies Headers (5) Test Results 201 Created • 1075 ms • 176 B • ...

Key	Value
Location	<a href="http://localhost:8080/cf/api/users/2">http://localhost:8080/cf/api/users/2</a>
Content-Length	0
Date	Sat, 21 Sep 2024 07:28:46 GMT
Keep-Alive	timeout=20
Connection	keep-alive



# Negative Scenario

The screenshot shows a REST client interface with the following elements highlighted by red circles:

- POST** method selected in the top left.
- Body** tab selected in the top middle.
- raw** radio button selected for the body type.
- JSON** dropdown menu selected for the body format.
- The request body content: 

```
1 {  
2   "username": "thanos@aueb.gr",  
3   "password": "12345"  
4 }
```
- 400 Bad Request** status message in the bottom right.
- The response body content: 

```
1 [  
2   "Invalid password"  
3 ]
```



# Teacher Insert DTO - Άσκηση

Java EE – REST API

```
1 package gr.aueb.cf.jax.dto;
2
3 import ...
4
10
11 @NoArgsConstructor
12 @AllArgsConstructor
13 @Getter
14 @Setter
15 public class TeacherInsertDTO {
16
17     @Size(min = 6, max = 6, message = "Ssn must be 6-digit long")
18     private String ssn;
19
20     @NotBlank(message = "Please fill the firstname")
21     private String firstname;
22
23     @NotBlank(message = "Please fill the lastname")
24     private String lastname;
25 }
```

- Έστω το Teacher Insert DTO
- Δημιουργήστε ένα POST endpoint για την εισαγωγή καθηγητή
- Δημιουργήστε ένα TeacherReadOnlyDTO
- Δημιουργήστε GET endpoints για `getOneTeacher` και `getAllTeachers`