



S|CASE

Generating Rapidly RESTful Services with S-CASE

Christoforos Zolotas

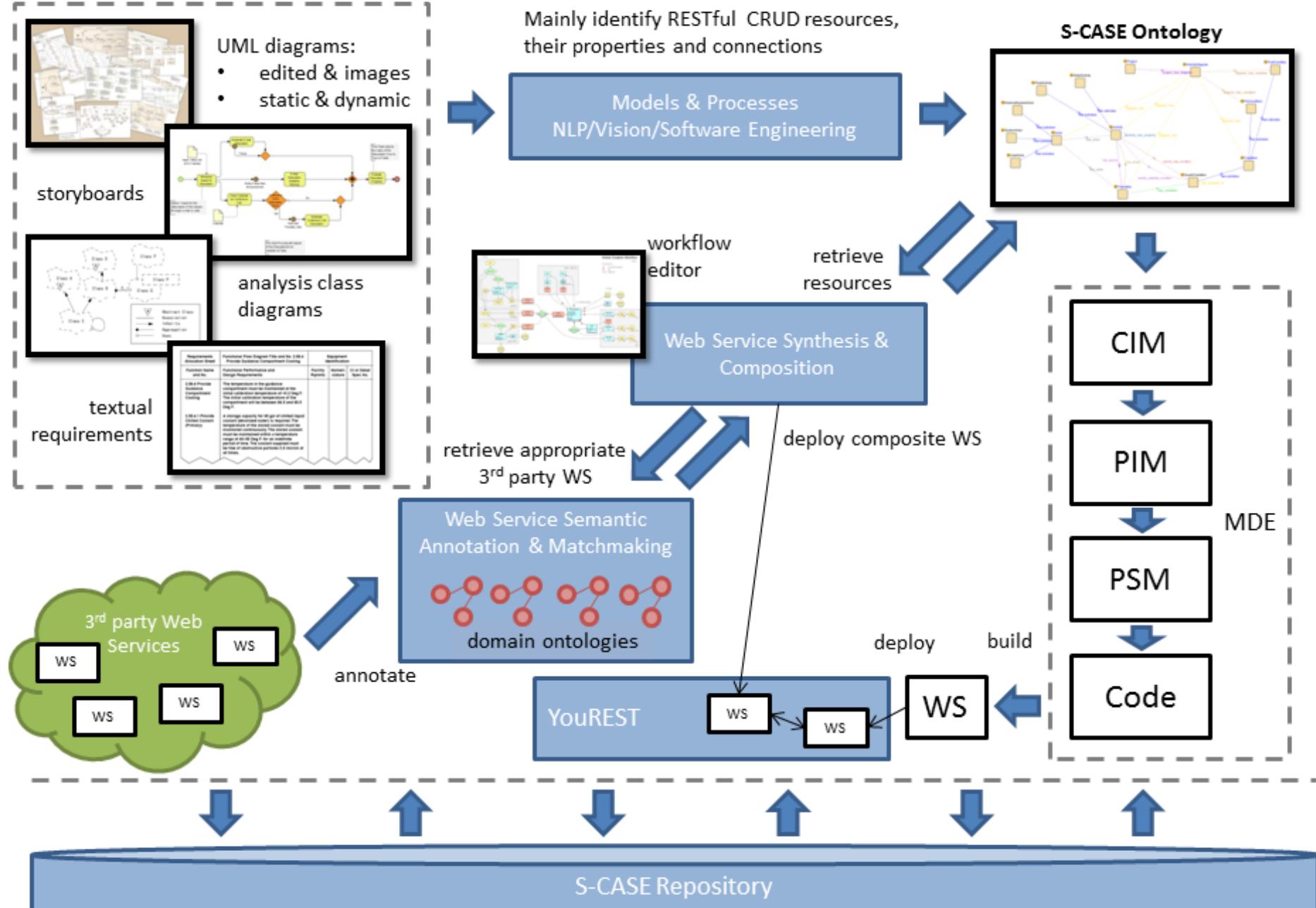


Overview

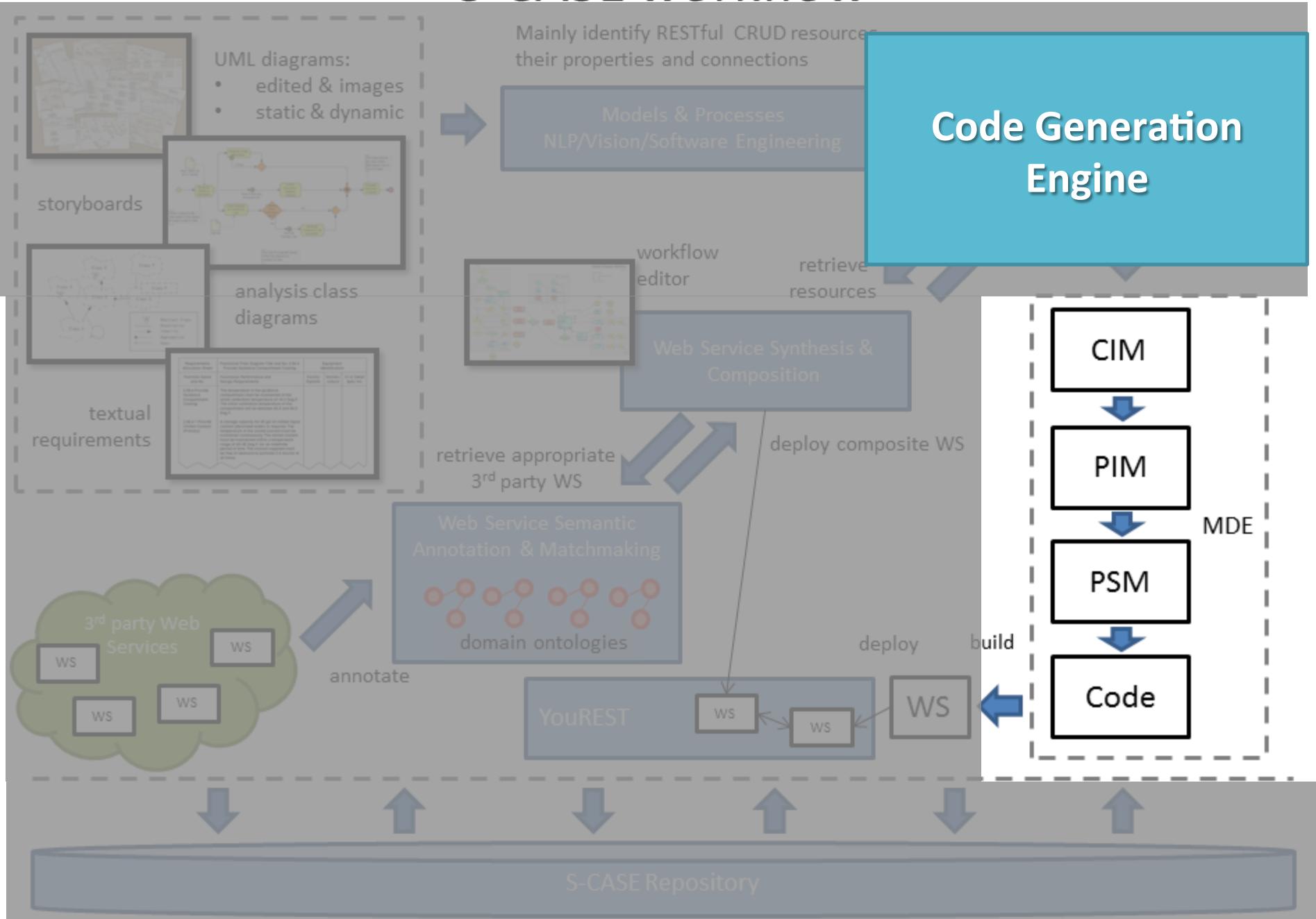
- S-CASE: Overview
- The Coffee Machine Paradigm
- S-CASE: A two-fold Mechanism
- Code Generation Engine Capabilities Overview
- REST & Richardson's Maturity Model
- REST Wizard
- Basic Authentication Wizard
- Keyword Searching Wizard
- 3rd Party Services Interoperation Wizard
- Relating S-CASE Upper CASE tools with Code Generation Engine



S-CASE workflow



S-CASE workflow



The Coffee Machine Paradigm

Easy handling, (almost) ready to use output



The Coffee Machine Paradigm

Easy handling, (almost) ready to use output

1) The user provides the desired input to the machine using its interface.

It is supposed to be easy to use for domain experts (e.g. someone who has used several coffee machines)



The Coffee Machine Paradigm

Easy handling, (almost) ready to use output



2) The machine makes any needed calculations and produces the outcome.

The user is **not** supposed to understand how the machine works.



The Coffee Machine Paradigm

Easy handling, (almost) ready to use output

- 3) The user gets the output. Some post-actions might be required e.g. add some sugar.



S-CASE: A two-fold Mechanism

S-CASE Framework

**S-CASE Code Generation Engine
(lower CASE)**



S-CASE: A two-fold Mechanism

S-CASE Framework

S-CASE Developer Tools (upper CASE)



S-CASE Code Generation Engine (lower CASE)



S-CASE: A two-fold Mechanism

S-CASE Framework

S-CASE Developer



Multi-modal
requirements



S-CASE Developer Tools



S-CASE Code Generation Engine



Code Generation Engine Automation Capabilities

1) RESTful data API + database:



2) Basic Authentication with
username and password:

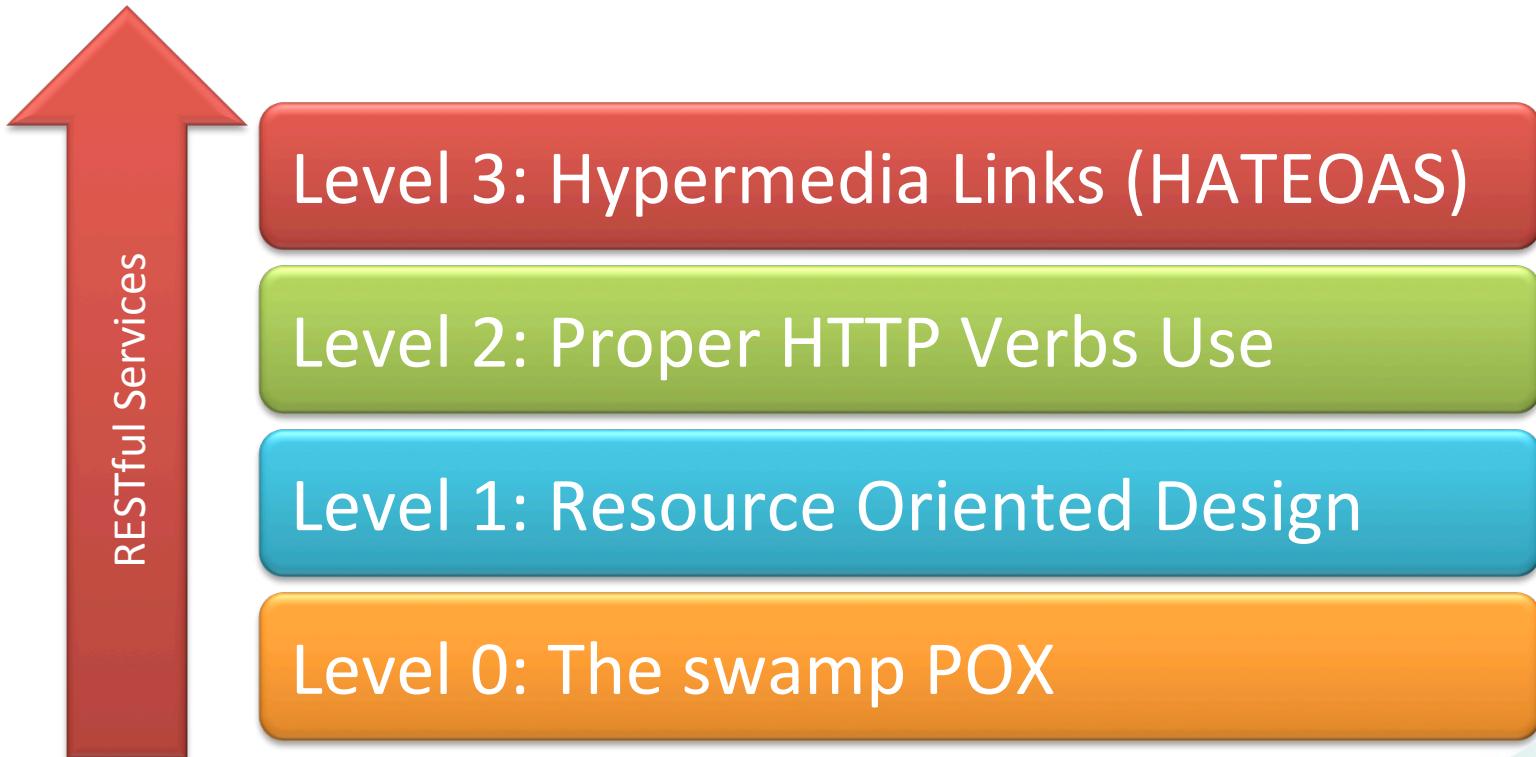


3) Database Keyword Searching:



4) Integrate existing services from the Web

REST Definition & Richardson's Maturity Model



REST Definitions & HATEOAS

With every request the server sends back to the client the next possible actions:



REST in S-CASE

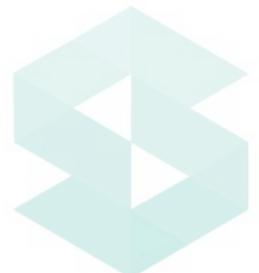
CRUD Resources and More

CRUD Resources:

1. They model the service's data
2. They are capable of the 4 primitive CRUD actions:
 - CREATE (POST)
 - READ (GET)
 - UPDATE (PUT)
 - DELETE (DELETE)
3. They come always with Hypermedia Links.

Algorithmic Resources:

1. They embed some manual algorithm in order to perform some developer defined **non** primitive action.
2. They come with a RESTful placeholder/wrapper in which the developer fills in manual code
3. They can be search resources or interoperate with existing services in the Web. (More on this later)



Code Generation Engine Preferences



Eclipse Preferences -> S-CASE -> Code Generation

Properties for testproject

Type filter text

- > Resource
- > Builders
- > OCL
- > Papyrus
- Project Facets
- Project References
- Run/Debug Settings
- S-CASE**
 - Code generation**
 - Project domain
 - Project folders
- > Task Repository
- Task Tags
- > Validation
- WikiText

Code generation

Use project-specific settings [Configure Workspace Settings...](#)

Web service name: SampleService

Use project name+ 'Api' for service name

Use project's output folder

Output path: C:/MarsWorkspace2 [Browse...](#)

Import generated project to workspace:

Database type: PostgreSQL server

Database server address: localhost

Database server port: 5432

Database user name: postgres

Database password: fp7s-case

Add Basic authentication

Add ABAC authentication

Add database searching

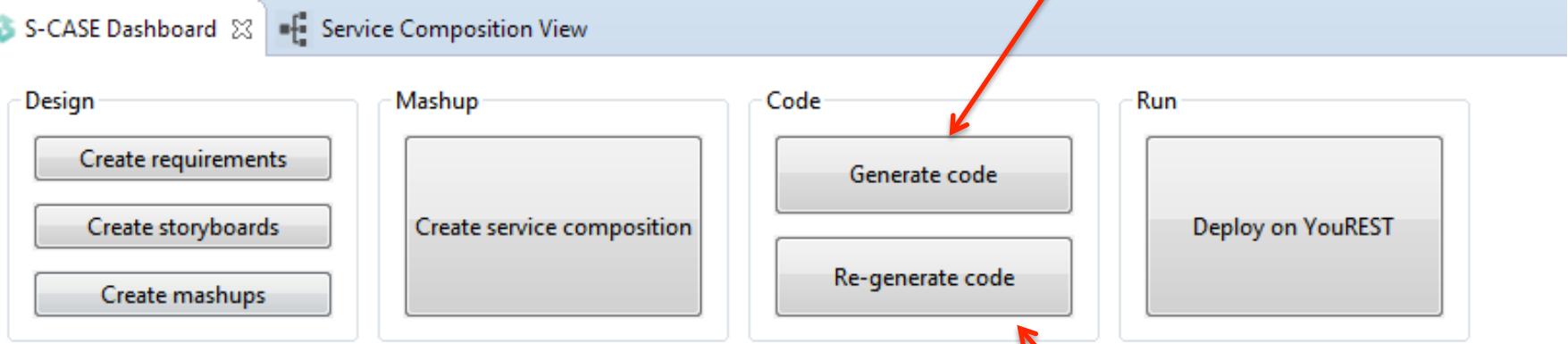
Add External compositions

[Restore Defaults](#) [Apply](#)

[OK](#) [Cancel](#)



S-CASE Dashboard



S-CASE Dashboard Service Composition View

Design

- Create requirements
- Create storyboards
- Create mashups

Mashup

- Create service composition

Code

- Generate code
- Re-generate code

Run

- Deploy on YouREST

Starts the Code Generation Engine

Starts the Code Generation Engine with
the last saved state reloaded.



REST Wizard

User Interface and its Components

RESTMarksTestApi CIM Editor

CRUD resources must have at least one CRUD Activity. CRUD resource tagSearch does not have any.

RESTMarksTestApi resources:

- account
- bookmark
- tag
- tagSearch
- emailBookmark

Algorithmic

Create **Rename** **Delete**

Selected resource properties:

- password
- username**
- email

Create **Rename** **Delete**

CRUD Activities:

- CREATE:
- READ:
- UPDATE:
- DELETE:

Input media format:

- Application/JSON
- Application/XML

Output media format:

- Application/JSON
- Application/XML

Property Configuration:

Collection **Type:**

- String
- Integer
- Double
- Float
- Long
- Boolean
- Date

naming property

Selected resource's unrelated resources:

- account
- bookmark
- tag
- tagSearch

Add relation **Delete relation**

Selected resource's related resources:

? **Cancel** **Finish**

REST Wizard

User Interface and its Components

Model Resources

RESTMarksTestApi CIM Editor

CRUD resources must have at least one CRUD Activity. CRUD resource tagSearch does not have any.

RESTMarksTestApi resources:

- account
- bookmark
- tag
- tagSearch
- emailBookmark

Algorithmic

Create Rename Delete

CRUD Activities:

CREATE:
 READ:
 UPDATE:
 DELETE:

Input media format:

Application/JSON
 Application/XML

Output media format:

Application/JSON
 Application/XML

Selected resource properties:

- password
- username**
- email

Create Rename Delete

Property Configuration:

Collection Type:
 naming property String
 Integer
 Double
 Float
 Long
 Boolean
 Date

Selected resource's unrelated resources:

- account
- bookmark
- tag
- tagSearch

Add relation Delete relation

Selected resource's related resources:

?

Cancel Finish

REST Wizard

User Interface and its Components

RESTMarksTestApi CIM Editor

CRUD resources must have at least one CRUD Activity. CRUD resource tagSearch does not have any.

RESTMarksTestApi resources:

- account
- bookmark
- tag
- tagSearch
- emailBookmark

Algorithmic

Create Rename Delete

CRUD Activities:

CREATE:
 READ:
 UPDATE:
 DELETE:

Input media format:

Application/JSON
 Application/XML

Output media format:

Application/JSON
 Application/XML

Selected resource properties:

- password
- username**
- email

Create Rename Delete

Property Configuration:

Collection Type:
 naming property String
 Integer
 Double
 Float
 Long
 Boolean
 Date

Selected resource's unrelated resources:

- account
- bookmark
- tag
- tagSearch

Add relation Delete relation

Selected resource's related resources:

Cancel Finish

REST Wizard

User Interface and its Components

RESTMarksTestApi CIM Editor

CRUD resources must have at least one CRUD Activity. CRUD resource tagSearch does not have any.

RESTMarksTestApi resources:

- account
- bookmark
- tag
- tagSearch
- emailBookmark

Algorithmic

Create Rename Delete

CRUD Activities:

CREATE:
 READ:
 UPDATE:
 DELETE:

Input media format:

Application/JSON
 Application/XML

Output media format:

Application/JSON
 Application/XML

Selected resource properties:

- password
- username**
- email

Create Rename Delete

Property Configuration:

Collection Type:
 naming property

- String
- Integer
- Double
- Float
- Long
- Boolean
- Date

Selected resource's unrelated resources:

- account
- bookmark
- tag
- tagSearch

Add relation Delete relation

Selected resource's related resources:

Cancel Finish

?



REST Wizard

User Interface and its Components

RESTMarksTestApi CIM Editor

CRUD resources must have at least one CRUD Activity. CRUD resource tagSearch does not have any.

RESTMarksTestApi resources:

- account
- bookmark
- tag
- tagSearch
- emailBookmark

Algorithmic

Create **Rename** **Delete**

Selected resource properties:

- password
- username**
- email

Create **Rename** **Delete**

CRUD Activities:

CREATE:
 READ:
 UPDATE:
 DELETE:

Input media format:

Application/JSON
 Application/XML

Output media format:

Application/JSON
 Application/XML

Property Configuration:

Collection Type:
 naming property String
Integer
Double
Float
Long
Boolean
Date

Selected resource's unrelated resources:

- account
- bookmark
- tag
- tagSearch

Add relation Delete relation

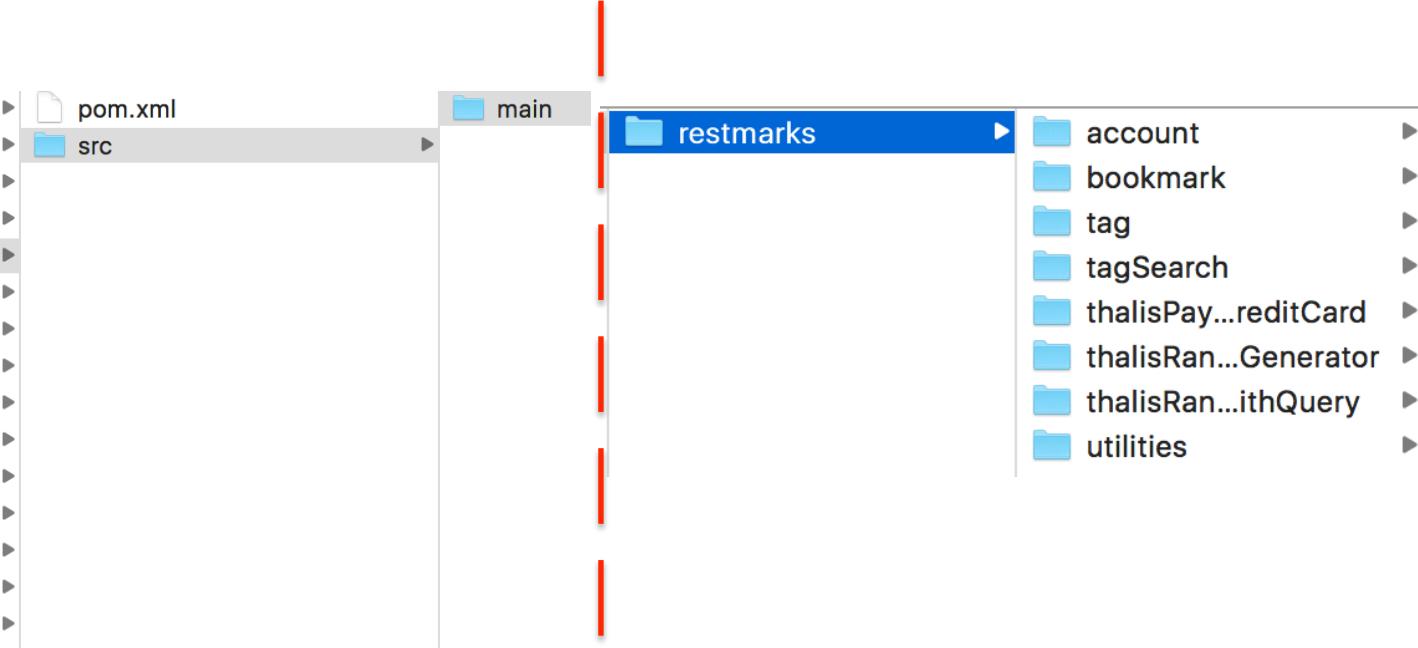
Selected resource's related resources:

?

Cancel **Finish**

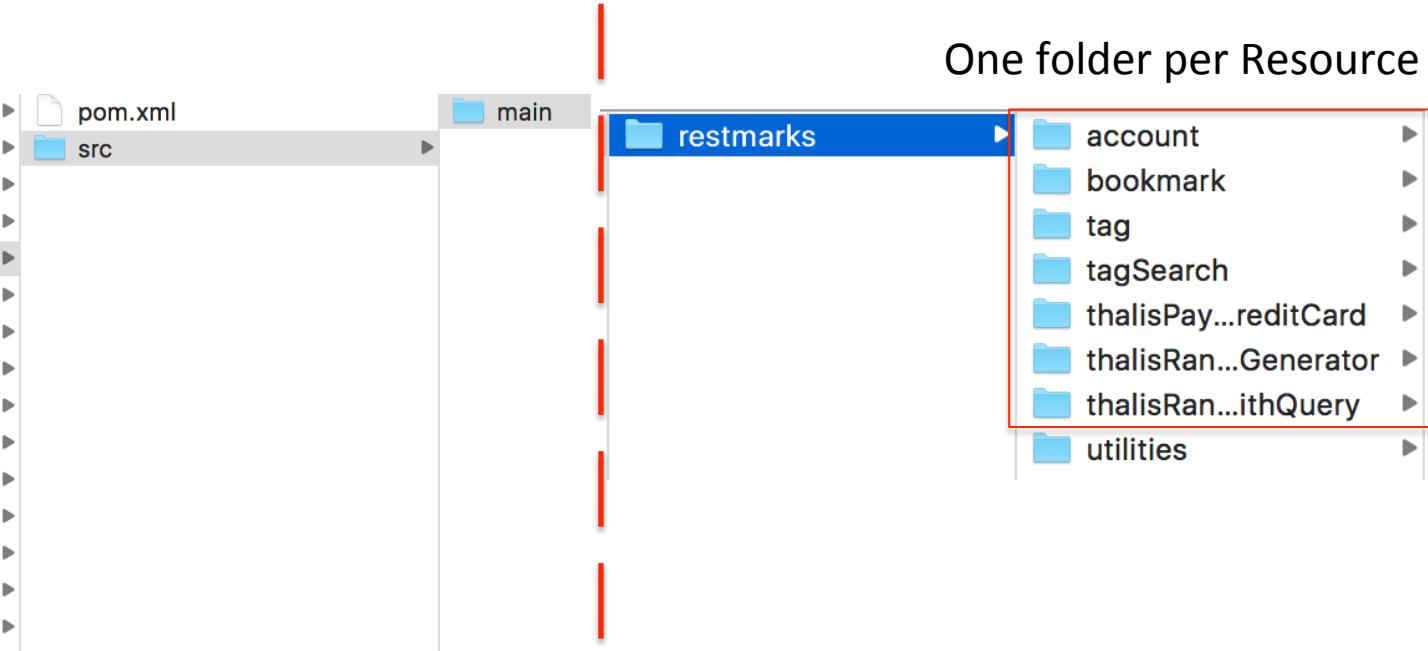
Code Generation Output Structure

- eBucks
- GiftCase
- GiftCaseApi
- MDETestProject
- RESTMarks **(selected)**
- RESTMarksAuth
- RESTMarksEmpty
- RESTMarksFull
- RESTMarksSimple
- SimpleService
- TestMarks
- TestMDEService
- testWapoService
- wapoAdminToolApi
- WSAT



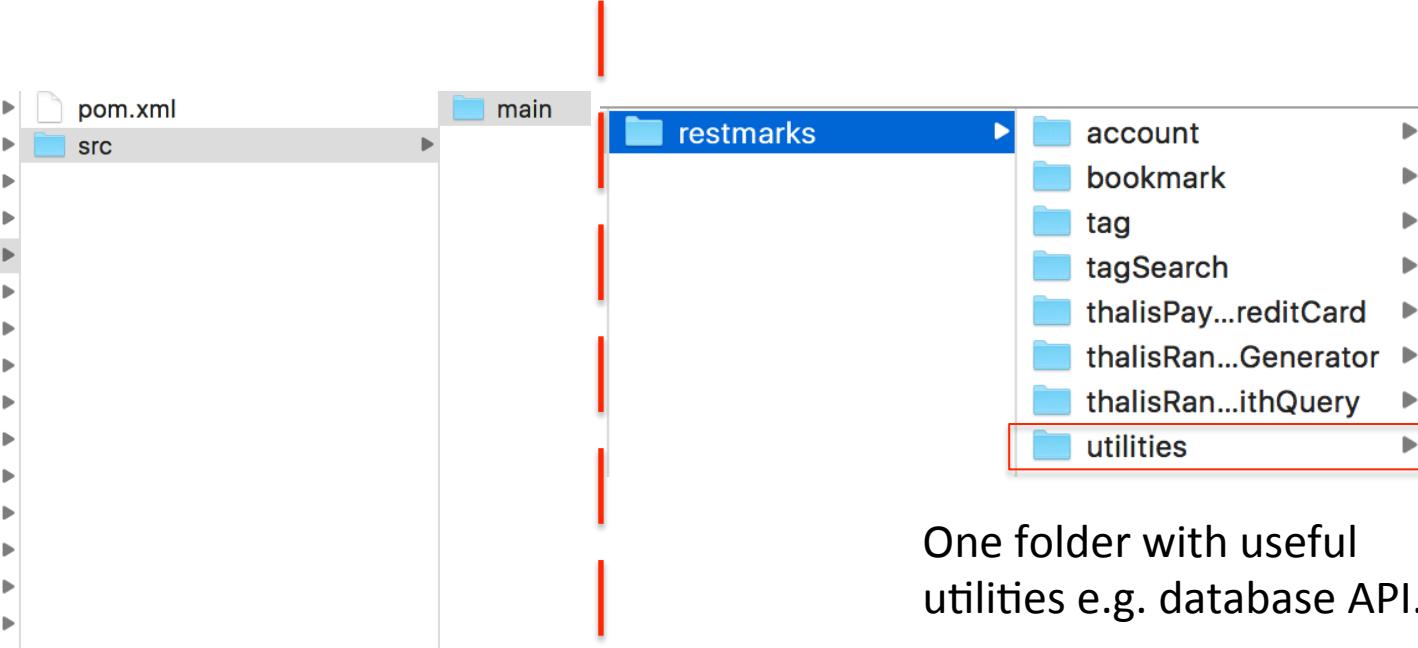
Code Generation Output Structure

- eBucks
- GiftCase
- GiftCaseApi
- MDETestProject
- RESTMarks ■
- RESTMarksAuth
- RESTMarksEmpty
- RESTMarksFull
- RESTMarksSimple
- SimpleService
- TestMarks
- TestMDEService
- testWapoService
- wapoAdminToolApi
- WSAT



Code Generation Output Structure

- eBucks
- GiftCase
- GiftCaseApi
- MDETestProject
- RESTMarks **(selected)**
- RESTMarksAuth
- RESTMarksEmpty
- RESTMarksFull
- RESTMarksSimple
- SimpleService
- TestMarks
- TestMDEService
- testWapoService
- wapoAdminToolApi
- WSAT



One folder with useful utilities e.g. database API.



Packaging and Deploying a generated Service

Packaging the Generated Service:

1st Way: Right Click on the imported maven project in your workspace and click Run As -> Maven Install

2nd Way: Navigate with the terminal to the output folder of your service and execute the command “mvn package”.



Deploying:

Copy paste the produced WAR file in the appropriate Jetty/Tomcat folder.

Note: within this Hackathon follow the specific deployment instructions with which you will be provided.



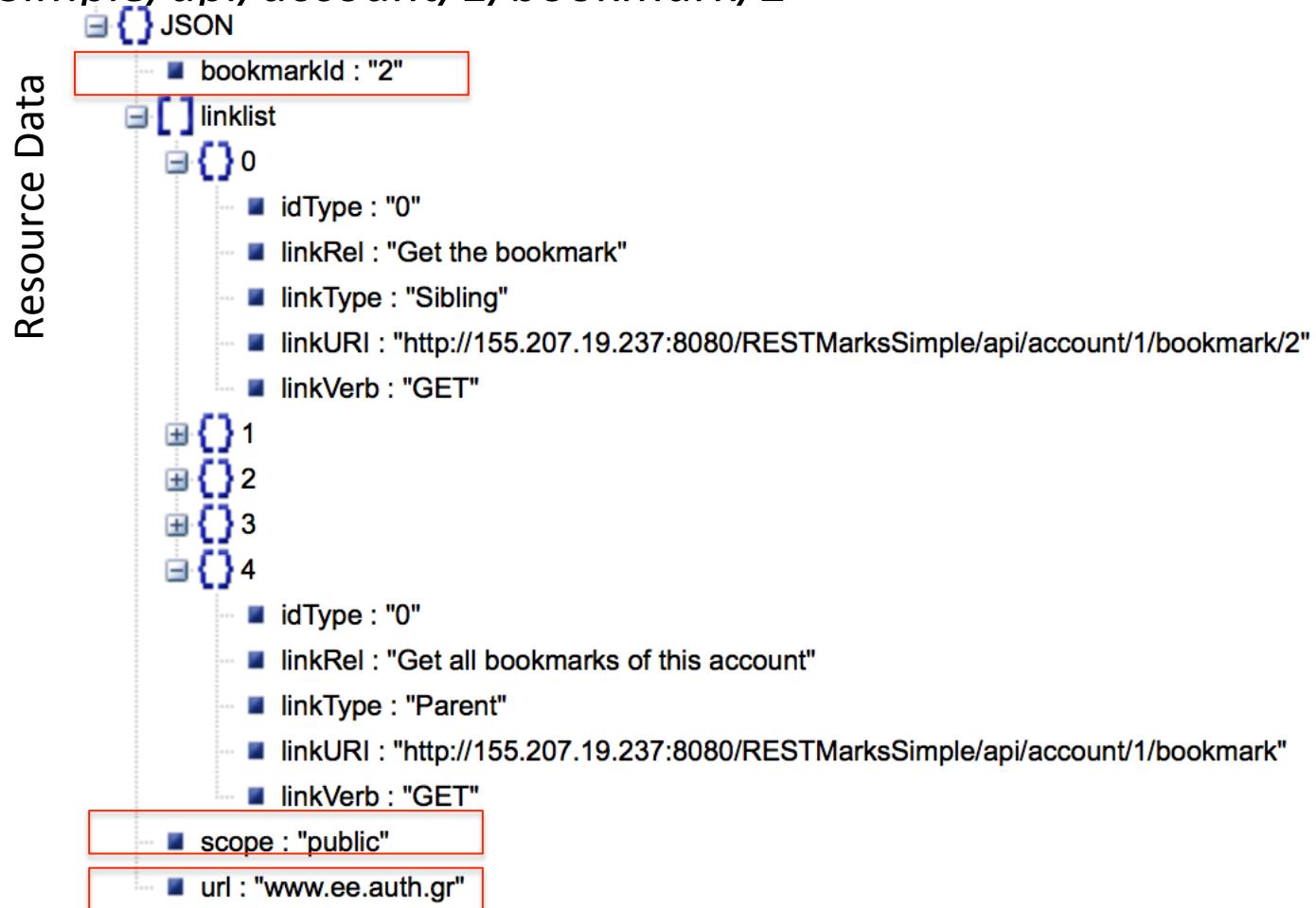
Interacting with the Generated Service

Reading a resource: `curl -i -X GET http://155.207.19.237:8080/RESTMarksSimple/api/account/1/bookmark/2`



Interacting with the Generated Service

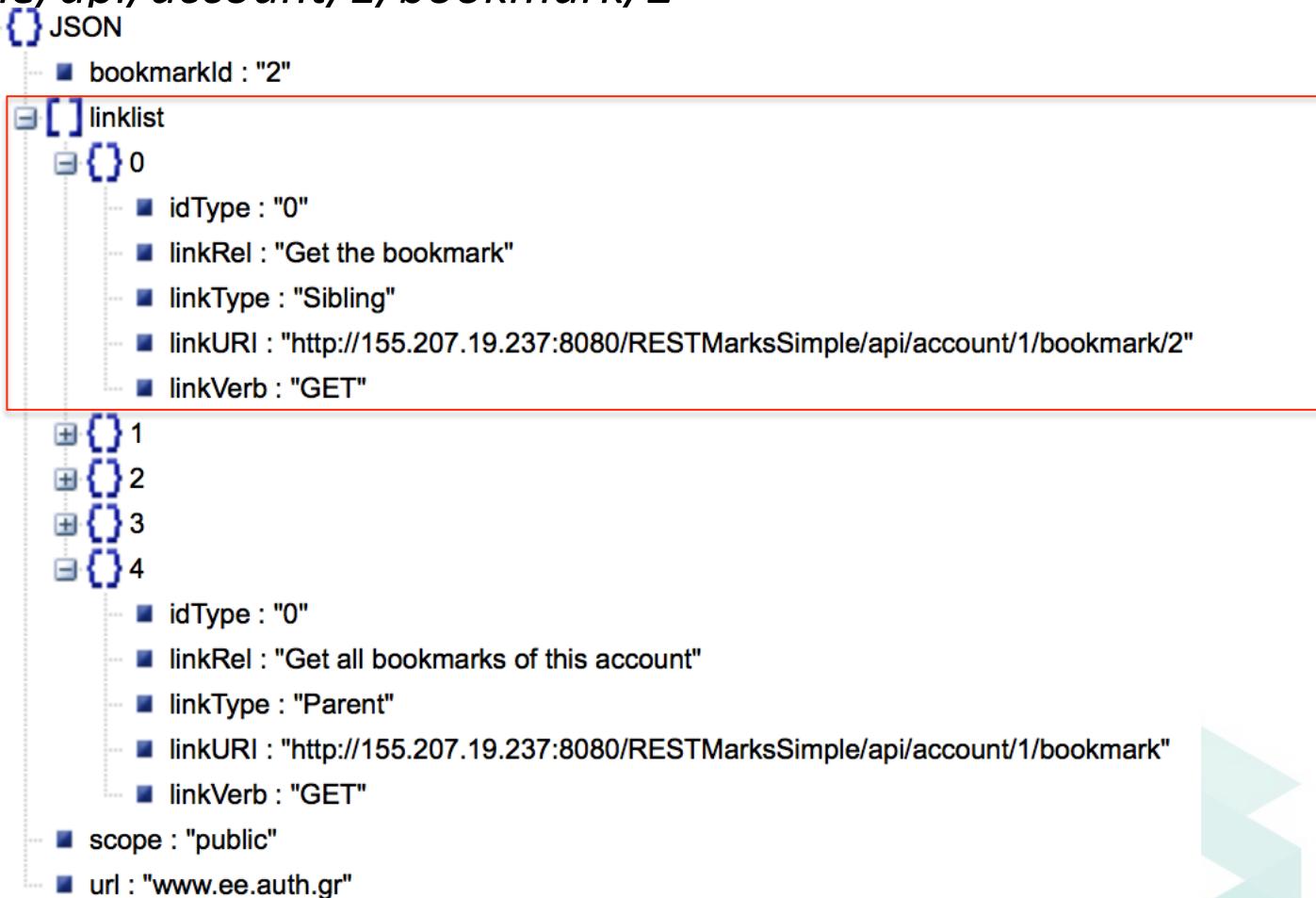
Reading a resource: `curl -i -X GET http://155.207.19.237:8080/RESTMarksSimple/api/account/1/bookmark/2`



Interacting with the Generated Service

Reading a resource: `curl -i -X GET http://155.207.19.237:8080/RESTMarksSimple/api/account/1/bookmark/2`

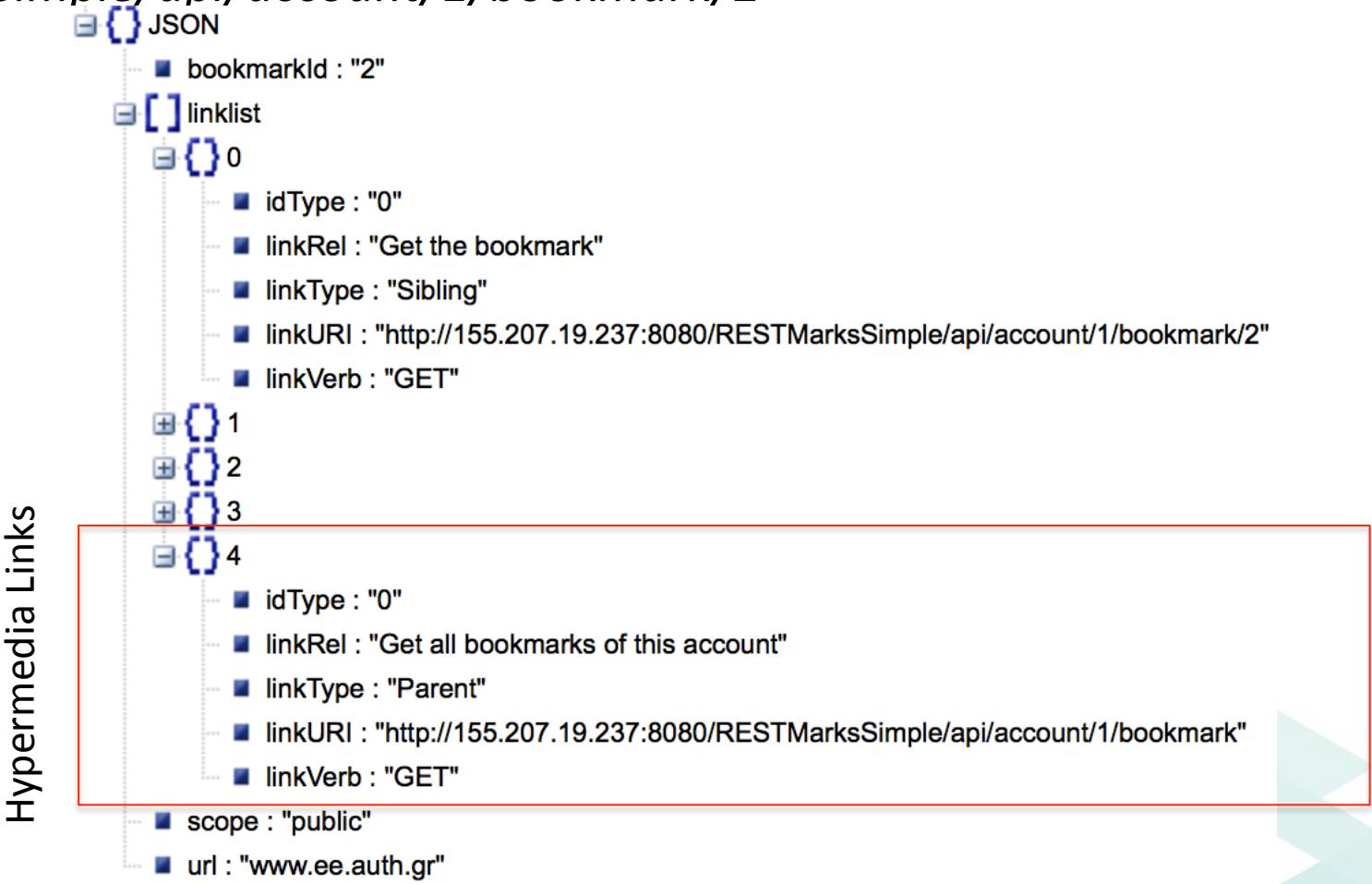
Hypermedia Links



```
JSON
  bookmarkId : "2"
  linklist
    0
      idType : "0"
      linkRel : "Get the bookmark"
      linkType : "Sibling"
      linkURI : "http://155.207.19.237:8080/RESTMarksSimple/api/account/1/bookmark/2"
      linkVerb : "GET"
    1
    2
    3
    4
      idType : "0"
      linkRel : "Get all bookmarks of this account"
      linkType : "Parent"
      linkURI : "http://155.207.19.237:8080/RESTMarksSimple/api/account/1/bookmark"
      linkVerb : "GET"
      scope : "public"
      url : "www.ee.auth.gr"
```

Interacting with the Generated Service

Reading a resource: `curl -i -X GET http://155.207.19.237:8080/RESTMarksSimple/api/account/1/bookmark/2`



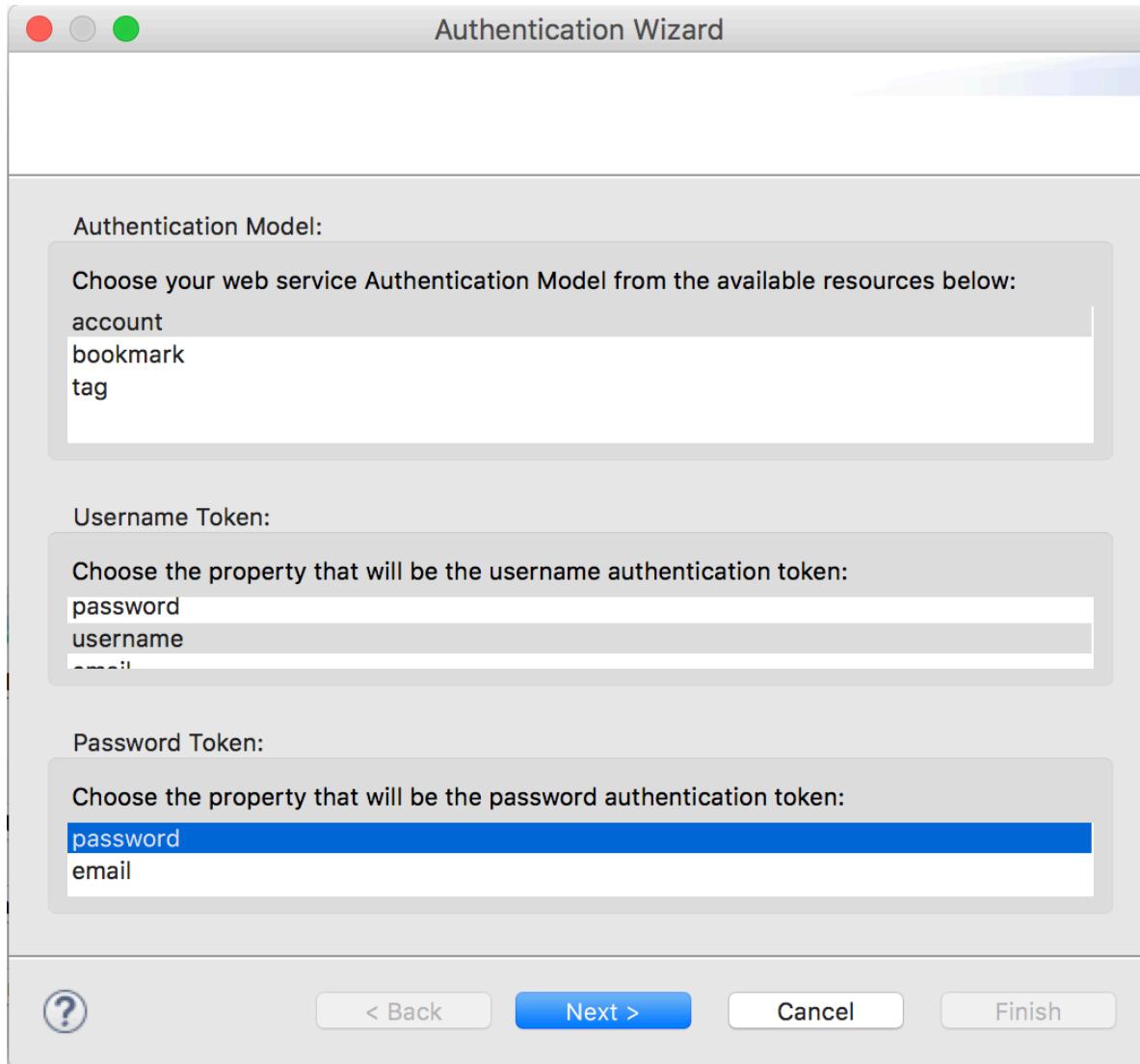
Embedding Basic Authentication

Provided automated functionality :

1. **Username/Password** automated login mechanism
2. Authentication **granularity per CRUD Verb** of each resource.
3. Two access modes: “**Authenticated**” to allow only authenticated users, “**All**” to allow anyone.



Embedding Basic Authentication



Embedding Basic Authentication

S|CASE

Authentication Wizard

Authentication Model:
Choose your web service Authentication Model from the available resources below:
 account
 bookmark
 tag

Username Token:
Choose the property that will be the username authentication token:
 password
 username

Password Token:
Choose the property that will be the password authentication token:
 password
 email

[?](#) < Back [Next >](#) Cancel Finish

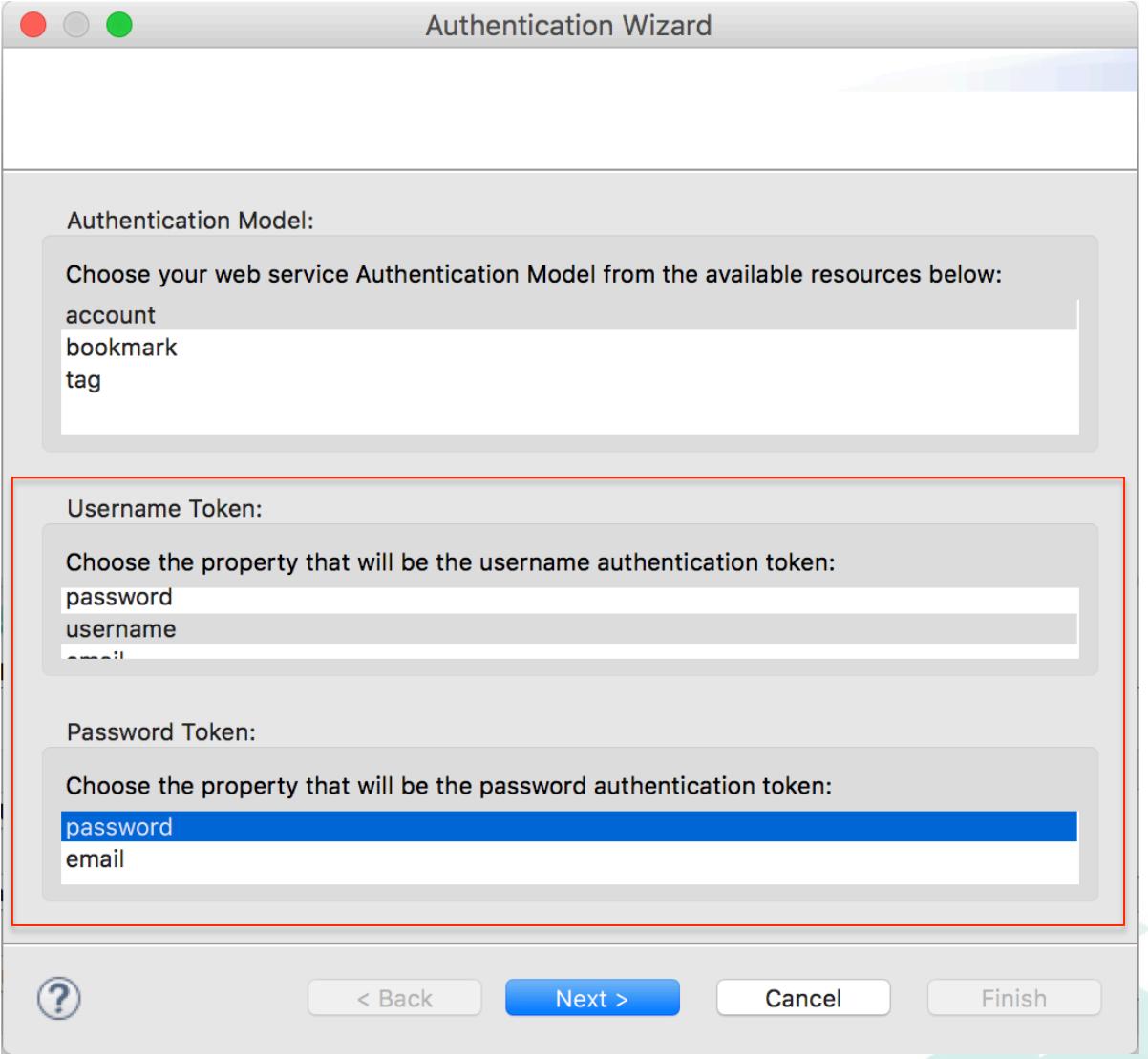
- 1) Select a CRUD resource as authentication model.



Embedding Basic Authentication



2) Select two **String properties** of this resource as **username** and **password** authentication tokens.



The screenshot shows the 'Authentication Wizard' dialog box. In the 'Authentication Model' section, 'account' is selected. In the 'Username Token' section, 'username' is selected. In the 'Password Token' section, 'password' is selected. The 'password' selection in the 'Password Token' section is highlighted with a red border.

Authentication Model:
Choose your web service Authentication Model from the available resources below:
account
bookmark
tag

Username Token:
Choose the property that will be the username authentication token:
password
username
email

Password Token:
Choose the property that will be the password authentication token:
password
email

< Back Next > Cancel Finish

Embedding Basic Authentication



3) Select the **authentication mode** for every **CRUD Activity** of each resource.

Authentication Wizard

Web Service Resources:
 Pick any resource from the list below to apply change the default authentication mode:
 account
 bookmark
 tag
 tagSearch

Select the Authentication Mode for each of the following selected resource's CRUD activities:

Create Activity: All Authenticated

Read Activity: All Authenticated

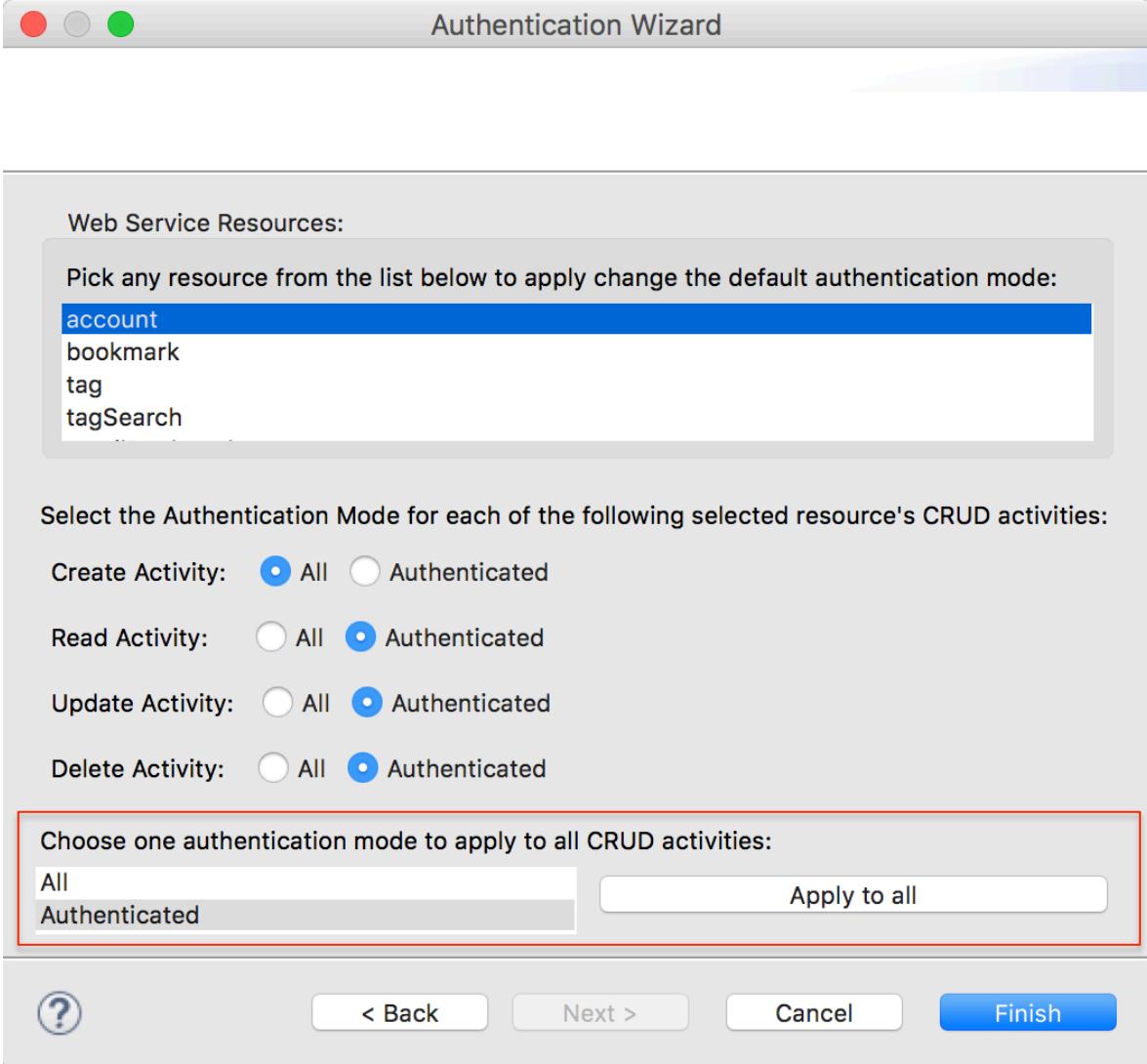
Update Activity: All Authenticated

Delete Activity: All Authenticated

Choose one authentication mode to apply to all CRUD activities:
 All
 Authenticated Apply to all

[?](#) < Back Next > Cancel **Finish**

Embedding Basic Authentication



The screenshot shows a software window titled "Authentication Wizard". At the top, there are three colored circles (red, grey, green) and the title "Authentication Wizard". Below the title, the window is divided into several sections:

- Web Service Resources:** A list of resources: account, bookmark, tag, tagSearch. The "account" item is highlighted with a blue background.
- Select the Authentication Mode for each of the following selected resource's CRUD activities:**
 - Create Activity: All Authenticated
 - Read Activity: All Authenticated
 - Update Activity: All Authenticated
 - Delete Activity: All Authenticated
- Choose one authentication mode to apply to all CRUD activities:** A dropdown menu with "All" and "Authenticated" options, and a button "Apply to all" to its right. This section is highlighted with a red border.
- At the bottom, there are navigation buttons: a question mark icon, "< Back" and "Next >", "Cancel", and "Finish".



Optionally use the **one click** authentication setup.

Embedding Keyword Searching

Using the **Keyword Searching** capabilities of the Code Generation Engine:

1. An **algorithmic resource** can be marked as a **Search Resource**, namely one that performs the search activity.
2. A **CRUD resource and some of its properties** can be marked as **Searchable**, namely they will be indexed and searched by a Search Resource.



Embedding Keyword Searching

Search DB Wizard

Add any of the available resources to the search resource list:

Available Resources: emailBookmark

Selected Search Resources: tagSearch

Add Remove

Add all the properties that must be searchable by the selected search resource

Select a Resource: account bookmark tag

Select a Property: keyword

Selected Properties: bookmark:url tag:keyword

Add Remove

?

Cancel Finish



Embedding Keyword Searching

1) Mark an algorithmic resource as **Search Resource**

Search DB Wizard

Add any of the available resources to the search resource list:

Available Resources:

- emailBookmark

Selected Search Resources:

- tagSearch

Add all the properties that must be searchable by the selected search resource

Select a Resorce:

- account
- bookmark
- tag

Select a Property:

- keyword

Selected Properties:

- bookmark:url
- tag:keyword

?

Cancel Finish

Embedding Keyword Searching

Search DB Wizard

Add any of the available resources to the search resource list:

Available Resources:
 emailBookmark

Selected Search Resources:
 tagSearch

Add
Remove

Add all the properties that must be searchable by the selected search resource

Select a Resorce:
 account
 bookmark
 tag

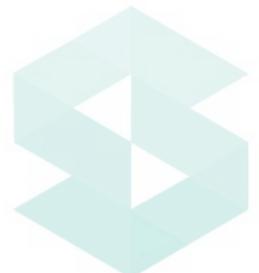
Select a Property:
 keyword

Selected Properties:
 bookmark:url
 tag:keyword

Add
Remove

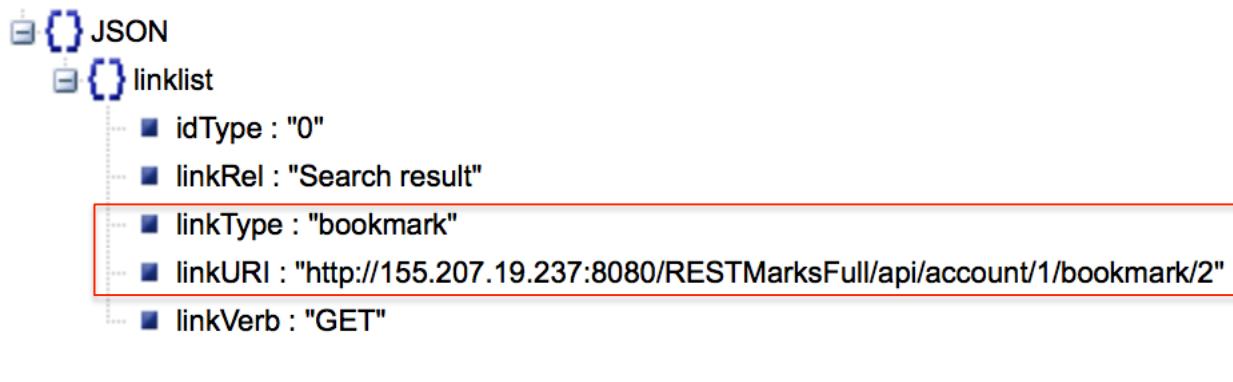
?
Cancel
Finish

2) Mark CRUD Resource and their Properties as Searchable. Any combination is available.

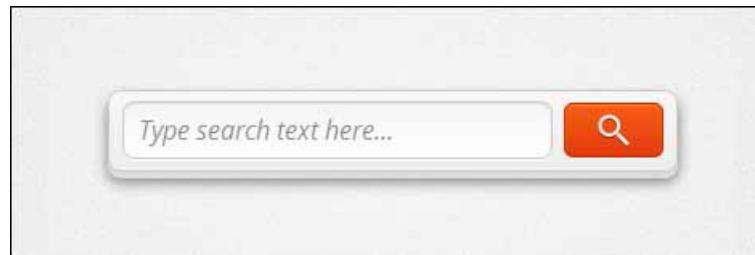


Interacting with a Search Resource

```
curl -i -X GET 'http://155.207.19.237:8080/RESTMarksFull/api/AlgobookmarkSearch?  
searchBookmarkKeyword=true&searchKeyword=best'
```



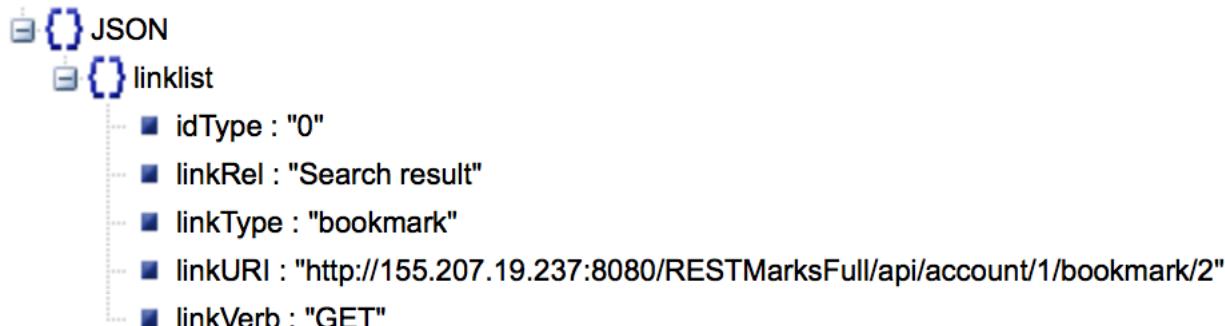
```
public JavaAlgobookmarksearchModel getbookmarkSearch( @DefaultValue("guest") @HeaderParam("authorization") String  
authHeader, @QueryParam("searchKeyword") String searchKeyword, @QueryParam("searchBookmarkKeyword") String  
searchBookmarkKeyword){
```



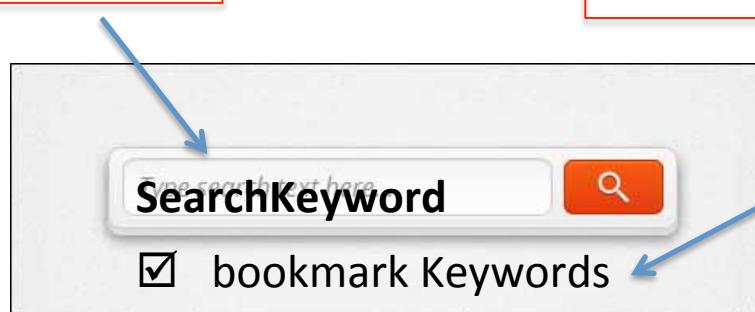
Interacting with a Search Resource

```
curl -i -X GET 'http://155.207.19.237:8080/RESTMarksFull/api/AlgobookmarkSearch?  

searchBookmarkKeyword=true&searchKeyword=best'
```



```
public JavaAlgobookmarksearchModel getbookmarkSearch( @DefaultValue("guest") @HeaderParam("authorization") String authHeader, @QueryParam("searchKeyword") String searchKeyword, @QueryParam("searchBookmarkKeyword") String searchBookmarkKeyword){
```

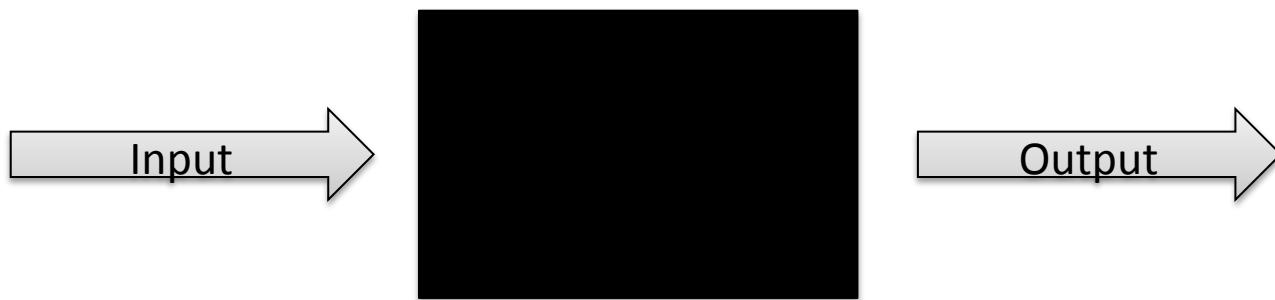


The user interface shows a search bar with placeholder text "Type something here" and a magnifying glass icon. Below the search bar is a checkbox labeled "bookmark Keywords" with a checked status.



Interacting with the Web

The External Composition Wizard provides automatically the capability to interoperate with existing Web Services. To do so the S-CASE developer has to model the input and output data model of it.



3rd Party Service as a black box



Interacting with the Web



External Service Composition Editor

Available Resources:

RESTClient Resources:

emailBookmark

Add Remove

External Composition Setup:

URL: www.someemailservice.com

CRUD Verb:

CREATE READ UPDATE DELETE

Query Parameters:

subject sender recipient

Create Delete Rename

Authorization Token

Authorization Token for URL:

Input Data Model

Input Representation:

Application/JSON Application/XML

Input Properties

Collection

Type:

String Boolean Integer Float

Output Data Model

Output Representation:

Application/JSON Application/XML

Output Properties

emailStatus emailSent

Collection

Type:

Integer Float Double Complex

Persist Output to local database

Type: Auto Existent

Resource:

?

< Back Next > Cancel Finish





Interacting with the Web

- 1) Mark some algorithmic resource as **RESTClient** Resource.

The screenshot shows the 'External Service Composition Editor' interface. At the top, there are three buttons (red, grey, green) and the title 'External Service Composition Editor'. Below the title, there are two panels: 'Available Resources:' (empty) and 'RESTClient Resources:' (containing 'emailBookmark'). A red box highlights this area. Below these panels is the 'External Composition Setup' section. It includes fields for 'URL:' (www.someemailservice.com), 'CRUD Verb:' (CREATE selected), 'Query Parameters:' (subject, sender, recipient), and 'Authorization Token for URL:' (empty). To the right of the URL field are buttons for Create, Delete, Rename, and Authorization Token. In the 'Input Data Model' section, there are checkboxes for 'Input Data Model' and 'Output Data Model'. The 'Output Data Model' checkbox is checked, and its settings are displayed: 'Output Representation:' (Application/JSON selected), 'Output Properties:' (emailStatus, emailSent), 'Type:' (Complex selected), and 'Collection' (unchecked). Below this, there is a checkbox for 'Persist Output to local database' and a 'Resource:' field with 'Type: Auto Existent'. At the bottom of the screen are navigation buttons: '?', '< Back', 'Next >', 'Cancel', and 'Finish'.

Interacting with the Web

External Service Composition Editor

Available Resources:

RESTClient Resources:

- emailBookmark

External Composition Setup:

URL:

CRUD Verb:

- CREATE
- READ
- UPDATE
- DELETE

Query Parameters:

- subject
- sender
- recipient

Create **Delete** **Rename** Authorization Token

Authorization Token for URL:

Input Data Model

Input Representation:

- Application/JSON
- Application/XML

Input Properties

Collection

Type:

- String
- Boolean
- Integer
- Float
- ...

Output Data Model

Output Representation:

- Application/JSON
- Application/XML

Output Properties

Collection

Type:

- Integer
- Float
- Double
- Complex

Persist Output to local database

Type: Existent

Resource:

? < Back Next > Cancel **Finish**

- 2) Provide for each RESTClient resource the connection details:
- URL
 - CRUD Verb
 - Any Query Parameter



Interacting with the Web

The screenshot shows the 'External Service Composition Editor' interface. At the top, there are three buttons (red, grey, green) and the title 'External Service Composition Editor'. Below the title, there are two sections: 'Available Resources:' (empty) and 'RESTClient Resources:' containing 'emailBookmark'. In the center, 'External Composition Setup:' includes fields for 'URL:' (www.someemailservice.com), 'CRUD Verb:' (CREATE selected), 'Query Parameters:' (subject, sender, recipient), and buttons for 'Create', 'Delete', 'Rename', and 'Authorization Token'. Below this, 'Authorization Token for URL:' is shown. A red box highlights the 'Input Data Model' and 'Output Data Model' sections. Under 'Input Data Model', 'Input Representation:' has radio buttons for 'Application/JSON' (selected) and 'Application/XML'. Under 'Output Data Model', 'Output Representation:' has radio buttons for 'Application/JSON' (selected) and 'Application/XML'. To the right of these sections are 'Input Properties' and 'Output Properties' tables, each with 'Create', 'Delete', and 'Rename' buttons. A 'Type:' column lists 'String', 'Boolean', 'Integer', 'Float', 'Complex' (highlighted in blue), 'Integer', 'Float', 'Double', and 'Complex'. At the bottom, there are buttons for '?', '< Back', 'Next >', 'Cancel', and 'Finish'.

- 3) Model the required input and/or output of the external service:
- Specify media representation
 - Properties and their types

Interacting with the Web



External Service Composition Editor

Available Resources:

RESTClient Resources:

emailBookmark

Add Remove

External Composition Setup:

URL: www.someemailservice.com

CRUD Verb:

CREATE READ UPDATE DELETE

Query Parameters:

subject sender recipient

Create Delete Rename Authorization Token

Authorization Token for URL:

Input Data Model

Input Representation:

Application/JSON
 Application/XML

Collection

Type:

String Boolean Integer Float

Output Data Model

Output Representation:

Application/JSON
 Application/XML

Collection

Output Properties:

emailStatus emailSent

Create Delete Rename

Type:

Integer Float Double Complex

Persist Output to local database

Type: Auto Existent

Resource:

? < Back Next > Cancel Finish

Optionally, use an existing CRUD Resource as Output Data Model (if it is compatible).

Modeling Complex Input/Output

- 1) Analyze the input/output structure of the 3rd party service.

```
{"widget": {  
    "debug": "on",  
    "window": {  
        "title": "Sample Konfabulator Widget",  
        "name": "main_window",  
        "width": 500,  
        "height": 500  
    },  
    "image": {  
        "src": "Images/Sun.png",  
        "name": "sun1",  
        "hOffset": 250,  
        "vOffset": 250,  
        "alignment": "center"  
    },  
    "text": {  
        "data": "Click Here",  
        "size": 36,  
        "style": "bold",  
        "name": "text1",  
        "hOffset": 250,  
        "vOffset": 100,  
        "alignment": "center",  
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"  
    }  
}
```



Modeling Complex Input/Output



2) Add any top level property in the input/output properties field in the wizard.

S-CASE

Output Data Model

Output Representation:

Application/JSON
 Application/XML

Output Properties

widget

Collection

Type:

Create
Delete
Rename

Persist Output to local database

Type: Auto Existent

Resource:

Complex

?

< Back

Next >

Cancel

Finish



Modeling Complex Input/Output

3) Identify any embedded property within the top level ones.

```
{"widget": {  
    "debug": "on",  
    "window": {  
        "title": "Sample Konfabulator Widget",  
        "name": "main_window",  
        "width": 500,  
        "height": 500  
    },  
    "image": {  
        "src": "Images/Sun.png",  
        "name": "sun1",  
        "hOffset": 250,  
        "vOffset": 250,  
        "alignment": "center"  
    },  
    "text": {  
        "data": "Click Here",  
        "size": 36,  
        "style": "bold",  
        "name": "text1",  
        "hOffset": 250,  
        "vOffset": 100,  
        "alignment": "center",  
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"  
    }  
}
```



Modeling Complex Input/Output

External Service Composition Editor

Every Complex Type must have at least one property. Complex Type ComplexTypeWidget of RESTclient Resource externalServiceWithComplexOutput

RESTClient Resources:	Properties with complex type:	Available Types:
thalisRandomGenerator thalisRandomGeneratorWithQuery externalServiceWithComplexOutput	Output:widget	ComplexTypeWidget ComplexTypeWindow ComplexTypeImage ComplexTypeText

Defined Complex Data Types:

ComplexTypeWidget ComplexTypeWindow ComplexTypeImage ComplexTypeText	<button>Create</button> <button>Rename</button> <button>Delete</button>
-------------------------------------------------------------------------------	-------------------------------------------------------------------------------

Complex Data Type Properties:

	Available Types:
	String Double Integer Float Boolean

Create Rename Delete Collection

4) Model all the identified complex data types recursively for every RESTClient Resource.



Modeling Complex Input/Output

External Service Composition Editor

Every Complex Type must have at least one property. Complex Type
ComplexTypeWindow of RESTclient Resource externalServiceWithComplexOutput

RESTClient Resources:	Properties with complex type:	Available Types:
thalisRandomGenerator thalisRandomGeneratorWithQuery externalServiceWithComplexOutput	Output:widget	ComplexTypeWidget ComplexTypeWindow ComplexTypeImage ComplexTypeText

Defined Complex Data Types:

- ComplexTypeWidget
- ComplexTypeWindow
- ComplexTypeImage
- ComplexTypeText

Create **Rename** **Delete**

Complex Data Type Properties:

- debug
- window
- image
- text

Available Types:

- Boolean
- ComplexTypeWidget
- ComplexTypeWindow
- ComplexTypeImage
- ComplexTypeText

Create **Rename** **Delete** Collection

5) Add any properties to each complex data type and their data type and its over.

ABAC Authorization

Finally, the Code Generation Engine supports embedding automatically an ABAC authorization mechanism to generated services.

If you are interested contact us after the Hackathon.



Old Database Auto Migration

Do you plan to run your business with a new RESTful service? Get your old data with you only with a few clicks!

If you are interested contact us after the Hackathon.



S-CASE Upper CASE tools

In the context of S-CASE, the Upper CASE tools will help you to:

- Identify Resources, their properties and their relations (**Requirements Annotation Tool/Storyboard Creator**).
- Create **RESTClient** resources to interact with the Web automatically (**Web Service Composition**).

The user guide tutorial for either of them will be presented right after this presentation.



Questions



Thank you!

christopherzolotas@issel.ee.auth.gr

