S|CASE

SEVENTH FRAMEWORK PROGRAMME

S|CASE

Generating Rapidly RESTful Services with S-CASE

Christoforos Zolotas

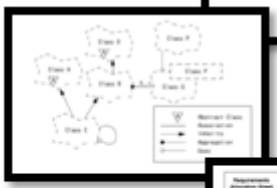
Overview

- 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

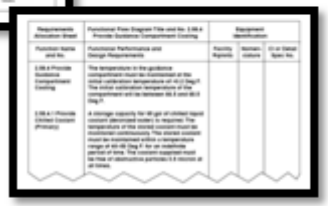
S-CASE workflow

UML diagrams:
- edited & images
- static & dynamic

storyboards



analysis class diagrams



textual requirements

Mainly identify RESTful CRUD resources, their properties and connections

Models & Processes
NLP/Vision/Software Engineering

S-CASE Ontology

workflow editor

retrieve resources

Web Service Synthesis & Composition

retrieve appropriate 3rd party WS

deploy composite WS

CIM

PIM

PSM

Code

MDE

Web Service Semantic Annotation & Matchmaking

domain ontologies

3rd party Web Services

WS

WS

WS

WS

annotate

YouREST

WS

WS

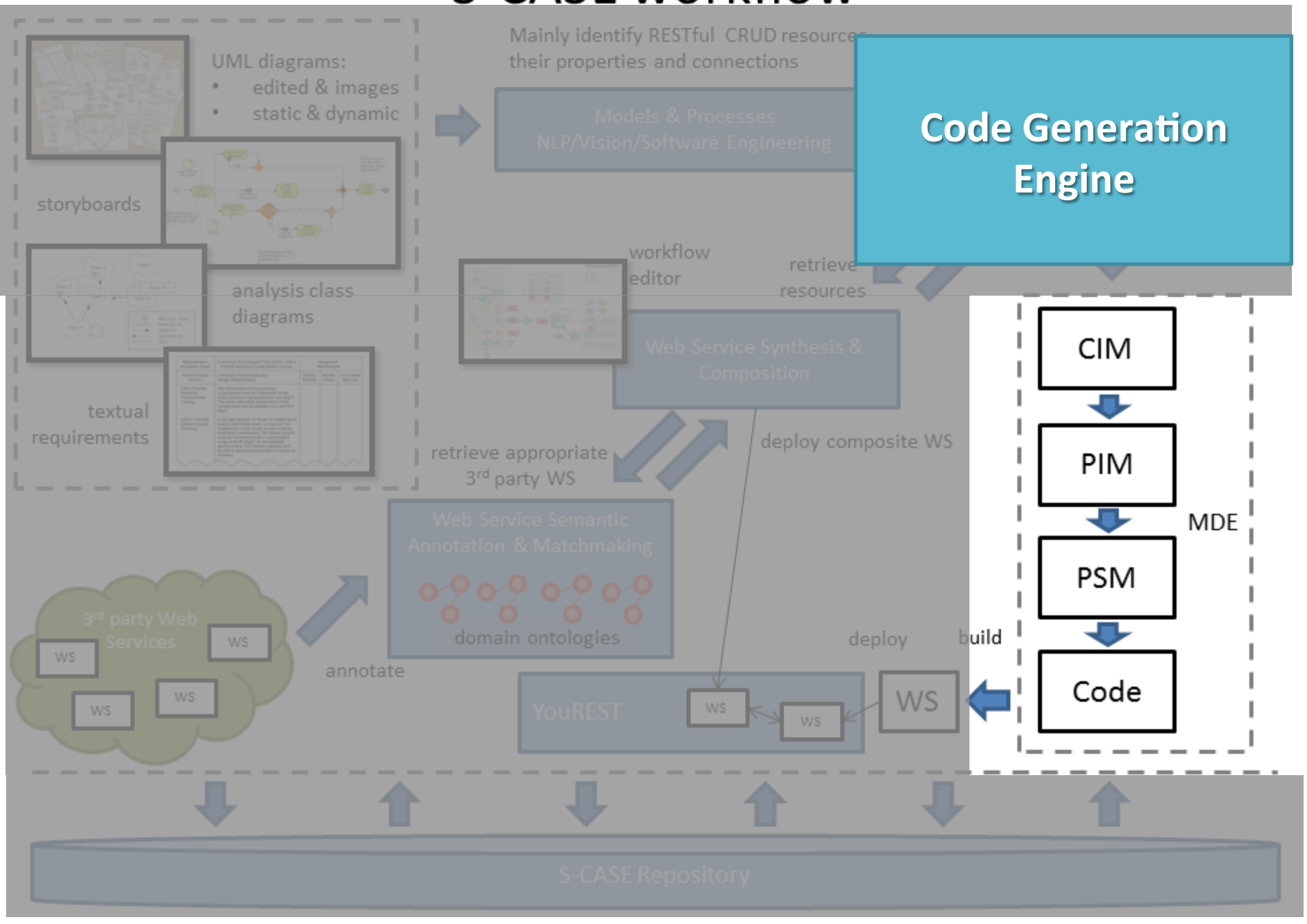
deploy

build

WS

S-CASE Repository

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

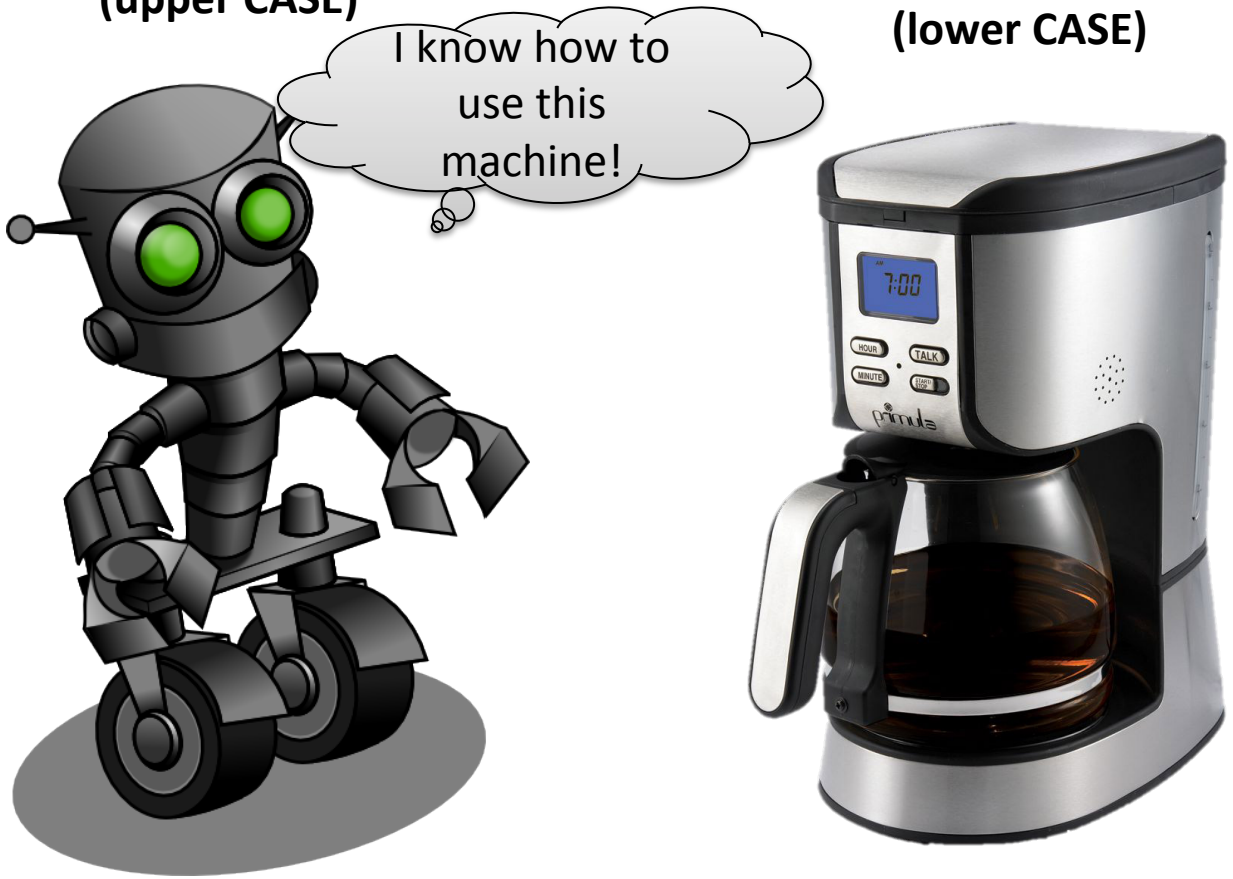
S-CASE Code Generation Engine (lower CASE)

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

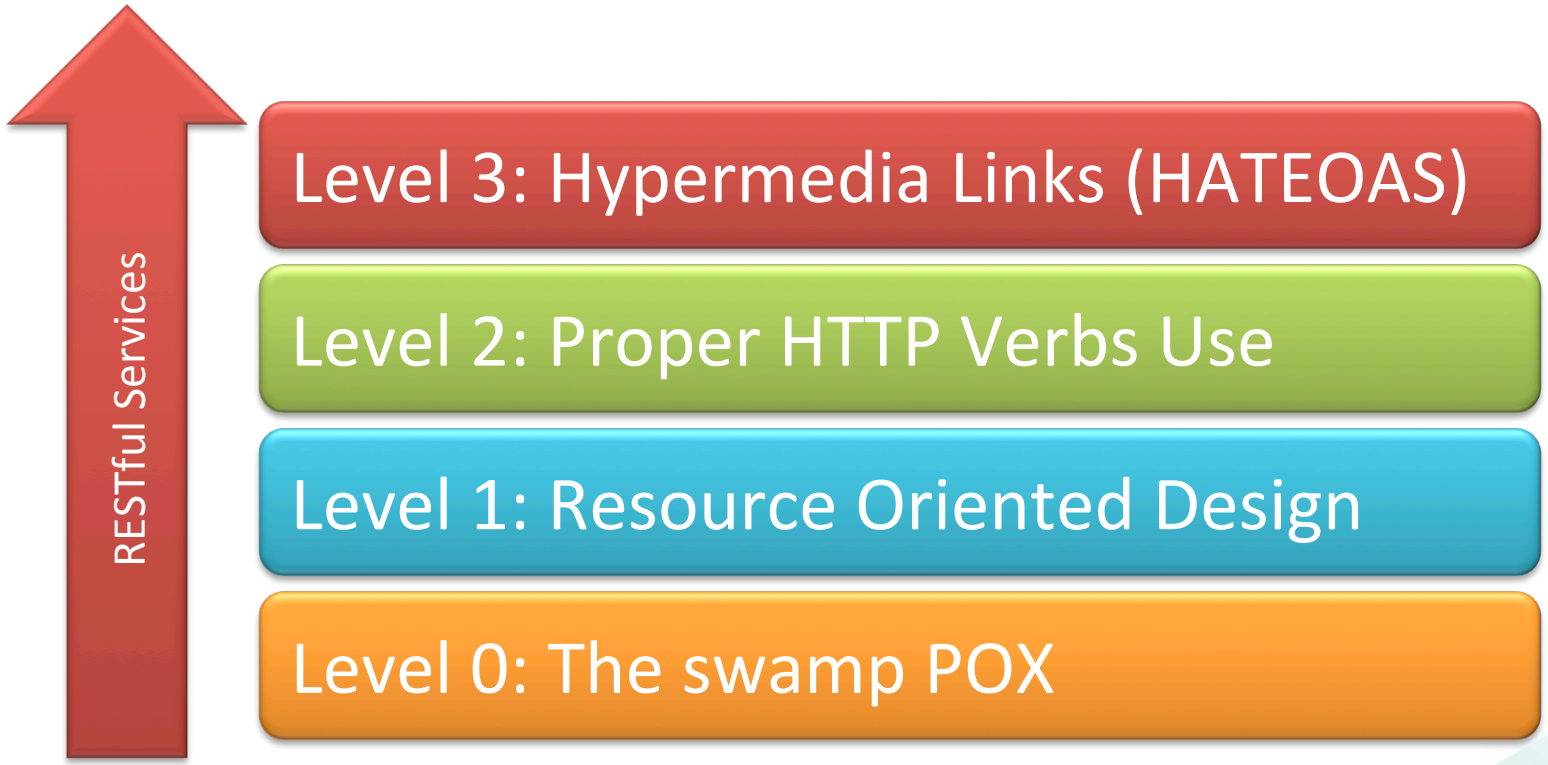
1) RESTful data API + database:

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:

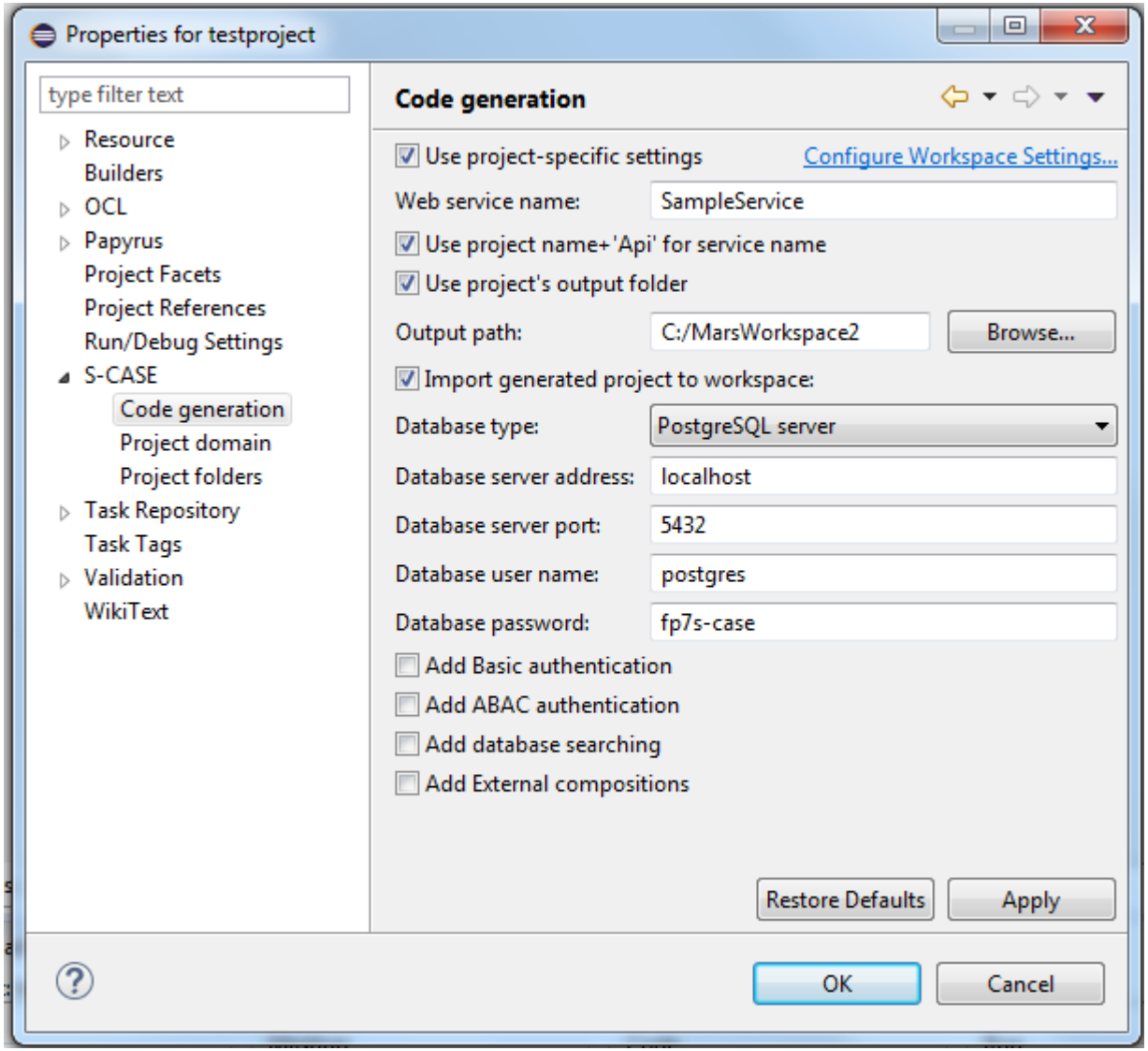
1. They model the service's data
2. They are capable of the 4 primitive CRUD actions:
   - CREATE (POST)
   - READ (GET)
   - UPDATE (PUT)
   - DELETE (DELETE)
3. The come always with Hypermedia Links.

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)

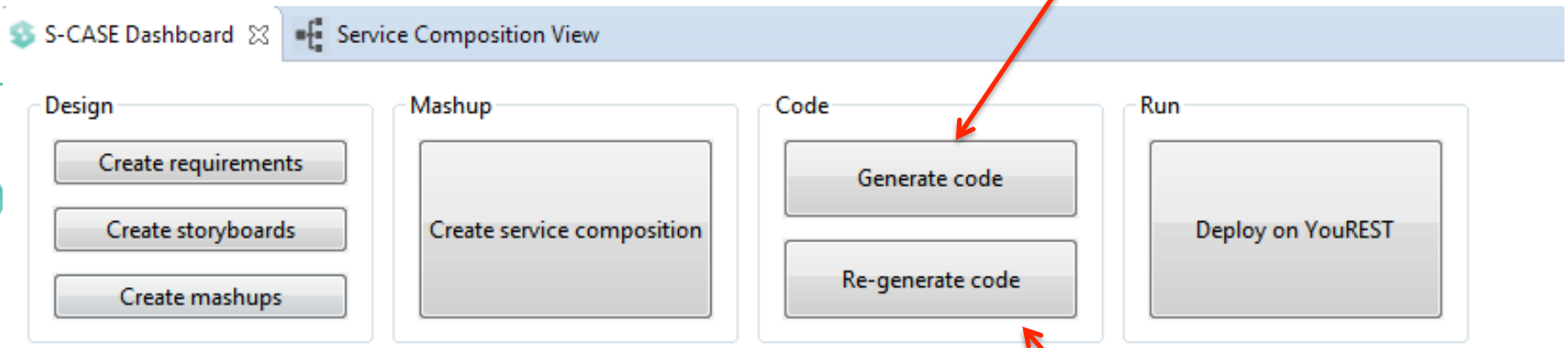
Code Generation Engine Preferences

Eclipse Preferences -> S-CASE -> Code Generation



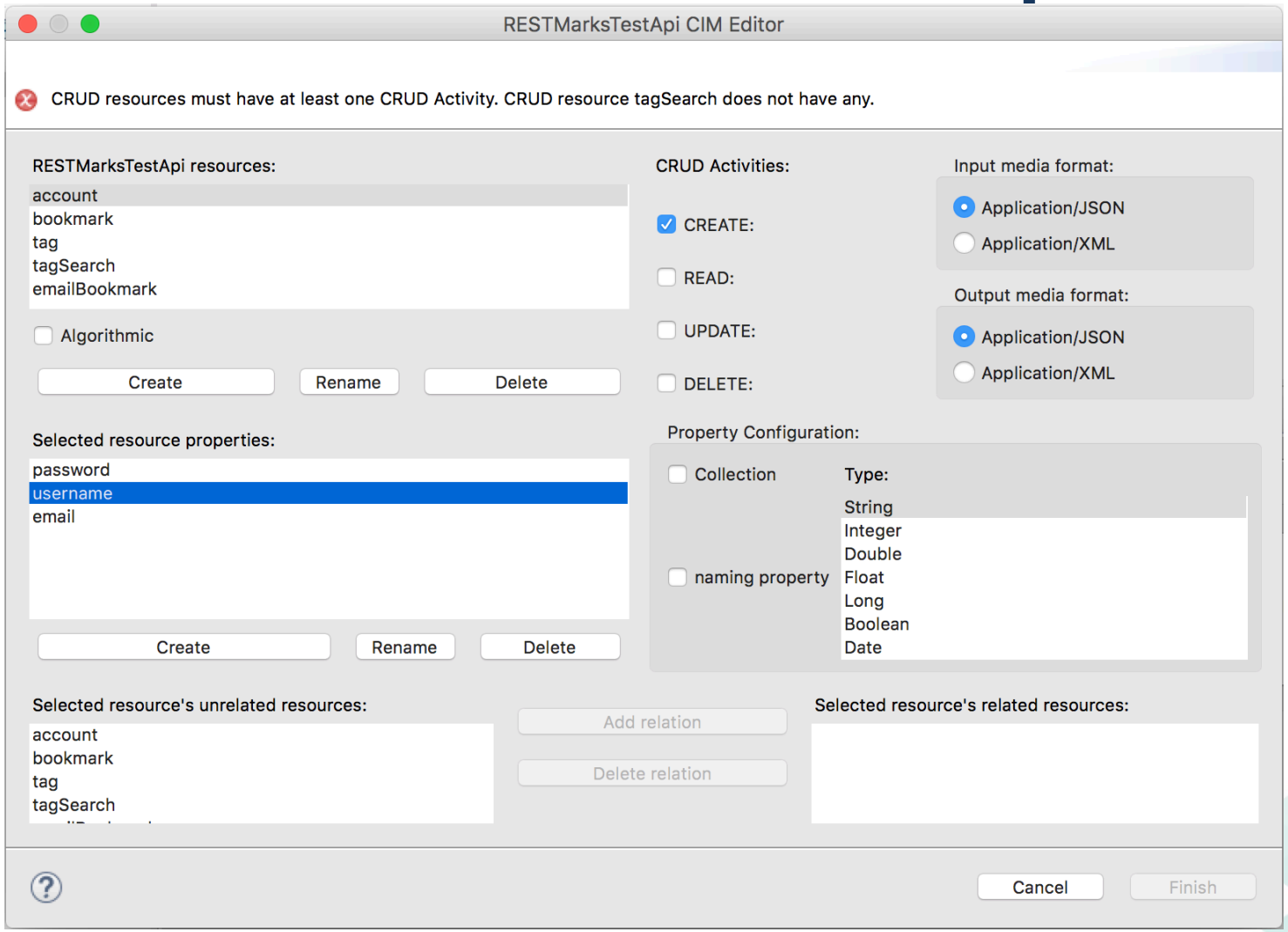
S-CASE Dashboard

Starts the Code Generation Engine



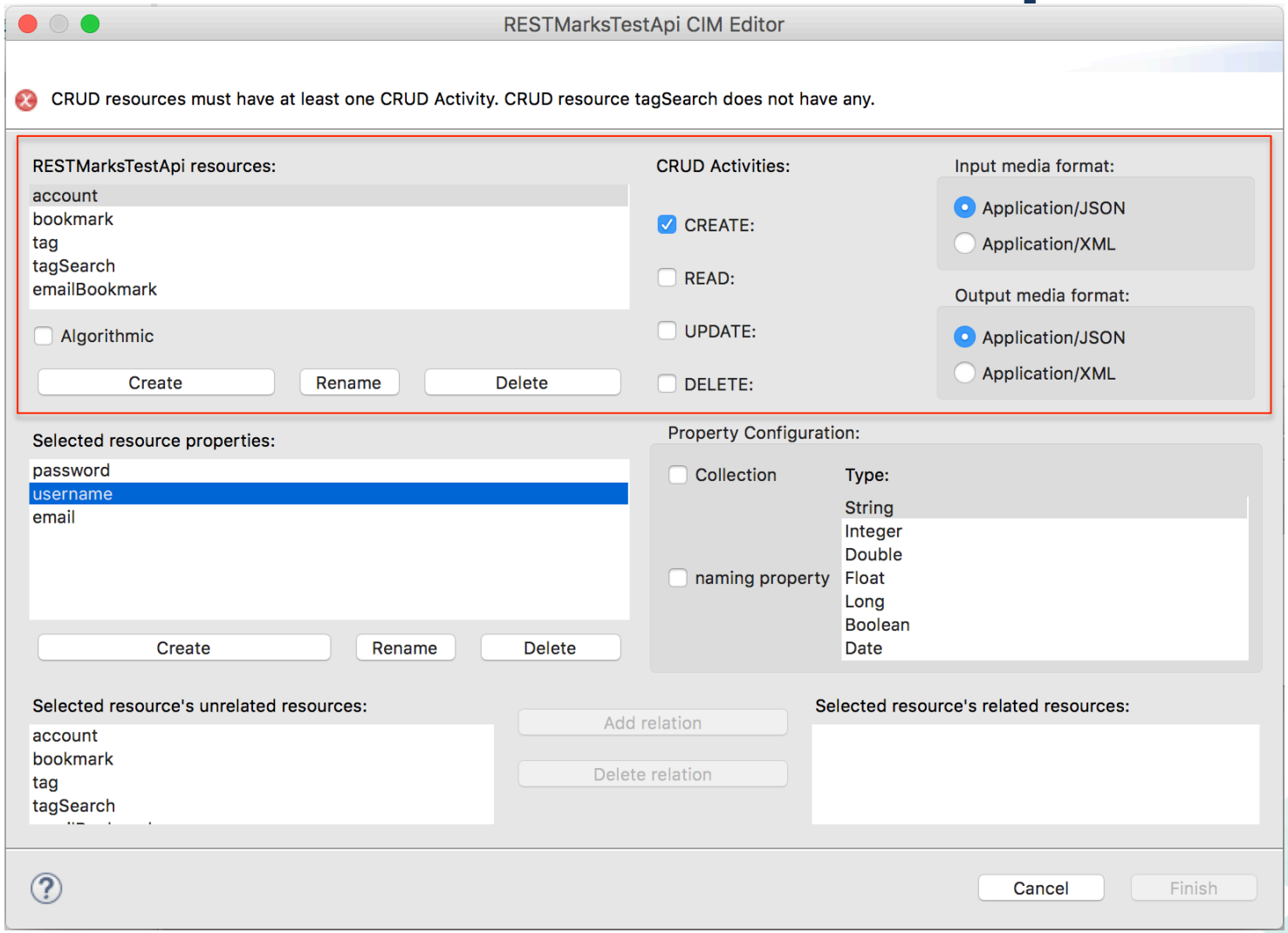
Starts the Code Generation Engine with the last saved state reloaded.

REST Wizard
User Interface and its Components



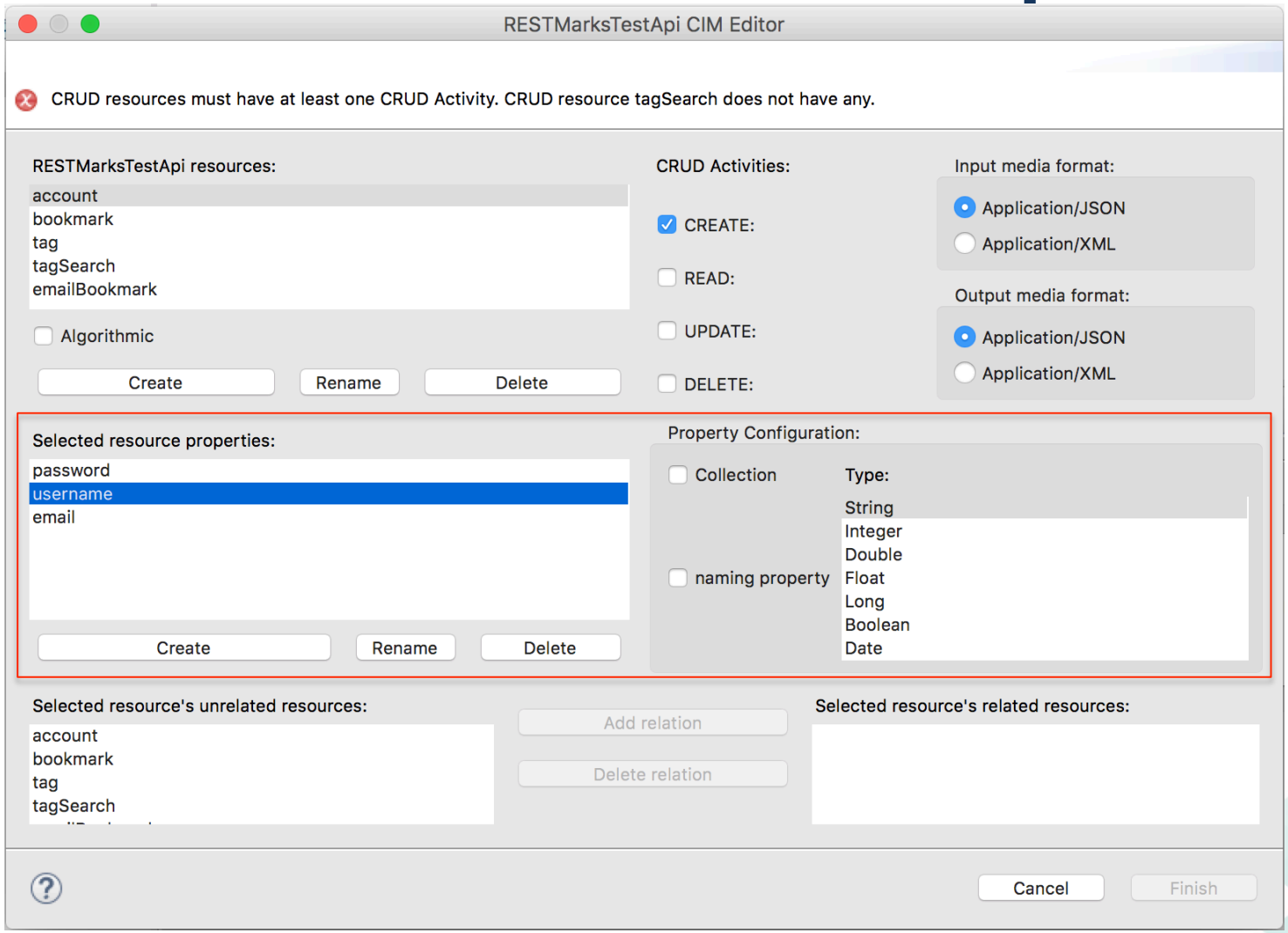
REST Wizard
User Interface and its Components

Model Resources

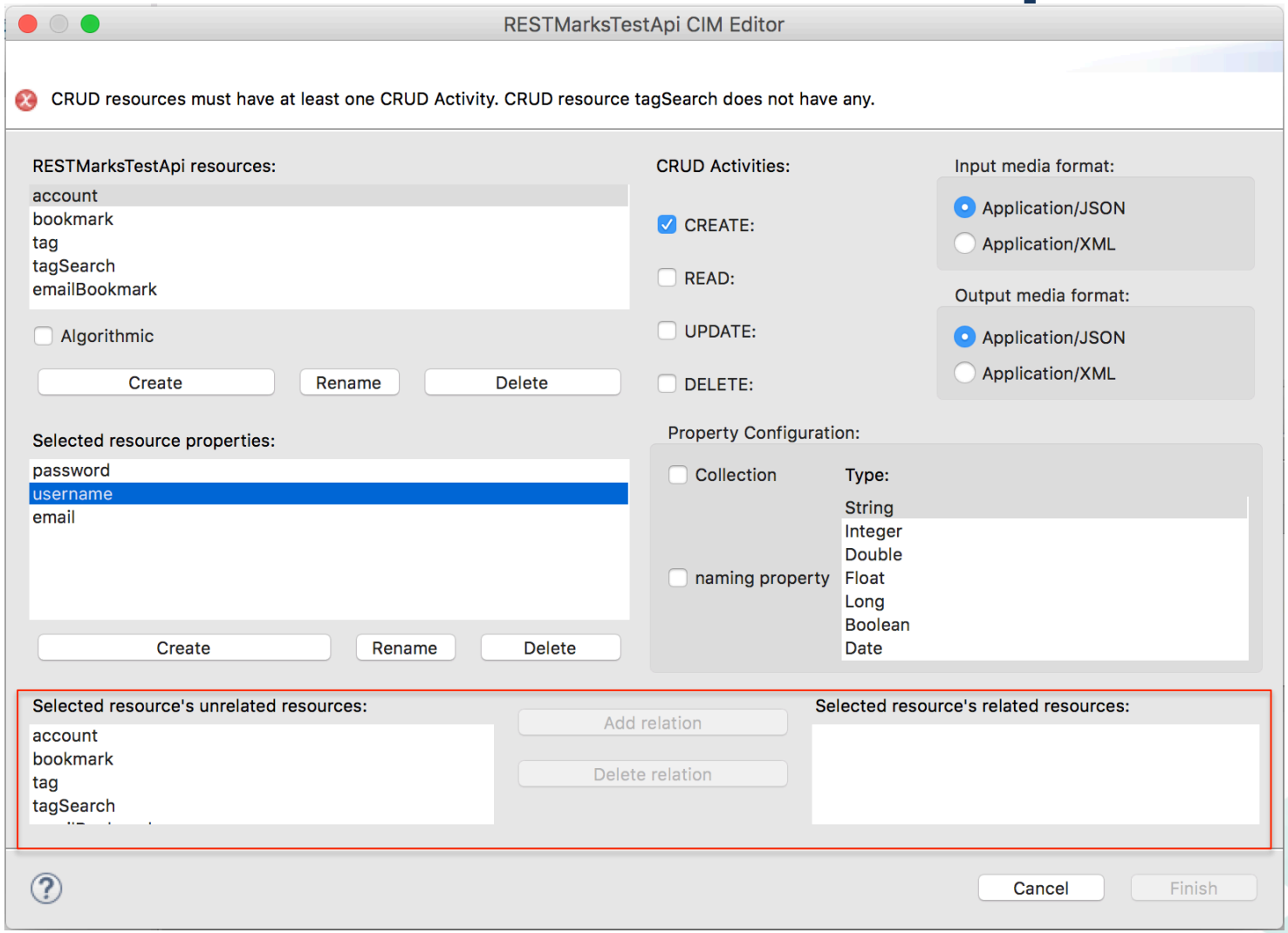


REST Wizard
User Interface and its Components

Model Properties



REST Wizard
User Interface and its Components

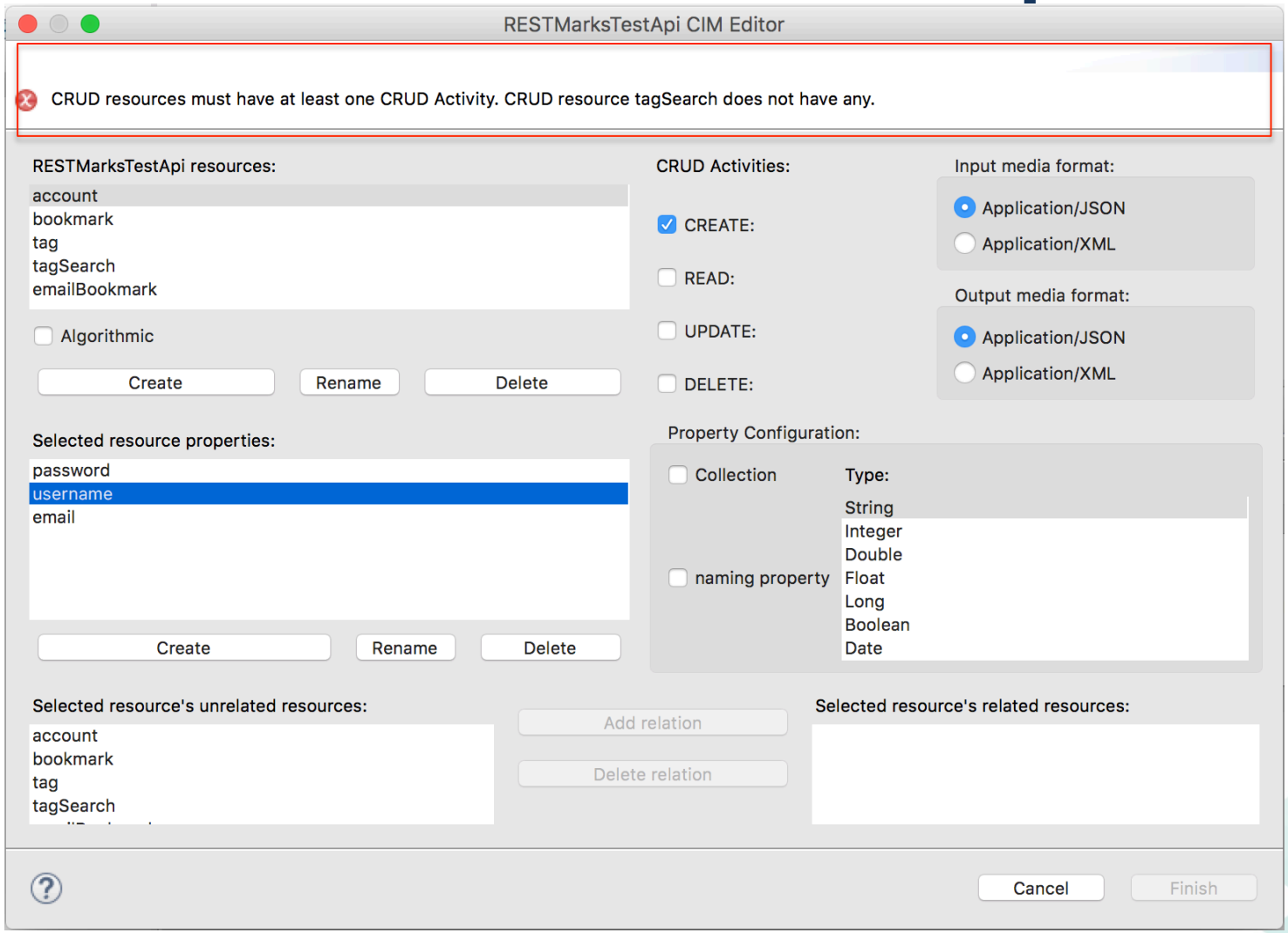


Model Relations

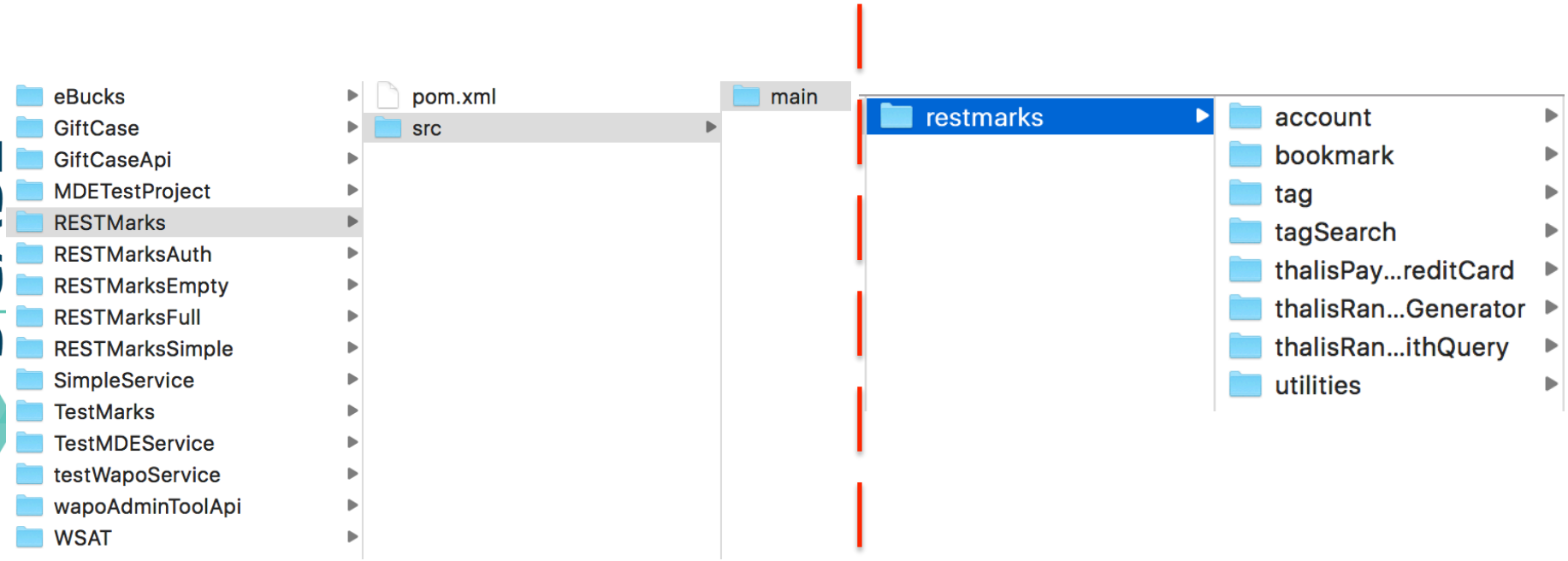
REST Wizard

User Interface and its Components

Validation Hints

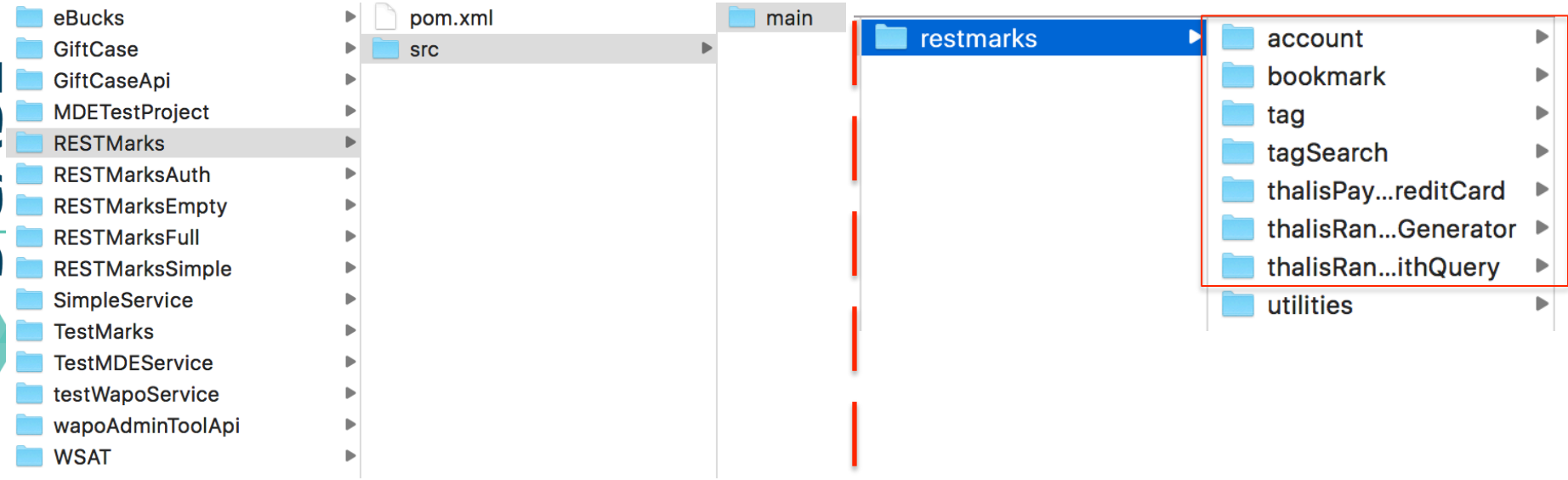


Code Generation Output Structure

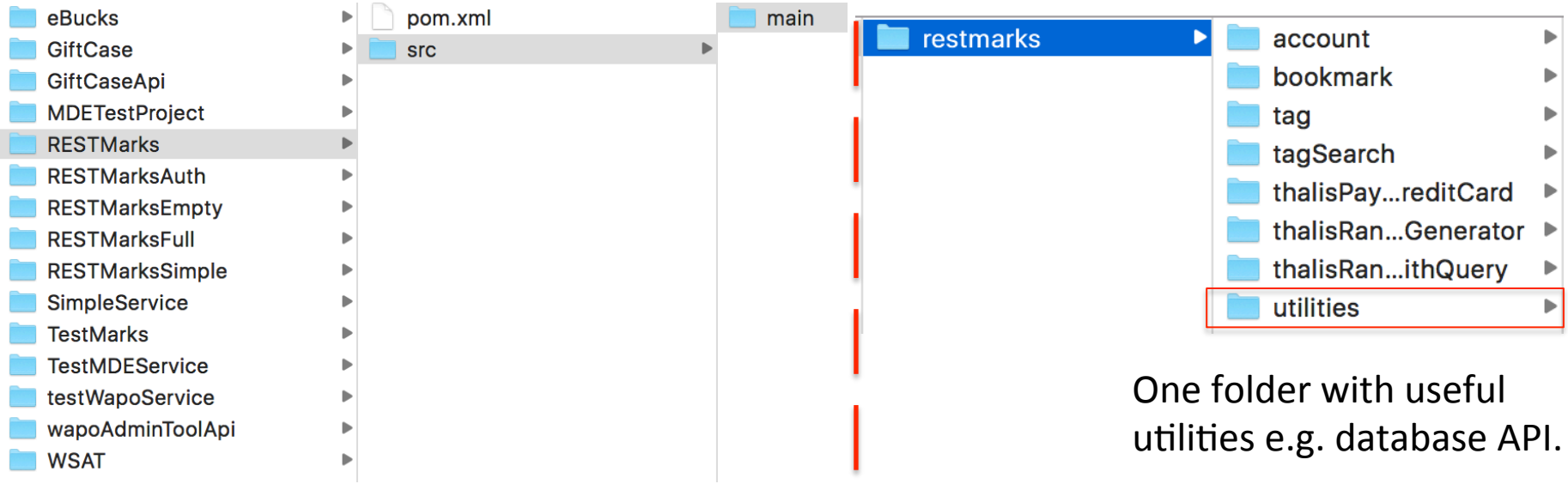


Code Generation Output Structure

One folder per Resource



Code Generation Output Structure



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 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.

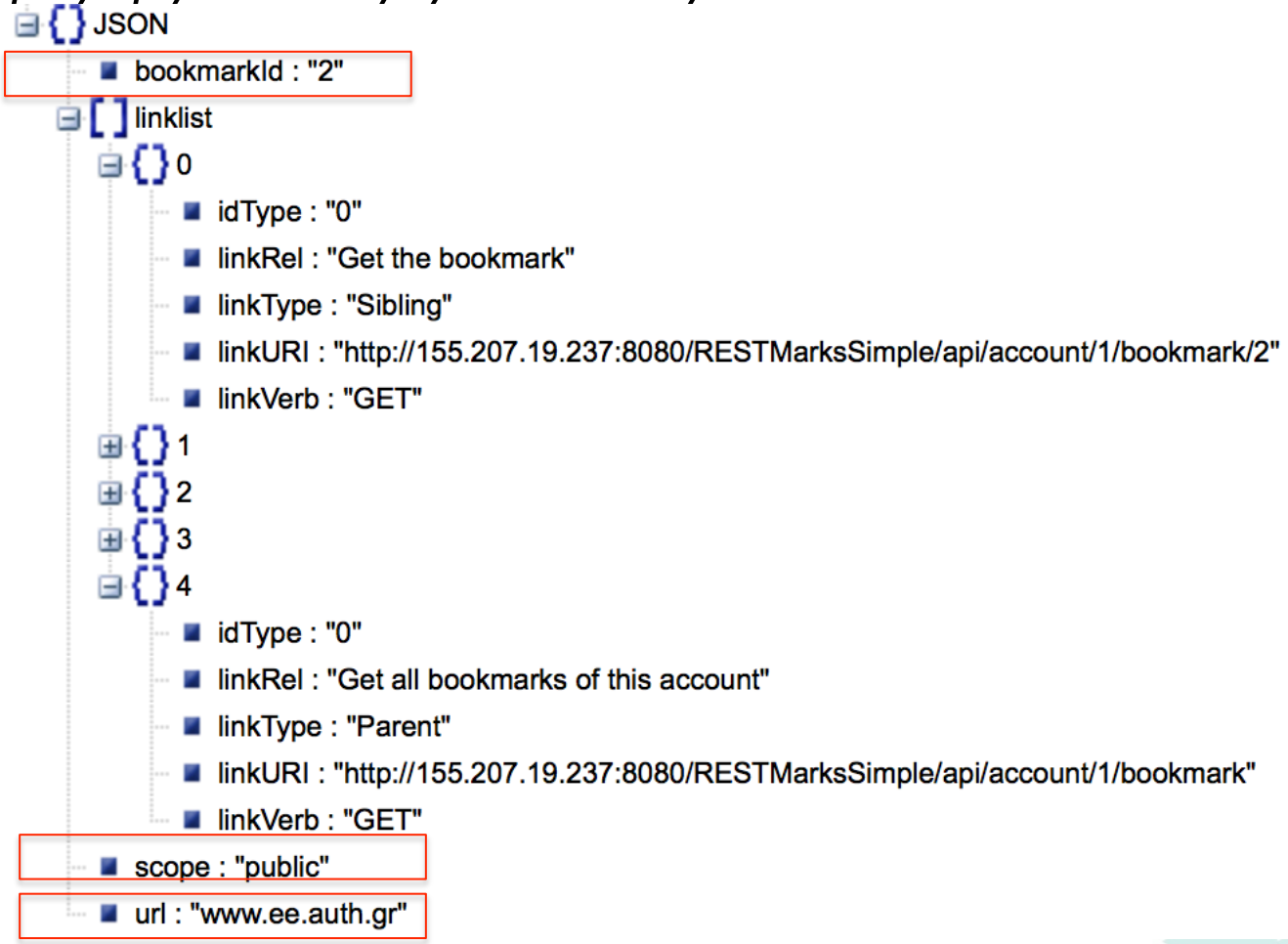
Interacting with the Generated Service

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*

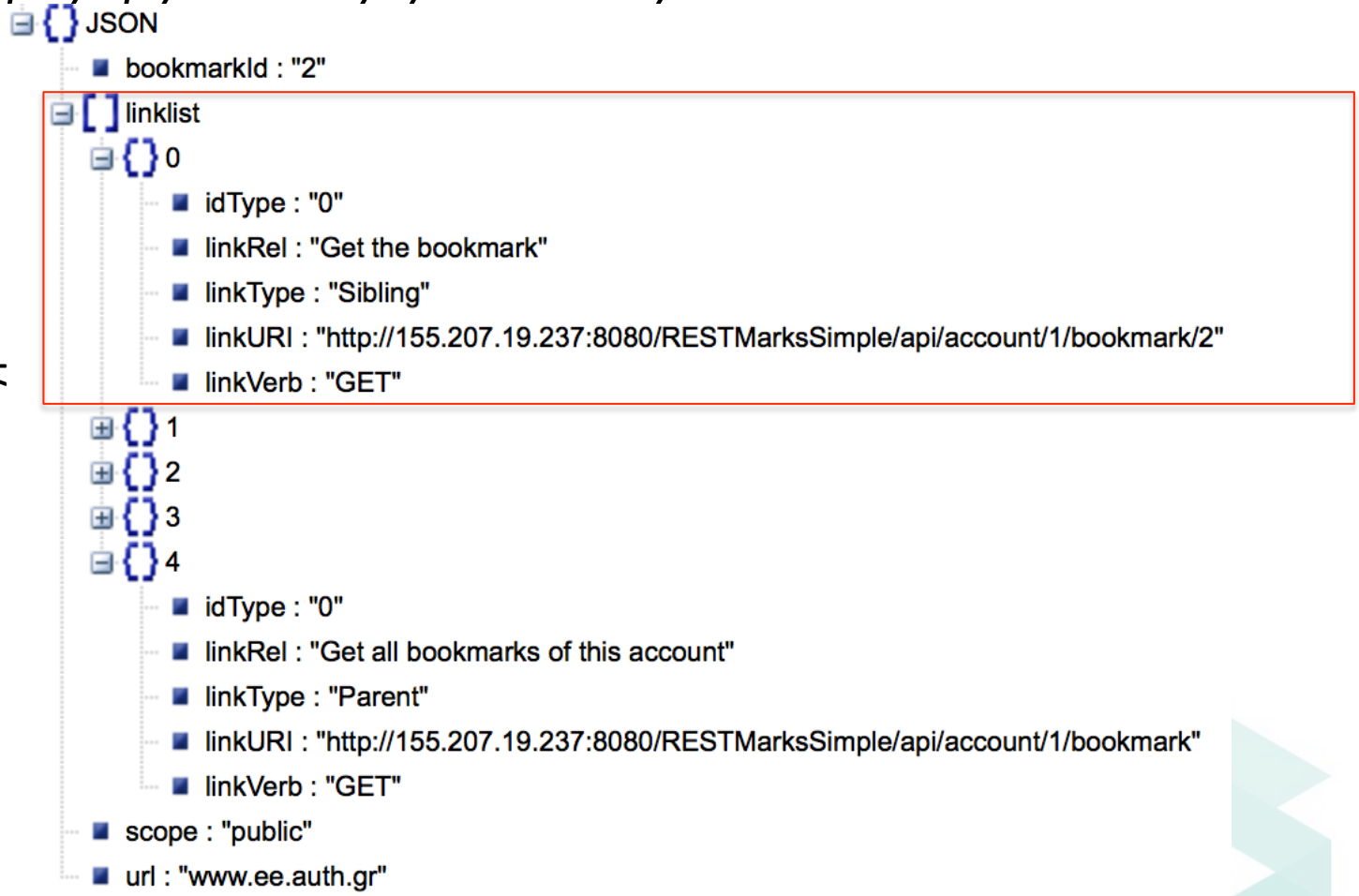
Resource Data



Interacting with the Generated Service

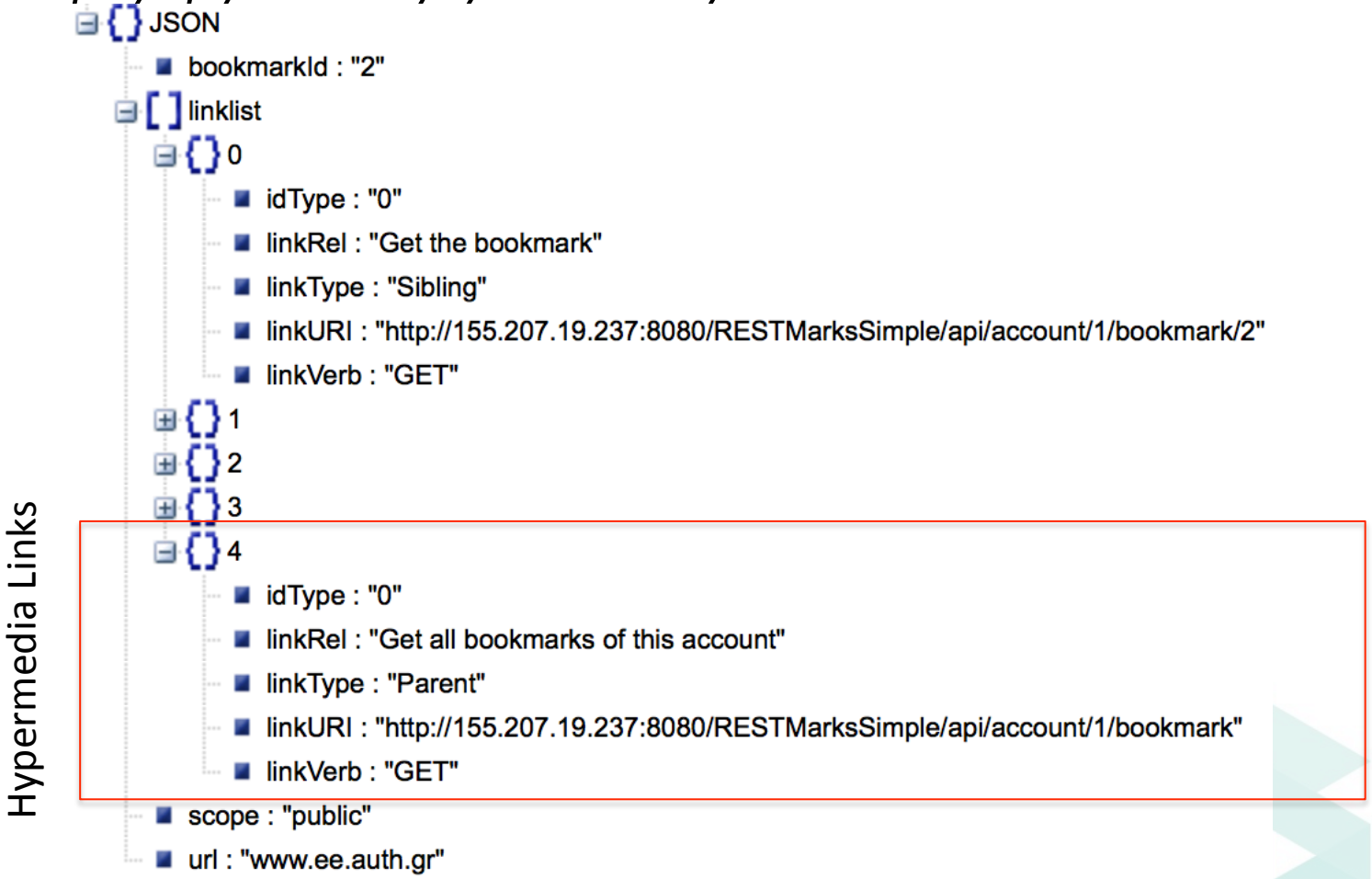
Reading a resource: *curl -i -X GET http://155.207.19.237:8080/RESTMarksSimple/api/account/1/bookmark/2*

Hypermedia Links



Interacting with the Generated Service

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

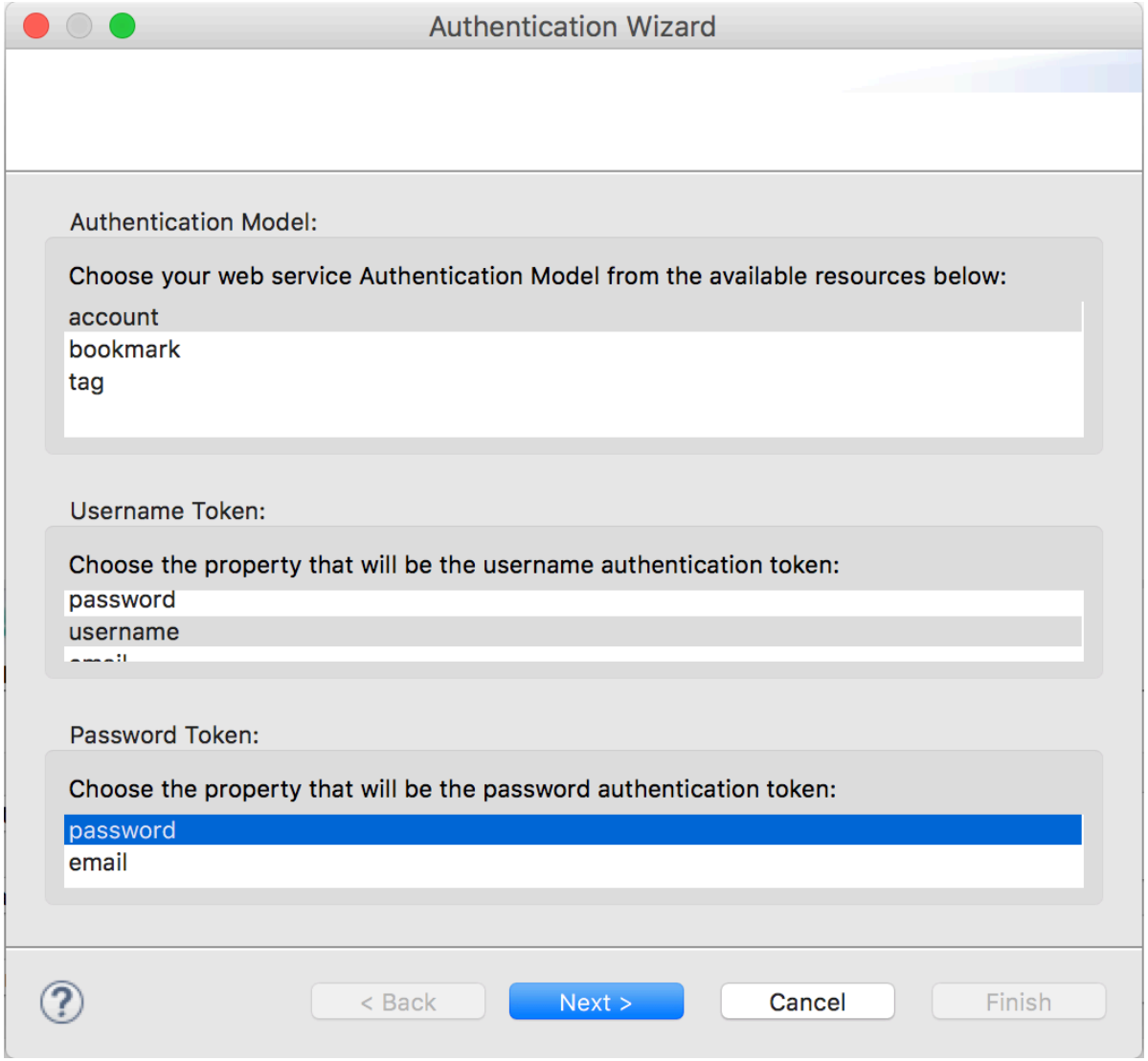


Embedding Basic Authentication

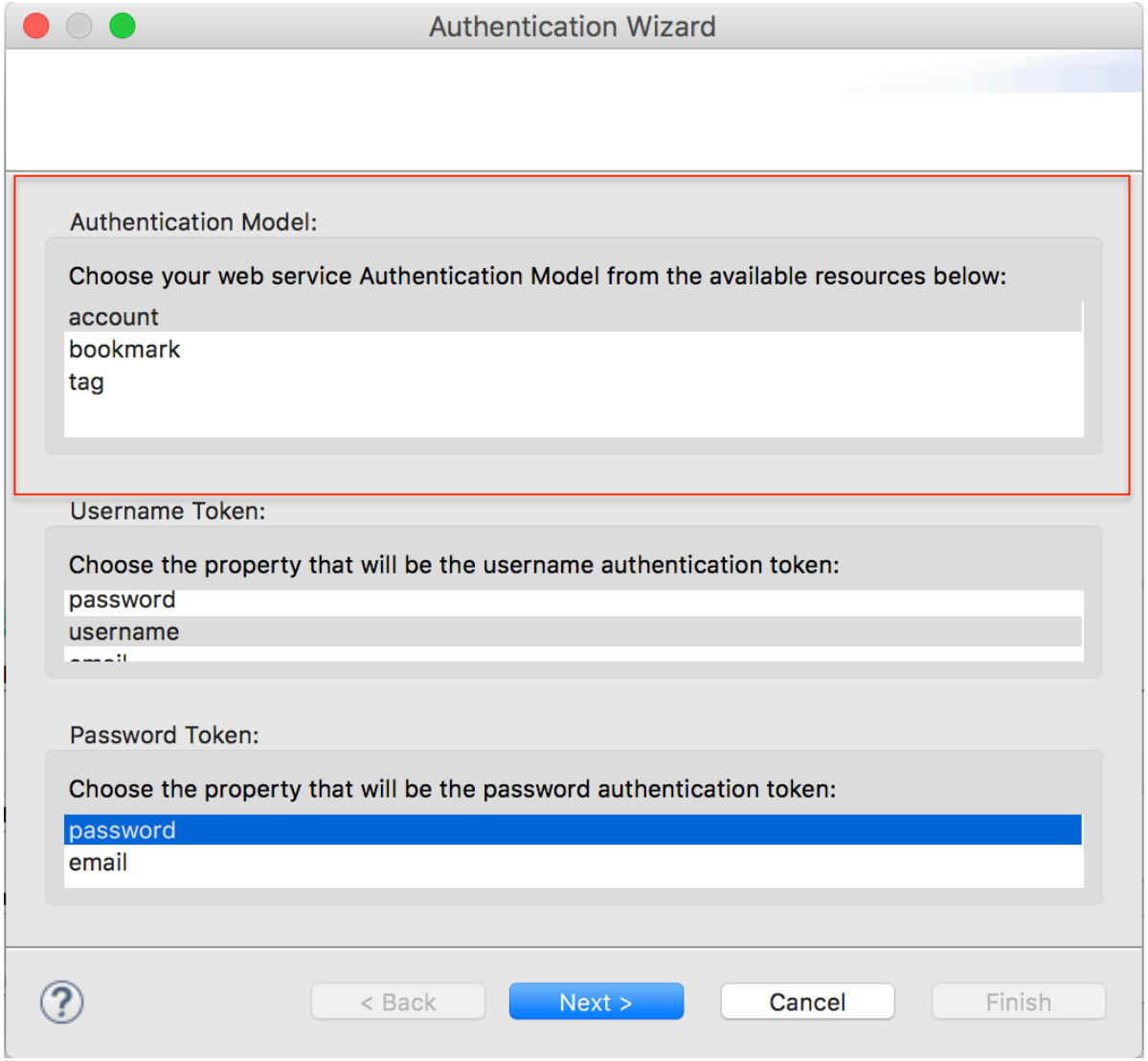
Provided automated functionality :

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



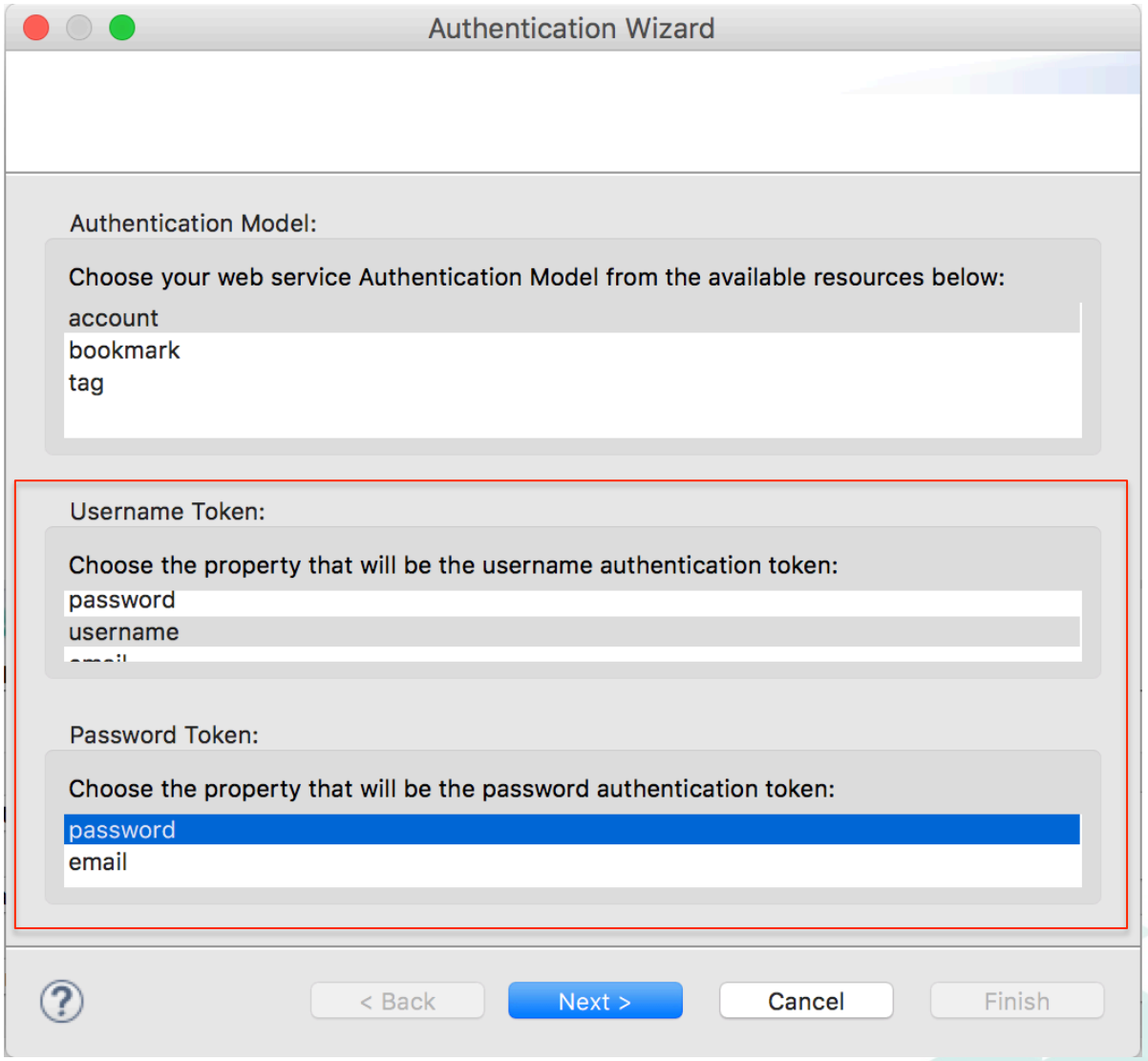
Embedding Basic Authentication



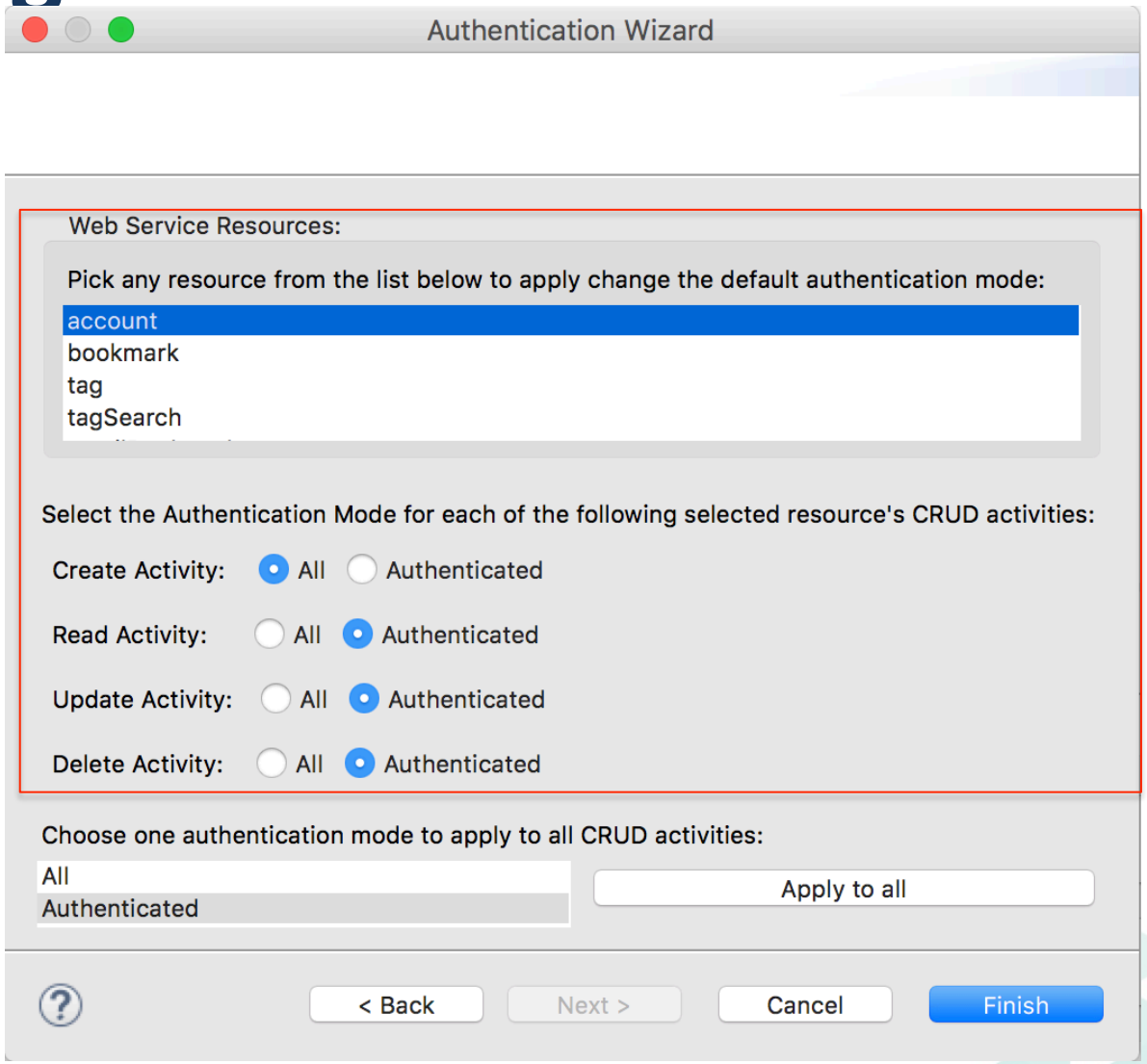
1) Select a CRUD resource as authentication model.

Embedding Basic Authentication

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

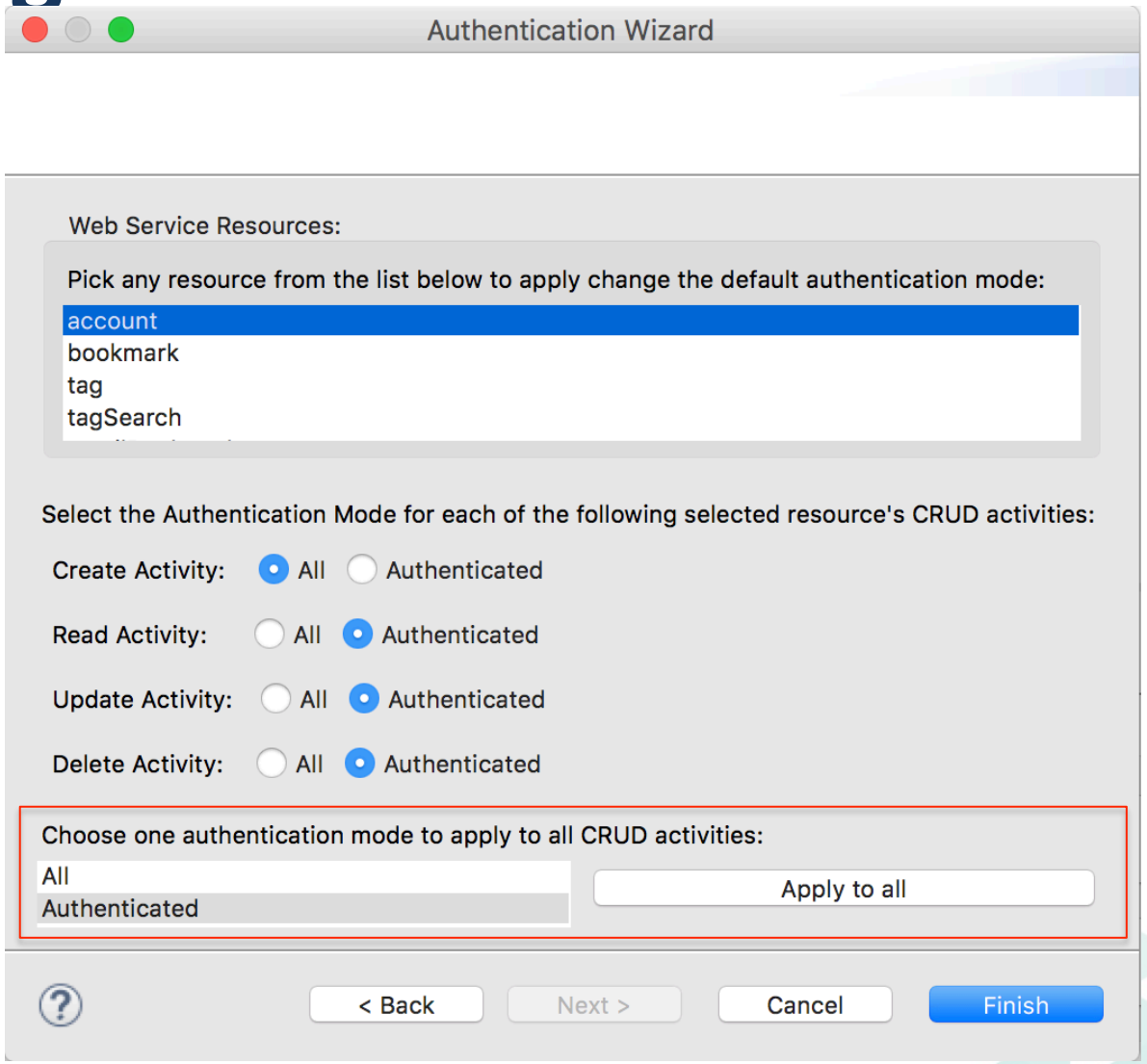


Embedding Basic Authentication



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

Embedding Basic Authentication



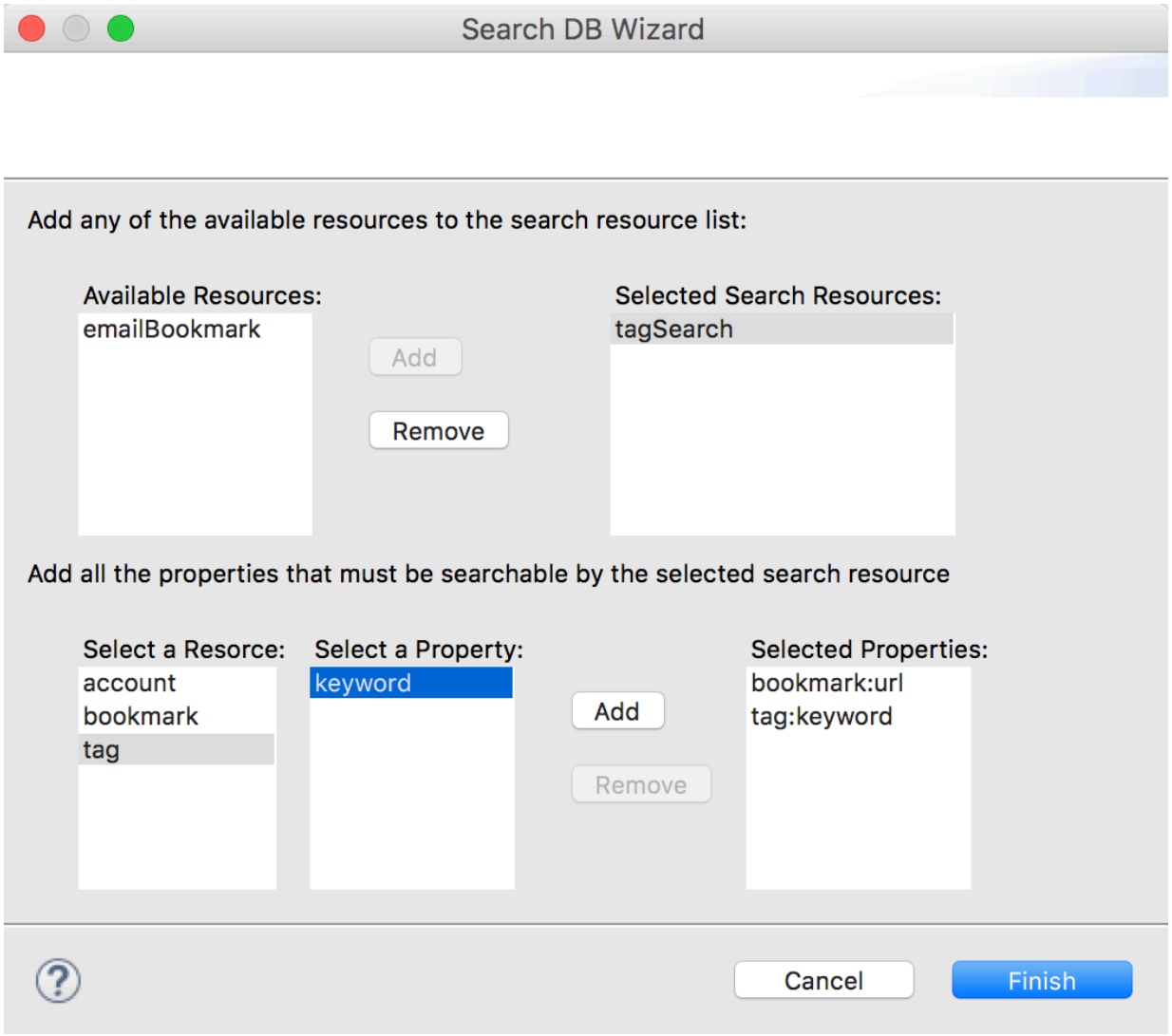
Optionally use the **one click** authentication **setup**.

Embedding Keyword Searching

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

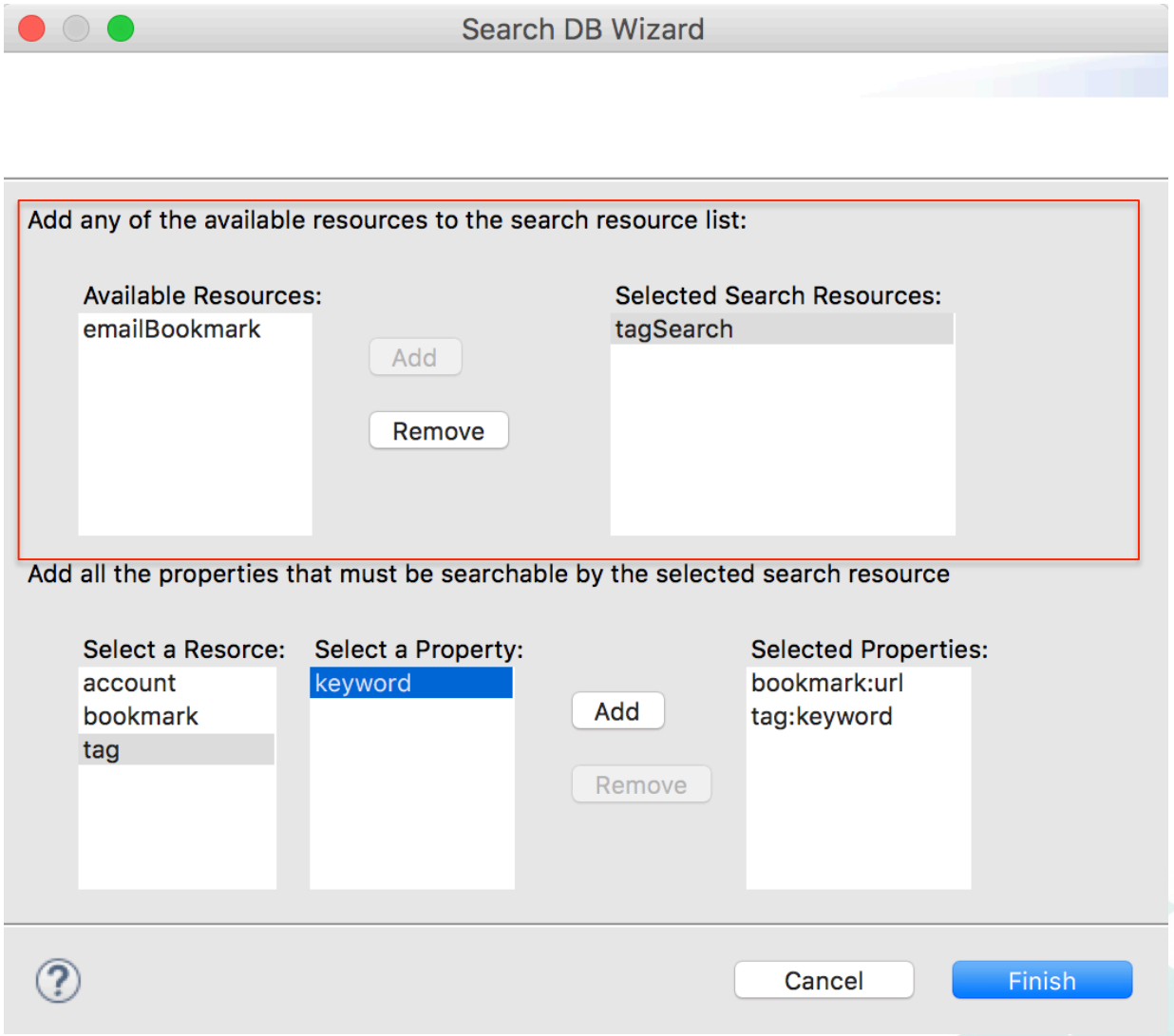
1. An **algorithmic resource** can be marked as a **Search Resource**, namely one that performs the search activity.
2. A **CRUD resource and** some of **its properties** can be marked as **Searchable**, namely they will be indexed and searched by a Search Resource.

Embedding Keyword Searching

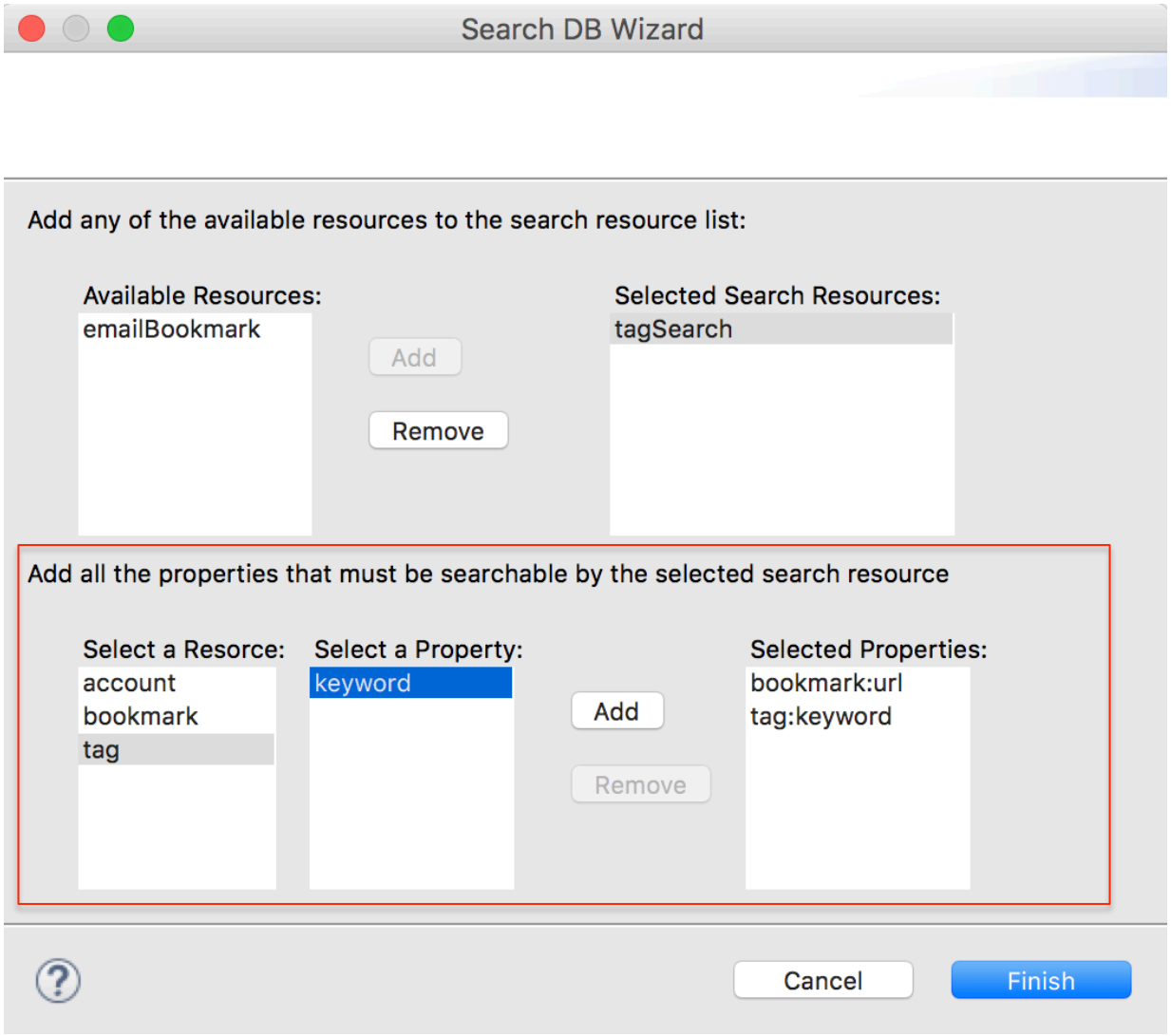


Embedding Keyword Searching

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



Embedding Keyword Searching



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'


```
JSON
  linklist
    idType : "0"
    linkRel : "Search result"
    linkType : "bookmark"
    linkURI : "http://155.207.19.237:8080/RESTMarksFull/api/account/1/bookmark/2"
    linkVerb : "GET"
```


```
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'


```
JSON
  linklist
    idType : "0"
    linkRel : "Search result"
    linkType : "bookmark"
    linkURI : "http://155.207.19.237:8080/RESTMarksFull/api/account/1/bookmark/2"
    linkVerb : "GET"
```

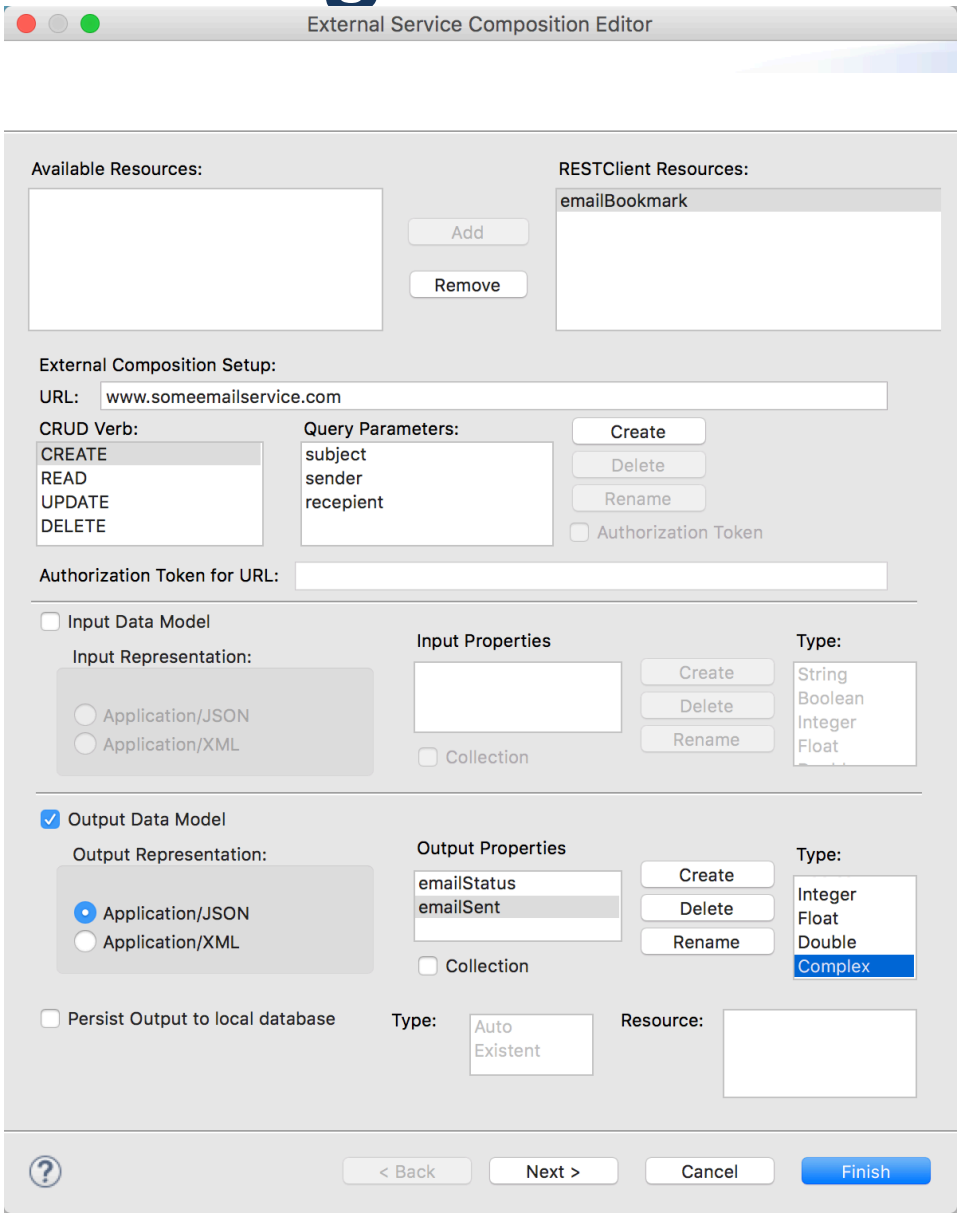

```
public JavaAlgobookmarkSearchModel getbookmarkSearch( @DefaultValue("guest") @HeaderParam("authorization") String
    authHeader,  @QueryParam("searchKeyword") String searchKeyword, @QueryParam("searchBookmarkKeyword") String
    searchBookmarkKeyword){
```

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.

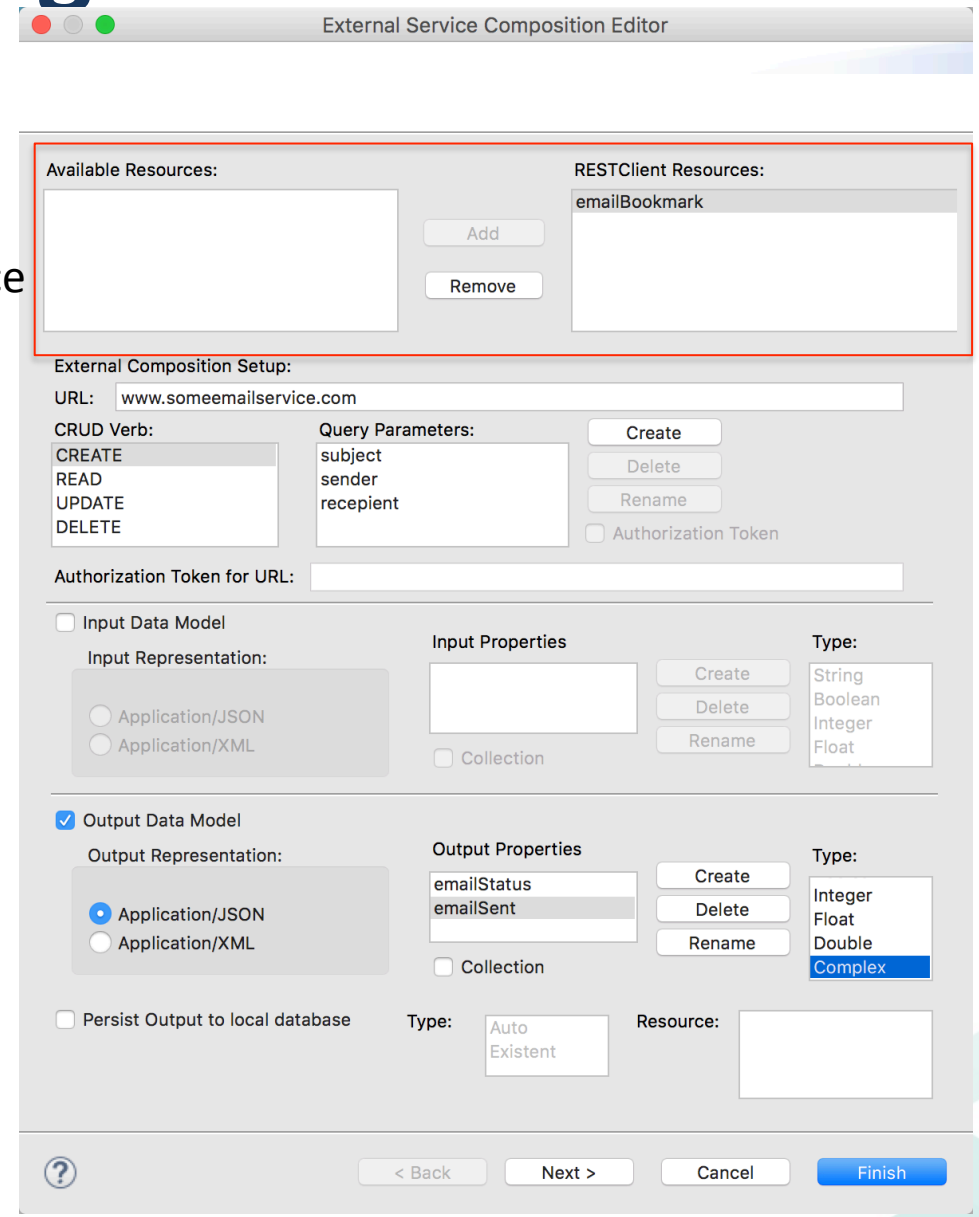
3rd Party Service as a black box

Interacting with the Web

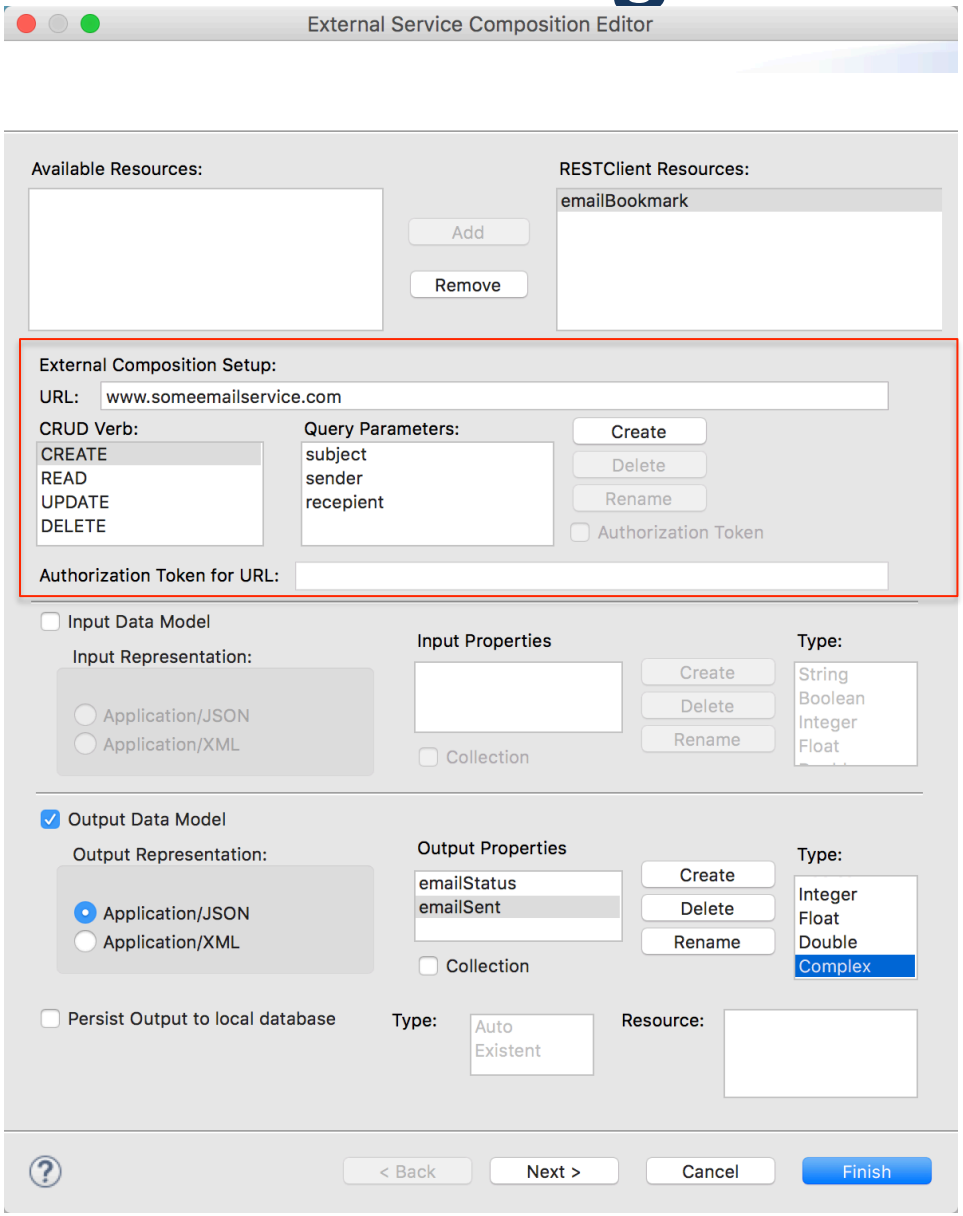


Interacting with the Web

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



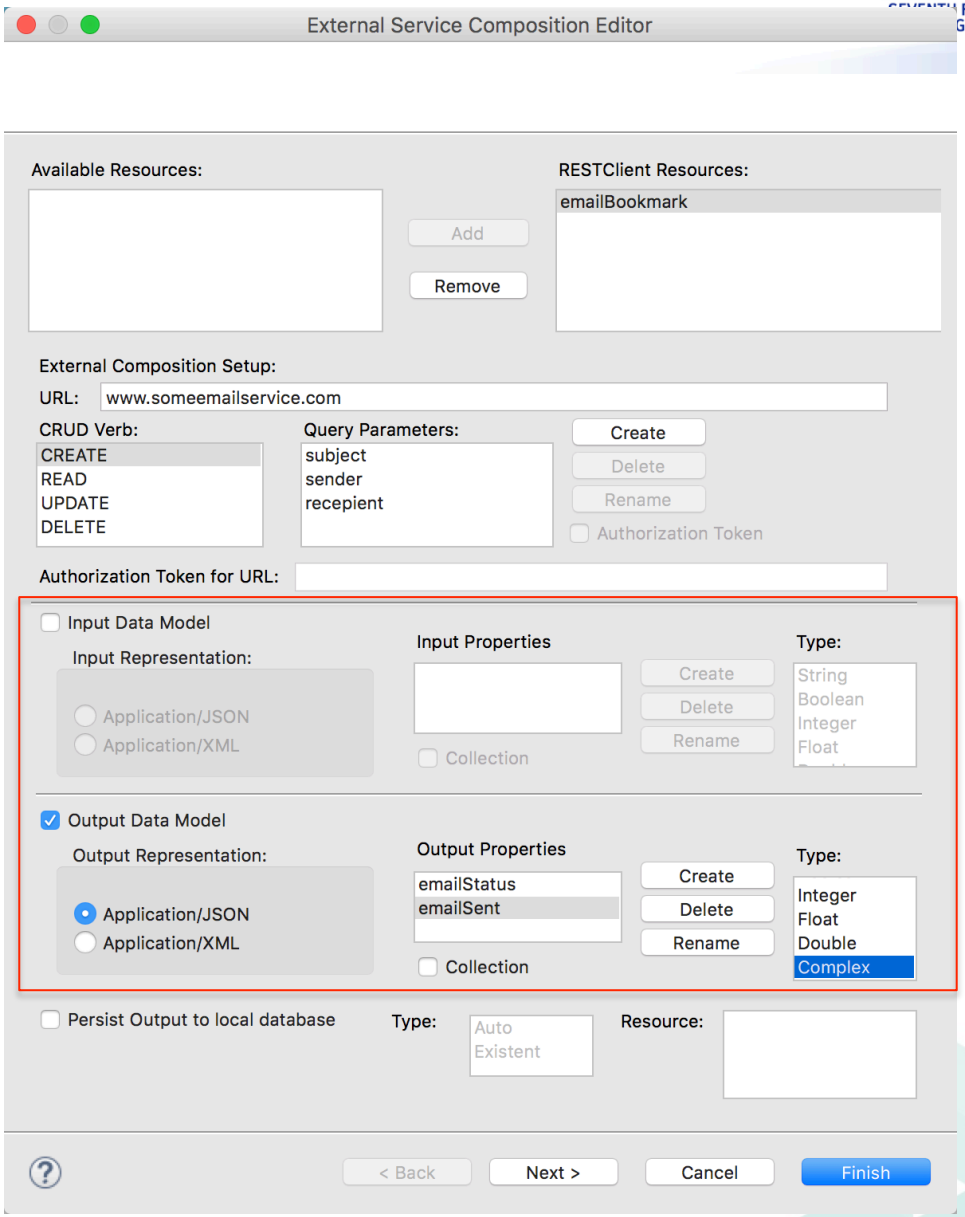
Interacting with the Web



2) Provide for each RESTClient resource the connection details:

- URL
- CRUD Verb
- Any Query Parameter

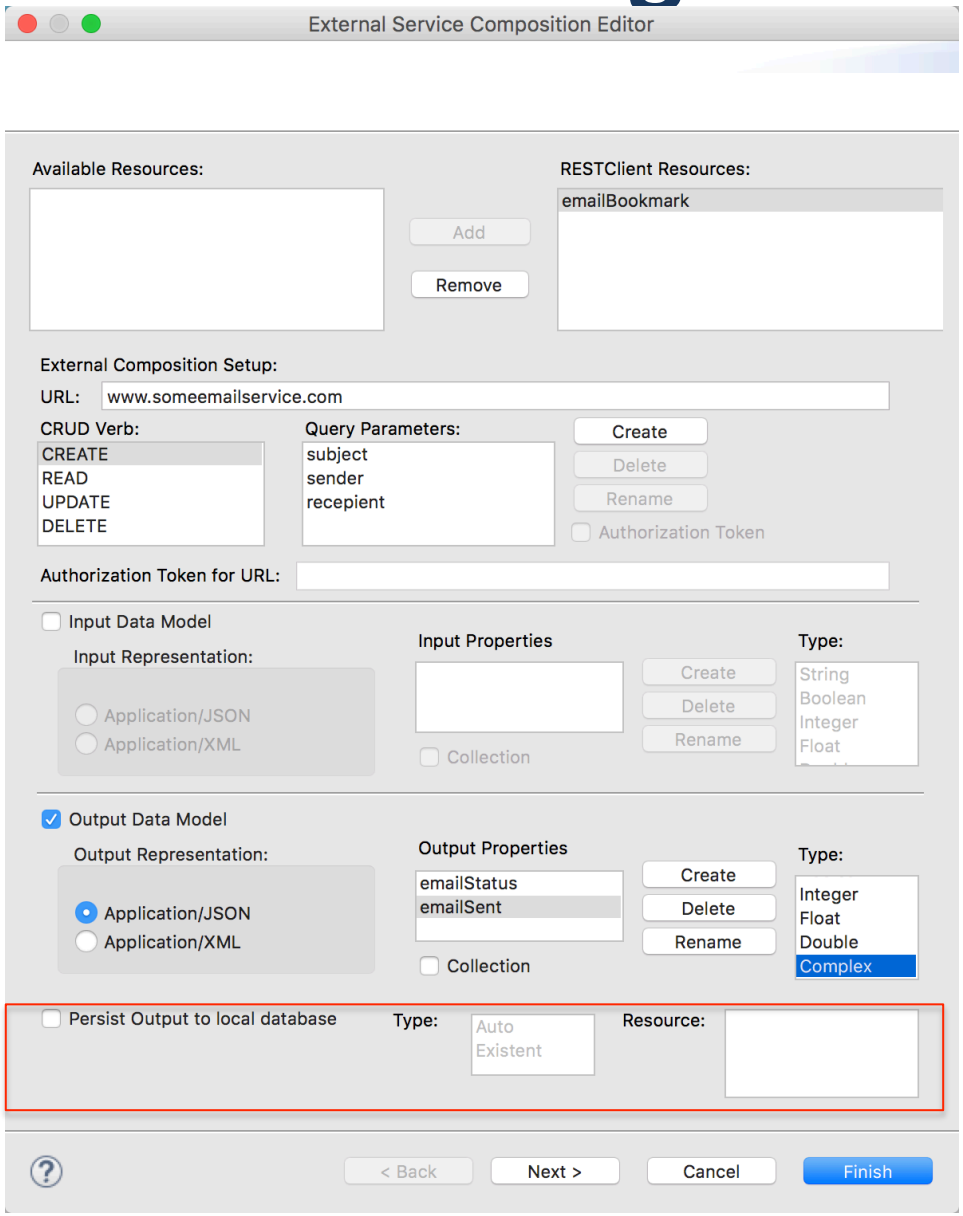
Interacting with the Web



3) Model the required input and/or output of the external service:

- Specify media representation
- Properties and their types

Interacting with the Web



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

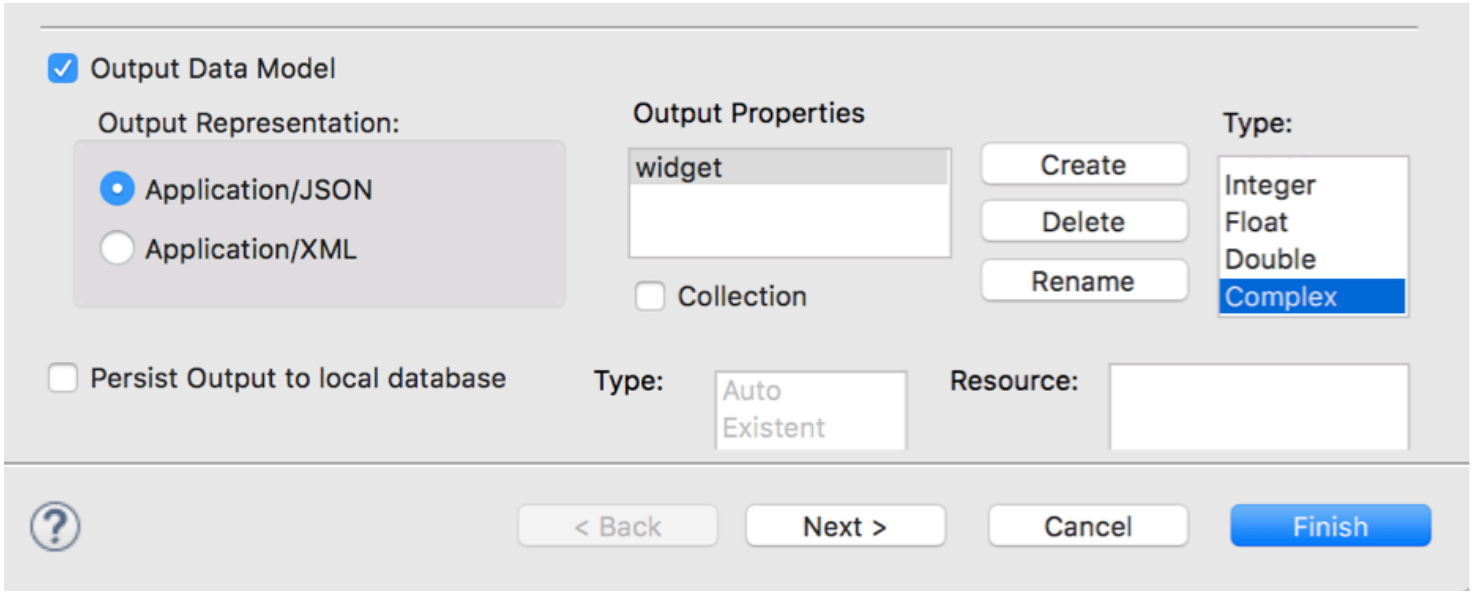
Modeling Complex Input/Output

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.

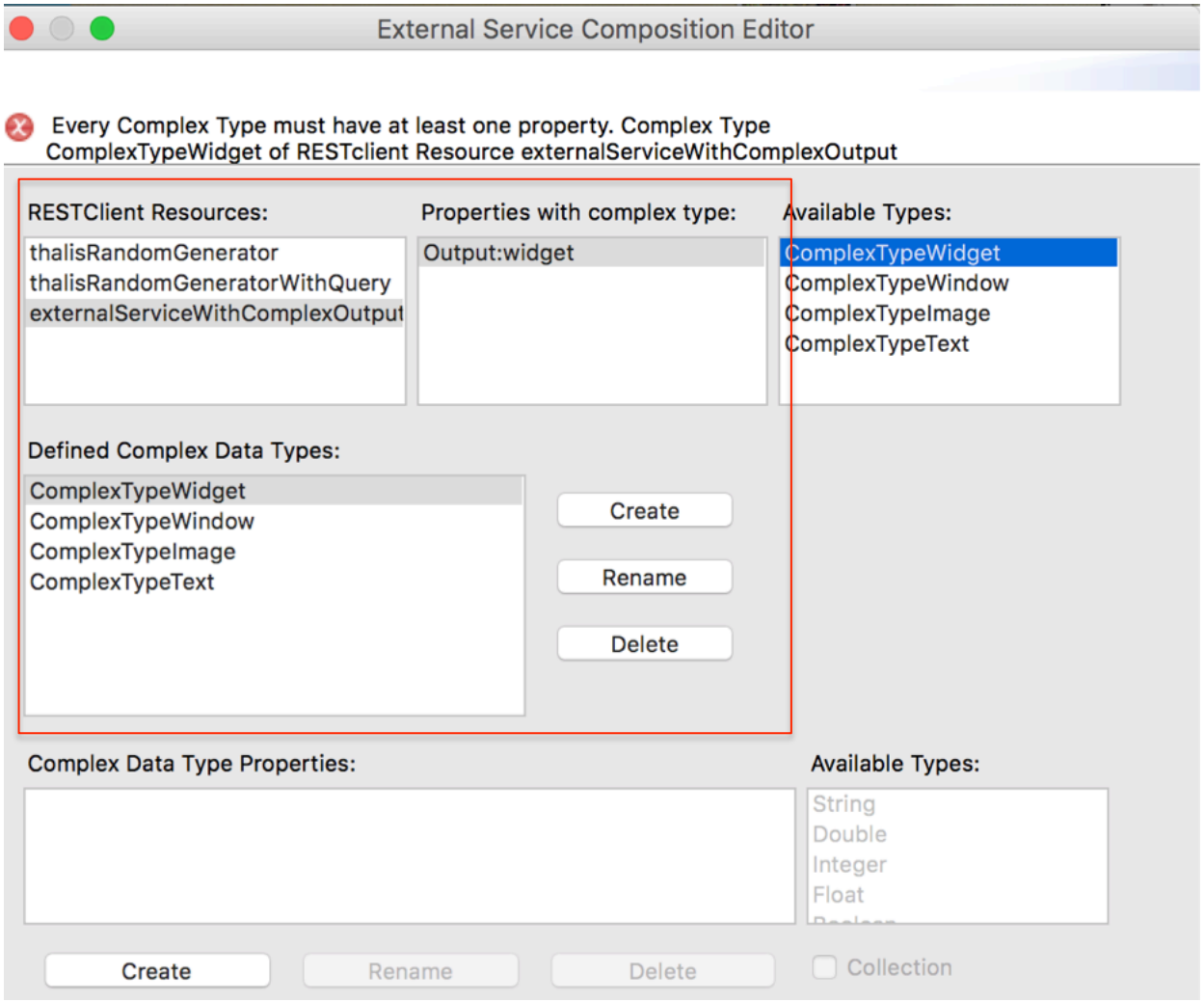


Modeling Complex Input/Output

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

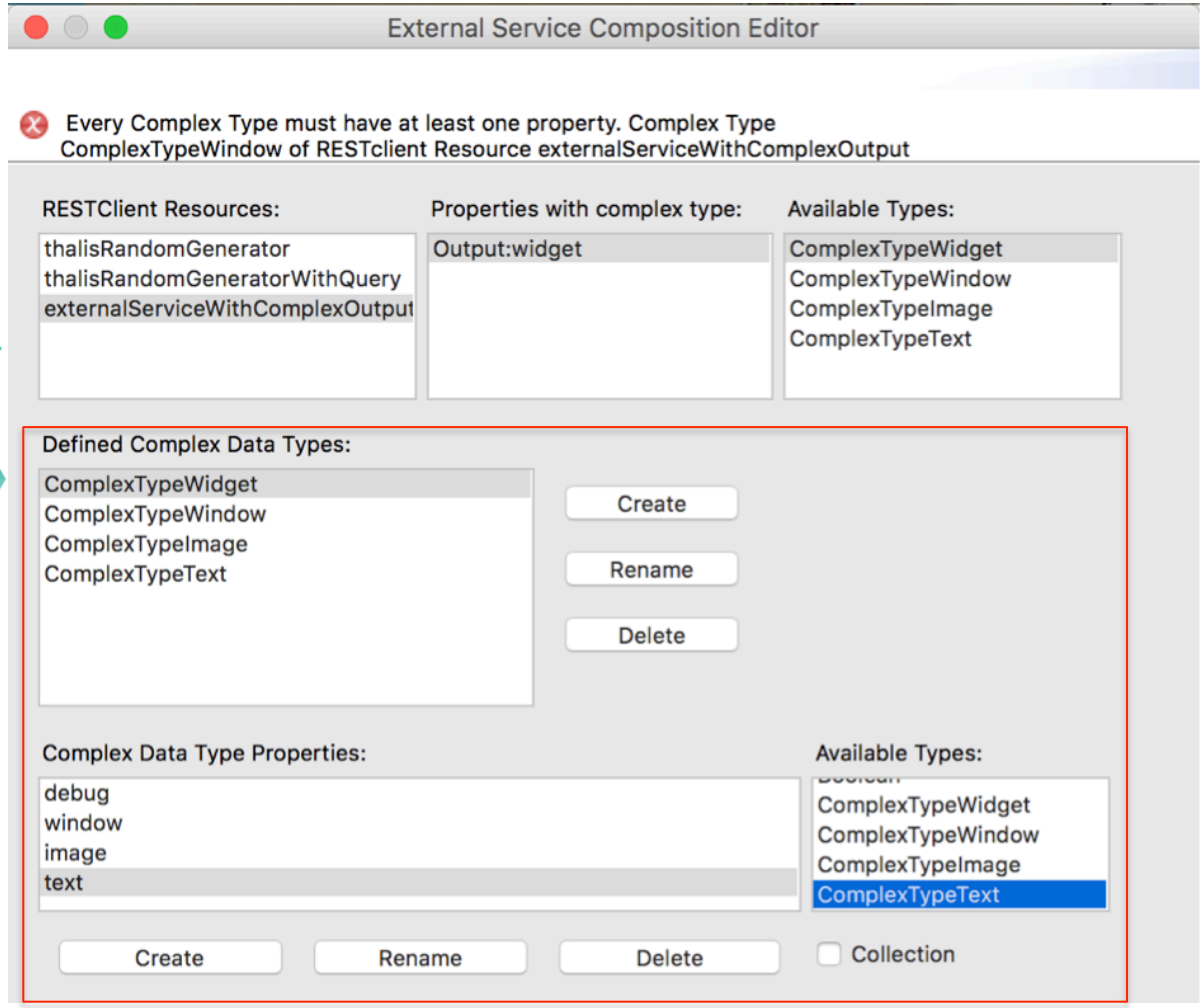
```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "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



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

Modeling Complex Input/Output



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