# Web Architectures

# Assignment 3



Andrew Munro

# Web Architectures Assignment 3: Implementing a RESTful web service using Groovy and Grails.

Table of Contents.

Instructions for cloning and building the project. – Page 3

Work undertaken for each of the tasks. – Page 4

Personal reflection – Page 8

# Instructions for cloning and building the project.

This project can be checked out of Github by cloning the repository. In order to clone the repository you must navigate to an empty folder or create an empty folder (mkdir <FOLDERNAME>). Once you have navigated to the correct folder you have to clone the repository by entering the following commands into the command line:

*git init*

*git clone:generic-gamer/WebArc3.git*

This will clone the contents of the WebArc3 repository to the local folder.

In order to build the project you will need Grails and all dependencies installed and will need to change directory to the root directory of the project (webapp). In order to test the app you need to start Grails by typing

*Grails*

into the command line and then begin running the site by typing

*run-app*

into the command line once Grails has started. This will allow you to open a browser and navigate to the provided URL and begin testing the site. If any problems are experienced at this stage it is helpful to check that the web server is configured to use this folder as the location for web pages. If you wish you can instead clone the repository to the public\_html folder in the /home/\*USER directory.

If you wish to compile the site for deployment you must enter

*war*

Into the Grails prompt, this command creates a web application archive which can be located on a server. Run-app should never be used for a completed application; run-app is purely for development purposes.

Notes regarding my implementation:

I restarted this project after making an early mistake. Owing to the speed at which I repeated the initial work the first four tasks were all committed as one change.

I also named my project ‘webapp’ instead of ‘placement’.

The Apply controller, the Apply View and form and the Opportunity domain model are commented. The bootstrap file is also commented.

# Work undertaken for each of the tasks.

Set up a new GIT Project

In order to create a new project on Github I logged in to the website and clicked the ‘new repository’ button. From there I filled in the form asking for the project’s name, an optional description and a homepage for the project. Once this form was completed I made sure the repository was public and clicked the ‘create repository’ button. I then created an empty folder in my home directory and changed to that folder using the command line. Once the directory had changed to my empty folder I entered ‘git init’ to create a blank local repository in this location. I then entered ‘gedit README’ into the command line and used gedit to write a readme for my project. Once I had finished this file I then closed gedit and entered ‘git add README’ into my command line to add that file to my commit. I then committed this file to my local repository by typing ‘git commit’ into the command line. The commit function supports a number of optional arguments such as ‘-m’, this argument allows you to add a commit message to the commit, this is used to explain the purpose and contents of the commit. Once the commit had been submitted I then typed ‘git remote add origin [git@github.com:generic-gamer/WebArc3.git](mailto:git@github.com:generic-gamer/WebArc3.git)’ to link my local repository to the repository created on Github.com and entered ‘git push –u origin master’ to upload the contents of my local repository to the Github server.

Create a clean empty grails application called placements

In order to create an empty Grails application I entered ‘grails’ into the command line to open Grails and once it had opened I entered ‘create-app webapp’ to create an empty application in this directory. The application is created inside a folder with the name specified above. At this stage it is important to note that I accidentally named my project ‘webapp’ instead of ‘placements’ as the brief specified. This does not affect the functionality of the program, except to mean that all URL generations will have ‘webapp’ in place of ‘placements’.

Convert the supplied business model into a set of concrete domain classes realised as grails domain classes.

In order to create the needed domain classes I changed directory to the project’s root directory and ran Grails. I then ran ‘create-domain-class org.webapp.Student’ to create the Student domain class. I repeated this process with each of the other three classes needed, substituting their name for ‘Student’ in the script. Once this was completed I changed directory to ‘grails-app/domain/org/webapp’ and opened each in turn in Gedit. I examined the files in turn and added the attributes specified in the brief. Entering the attributes was accomplished by inserting them between the braces following the class name; they took the form of an upper case word describing the data type and a lower case word to act as the attribute name.

***package*** *org.webapp*

*class Student {*

*String coursecode*

*String name*

*String notes*

***static*** *constraints = { }}*

After specifying the attributes for each class I then created the relationships between classes. I added these relationships using the following syntax.

***package*** *org.webapp*

*class Student {*

*Static hasMany = [Applications: Application]*

*Static mappedBy = [Application: “opportunity”]*

*String coursecode*

*String name*

*String notes*

***static*** *constraints = { }}*

This script states that a student has many applications and that an application is referred to as an ‘opportunity’. The corresponding syntax in the owned class is below.

*Static belongsTo = [Student: Student]*

This class also has the student linked to the instance as an attribute to act as a foreign key. This is entered in the same way as previous attributes but the class name is substituted for a data type.

*Student name*

Demonstrate the basic scaffold workflow by creating all scaffolds (Controller and

View) for Student, Placement Opportunity and Status

These scaffolds were automatically generated by returning to the Grails root directory, opening Grails and passing it the ‘generate-all’ command. This command creates views, necessary forms for the views and controllers for every domain in the project. ‘Generate’ differs from the syntactically identical ‘create’ command by generating all necessary routing instructions, methods and forms whereas ‘create-all’ simply creates empty files for all needed components.

At this stage I encountered a problem. Grails uses a specific layout for its’ naming scheme, it uses a lowercase word first with camel-case words concatenated to add multiple words. For example ‘name’ would be name but ‘student name’ would be studentName. I had initially created my domains with upper case attribute names and I was forced to review my naming scheme as Grails would not read the files correctly. I asked Ian for help with this problem and he explained the naming schema to me.

Demonstrate the modification of scaffolds by improving, in any way you see fit from

textual changes to improving reference data, the generated scaffold

I changed several features on the index page; removing the standard welcome paragraph, changing the page to make it refer to controllers by their name rather than their path and changing several of the heading boxes on this page. In the Student controller and view I changed ‘list’ so that it would display students by name and so that it would display their details in an order more suitable for searching by name.

Use the applications BootStrap class to configure default status codes

In order to change the bootstrap class I navigated to the file ‘grails-app/conf/BootStrap.groovy’ and opened it in gedit. I added seven lines to the file, this file loads test data into the domains at startup and I wanted to populate Status automatically. Each of those lines was structured identically to the example below.

*New Status(code: “Applied”, description: “Candidate applied for placement.”).save(failOnError: true)*

This enters a new Status entry with the attributed specified in the brackets and then saves it, it is further specified that failure to perform this action will result in the program failing to load.

Use your scaffold interfaces to create test students and placements

Using my scaffolded pages I added a test student and a test placement to my application. These entities are destroyed at the end of a testing session.

Create a new controller called "Apply" the controller should have 2 actions "index"

which shows a HTTP form, and a http-GET based "processApplication" which reads

Student and Job ID's from the form, creates a new Application and sets its status to

"Applied", and it's timestamp to now.

In order to create this controller and view I navigated to the app’s root directory, opened Grails and typed ‘create-controller org.webapp.Apply’ and ‘create-view org.webapp.Apply’ into the command line. This then created the Apply view and controller with no code inside them. Up to this point I had been using a pre-made scaffold to test Application functionality, I now deleted those scaffolds and constructed my new view. I created a separate form to allow it to be reused between views, this was not necessary but it allowed for future expansion. I then created two methods in the controller called ‘Apply’ and ‘processApplication’, the syntax for that is:

Def Apply(){}

I then created a default action of a new application form using the code:

[applicationInstance: new Application(params)]

This allowed me to pass values to the creator if I wished, but its’ main function was to create a new instance of Application.

Afterwards I created a process method using the code commented in my application. This code would read the parameters from the above form and use them to create a new application with those parameters. The operation also selected a preset status for the application.

At this stage I was supposed to also set the controller to add a datestamp automatically but I could not find out how to do this. As a compromise and to preserve usability I edited the Application controller to change the name of the date value to ‘dateCreated’. Grails automatically adds a date to an attribute with this name.

Create a new method on your placement opportunity controller called

"listOpenPlacements" which only lists placements who's status is "OPEN" and returns

XML or JSON data based on content negotiation.

In order to carry out this stage I opened the OpportunityController file and added a new method as above. In this method I defined a variable, set the variable as the result of a find request (specifically a ‘findAll’ request) that would search for every Opportunity instance with a status of ‘%pen’. The percent sign is a wildcard and stands in for any other character, this allows the search to return ‘Open’ and ‘open’. I then instructed the controller to supply the requested data in either extensible mark-up language (XML) or in Javascript Object Notation (JSON) depending on which format was requested by the browser.

Unfortunately I ran into an error whilst implementing this stage that I was unable to resolve. The syntax and grammar for the content negotiation were copied from the Grails user manual but the parser reported an error when trying to read those lines. Numerous capitalisations were attempted but the error was not

Create a new method on your placement opportunity controller called "listApplicants"

this method should accept a placement opportunity ID and return either a JSON or an

XML document based on content type negotiation.

In order to carry out this stage I added another method called ‘listApplicants’ that would return all applicants for a job. In order to add this functionality I entered a find request that would return all Applications with an Opportunity ID matching the number supplied in the URL. This information would then be supplied in the format requested in the URL, the choice offered is of XML or JSON.

This task met with the same unidentified error as the previous one and as such the functionality proved impossible to implement.

# Personal reflection.

I believe that my performance on this project was of a suitable quality for university work but I do have some actions and approaches I would review before undertaking such a project again. I felt that my level of effort was sufficient but that my lack of attendance at the original lessons and my general weakness at web coding left room for improvement. Whilst my lack of attendance was due to legitimate illness rather than laziness it is undeniable that it set me back, when the project was issued I had to learn concepts I should already have known. Web coding is my weakest skill that I have identified; it is something I will need to improve before it reaches a useful level.

I am however confident that my work is of an acceptable standard despite my lack of understanding at the outset of the project. I am satisfied with the functioning of my website and on my drastically improved knowledge of how Grails operates. I also have a far firmer grasp of the Model-View-Controller paradigm than I did before the start of the project.

If I were to repeat this exercise I would give greater importance to understanding the project at the beginning and more actively asking for assistance.