Team

- Classwork
  - 01 Assignment - CS in the Wild
  - Class Project
  - Unit 1 - Web Application
    - Unit 1 Assignments and Exercises
      - 02 Assignment - Pick a Product and Name for Your Store
      - 03 Assignment - Groovy Exercises
      - 04 Assignment - Setup Grails and IDEA
      - 05 Assignment - Domain Classes
      - 06 Assignment - Controller Scaffolding
      - 07 Assignment - Mockup
      - 07a Assignment - Balsamiq Tutorial
      - 08 Assignment - HTML
      - 09 Assignment - Your Own Store
      - 10 Assignment - Bootstrap Your Site
      - 11 Quiz - Introduction to SQL
      - 12 Assignment - Manufacturer Report
      - 13 Assignment - UX or Styling - Extra Credit
      - 14 Test - Web Applications
      - Exercise 02 - Classes and Instances
      - Exercise 03 - Domain Tests
  - Unit 2 - Modern Application
    - 15 - Version Control
    - 16 - Team Store
    - 16a - Team Store Redux
    - 17 - Services and Postman
  - Unit 3 - Mobile App
    - 18a - Mobile App Mockup
    - 18b - Adjust Postman Tests to Use AWS Server
    - 18c - Mobile Store Skeleton
    - 18d - Adding Categories, Manufacturers and Sort to the App
    - 18e - Adding Login
    - App Requirements
    - React Native Setup
- How-to articles
  - Bootstrapping
  - Clone Repository to disk from command line
  - Create a Controller
  - Create a Grails Domain Class
  - Create a Grails Project
  - Enable Image Upload for Products
  - Forwarding Schilling Emails
  - Import a Project from a Zip File
  - Installing Grails on windows
  - Running Tests
  - Take a Screenshot
  - Using the Application UI Debug Console and Database Console to View and Update Data
  - Working with Grails Domain Classes
- Implement Catgory Report
- Product requirements
- Resources
- Roadmap
- Scrum
- Setup
  - Setup Balsamiq
  - Setup Grails
  - Setup IntelliJ IDEA
  - Setup Lucidchart
  - Setup Status

Classwork

- 01 Assignment - CS in the Wild
- Class Project
- Unit 1 - Web Application
- Unit 2 - Modern Application
- Unit 3 - Mobile App

01 Assignment - CS in the Wild

Where do we find computers and software in our every day lives and what is good and/or bad about it?

Give a presentation about computers and software used in the world. This can be anything from a watch to a radio telescope.

From a computer mouse to a mainframe.

From a website to the Large Hadron Collider.

The presentation can be a simple discussion from notes to a powerpoint presentation to a demonstration of some kind.

Come prepared to tell us where the hardware and software is found and why this use of computer science is a good thing or a bad thing.

Pick a team of two or go it alone, its up to you.

Every other week on Thursday we'll have a presentation.

We will be working in "Sprints" of 2 weeks each. This comes from the Agile "Scrum" methodology.

At the end of each sprint we do a showcase of what we've accomplished that sprint.

This will be part of the showcase in the beginning.

Pick a subject and give a simple description. You can put down a sprint if you want. If not, Mr. Sayers will pick it for you.

Subject	Description	Links	Presenter(s)	Sprint	Approved
Guitars	Guitarists use a lot of technology these days.	Presentation Youtube Video	Mr. Sayers	1	23 Aug 2017
Eye Glasses			Daniel Frank	2	07 Sep 2017
Special Effects	Computer generated effects are cheaper, and often on par with practical effects		Ketu Sayers	TBD	
PLCs	What PLCs are, how they work, and what they do.	I will email you the completed file.	Emmett Looman	4	
Automated Cars			Alex F		
Smart Watches			Deryk F		
Youtube AI			Peter H		
Formula 1 25 Jan 2018	People drive faster and crash less when they're on a computer.	Presentation	Parker Siegfried		
Video Games 27 Feb 2018			Sean S		
08 Mar 2018			Sami S		
19 Apr 2018			William B		
			Ryan W		

Details

Your presentation should include the following sections:

A description of the context or problem being solved.

A description of the solution.

Some details of the hardware and / or software that provides the solution.

Extra commentary is welcome.

This should fit into a 5 to 10 minute presentation time.

Be sure to answer these questions:

- As a Computer Scientist
    - What software tools will I have to know?
    - What programming languages will I need to know?
    - What hardware will I be working on?

Class Project

For our project, we'll build an online store.

We need to decide what we will sell.

To make changes to this page, click the pencil to start editing.

Click in a table, then use one of the tools above to add a row.

Click publish when you are done.

Here are the options.

What we sell	Who	Votes
Schilling things (shirts, jackets, etc)		
inspirational cat posters	Bryn	
school supplies	Bryn	
Computer Peripherals		
Fidget Spinners	Sean	
Glasses	Daniel	
Bulk ANYTHING	Pete	
IT support	William	
Aquaman Loot	Nathan	
RC Helicopters/Cars + Replacement Parts	Deryk	
Bottles of water bottles	Parker	
Fencing Equipment	Ryan	
Drones and drone equipment	Emmett	

We need a name for our store.

Here are the options:

Name	Who	Votes
SDCI	Sami	
inspirationalcatposters.com	Bryn	
School Suppliers	Bryn	
Schilling Spinners Inc.	Sean	
Smart People llc		
Schilling Shop	Sami	
EyeCanSeeClearly Inc	Daniel	
MassIt.com	Pete	

Schilling Computing Inc. (SCI)	William	
Aquaman's Loot chest	Nathan	
Mech Tech	Deryk	
Cosmic Radio Bursts from Space	Parker	
Droning On Inc.	Emmett	

Unit 1 - Web Application

- Unit 1 Assignments and Exercises

Unit 1 Assignments and Exercises

- 02 Assignment - Pick a Product and Name for Your Store
- 03 Assignment - Groovy Exercises
- 04 Assignment - Setup Grails and IDEA
- 05 Assignment - Domain Classes
- 06 Assignment - Controller Scaffolding
- 07 Assignment - Mockup
- 07a Assignment - Balsamiq Tutorial
- 08 Assignment - HTML
- 09 Assignment - Your Own Store
- 10 Assignment - Bootstrap Your Site
- 11 Quiz - Introduction to SQL
- 12 Assignment - Manufacturer Report
- 13 Assignment - UX or Styling - Extra Credit
- 14 Test - Web Applications
- Exercise 02 - Classes and Instances
- Exercise 03 - Domain Tests

02 Assignment - Pick a Product and Name for Your Store

Fill out the row on the class project page for your store.

At the end of the quarter we'll pick on and use it as the class project going forward.

Due 31 Aug 2017

03 Assignment - Groovy Exercises

File	Modified
Exercise01b.groovy	Aug 27, 2017 by Ken Sayers
Exercise01a.groovy	Aug 28, 2017 by Stu Schilling

Drag and drop to upload or browse for files

Download All

04 Assignment - Setup Grails and IDEA

Follow the instructions available for Grails, IDEA and Lucidchart.

This will be required to finish the next assignment.

05 Assignment - Domain Classes

We will define the domain classes for our store.

There are a lot of things we might do with our domain. Anything outside the classes defined here can be used as extra credit.

Use the diagram below as a guide for developing the domain classes.

Follow these instructions to create a domain class.

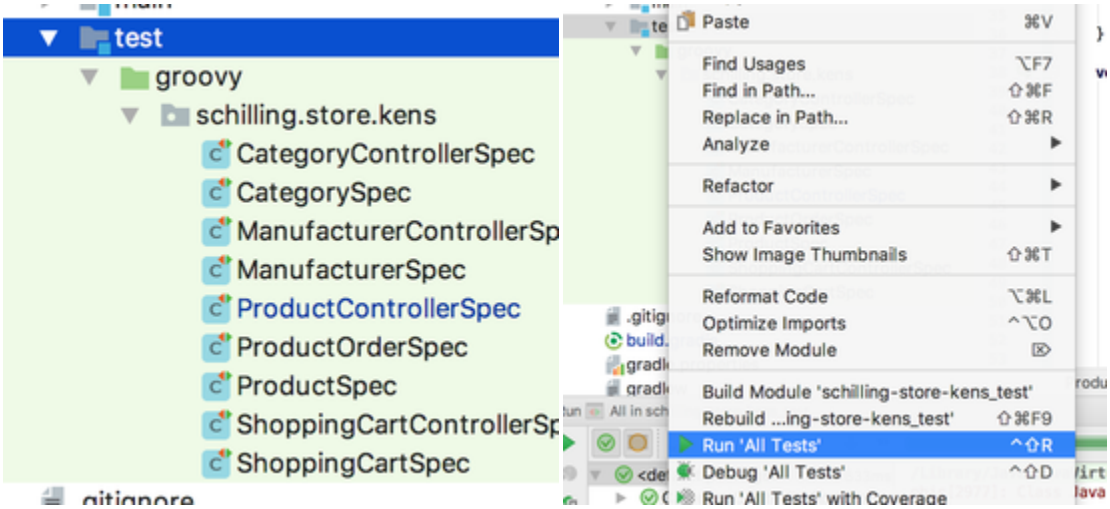
```
package schilling.store.kens

class Product {

    static constraints = {
    }
}
```

and should look like this when you are finished.

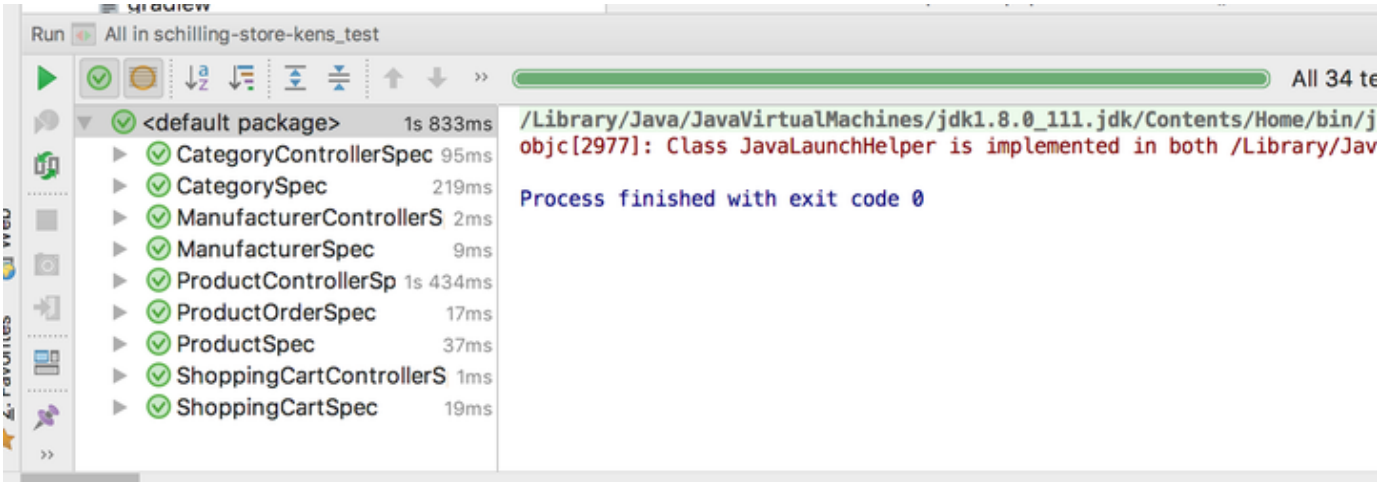
```
package schilling.store.kens

class Product {

    String name
    String description
    Manufacturer manufacturer
    Category category
    Double price

    static constraints = {
        name nullable:false, blank: false
        description nullable:false, blank: false
        manufacturer nullable:false
        category nullable:false
        price nullable:false
    }

    String toString() {
        "$name by $manufacturer price: $price"
    }
}
```

Run the attached tests to make sure your classes are implemented correctly.

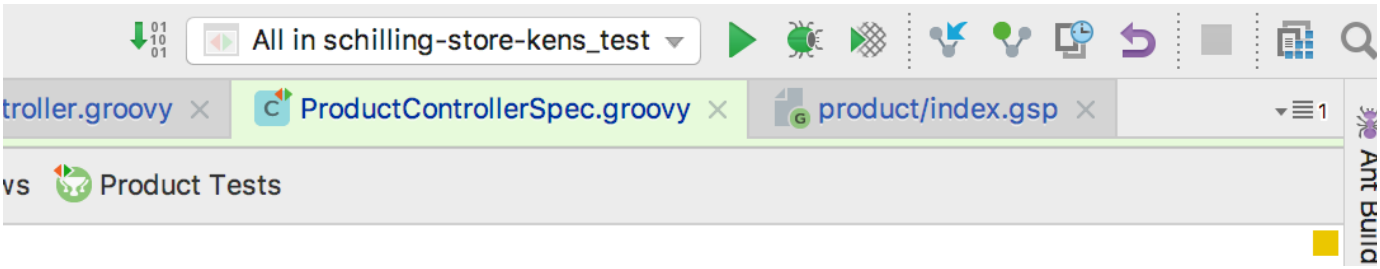
- Copy them into src/test/groovy/...package... folder.
- Run the tests



- You should see the green bar when it all works



- You can rerun the tests easily with the short cut in the top information bar:



We will go over this in class on thursday one at a time. You should be able to show your tests running successfully.

06 Assignment - Controller Scaffolding

For this assignment, we put scaffolded controllers into our application.

We then open the application and show that the links are generated onto the home page.

Further, we can walk around in our application to show that the controllers are working.

See this page for details on how to create controllers.

07 Assignment - Mockup

Mockup the UI for the store using the Balsamiq tool.

Install the tool with instructions from here: Setup Balsamiq

Check out the resources for Balsamiq here: Resources especially good is https://support.balsamiq.com/tutorials/bootstrap/

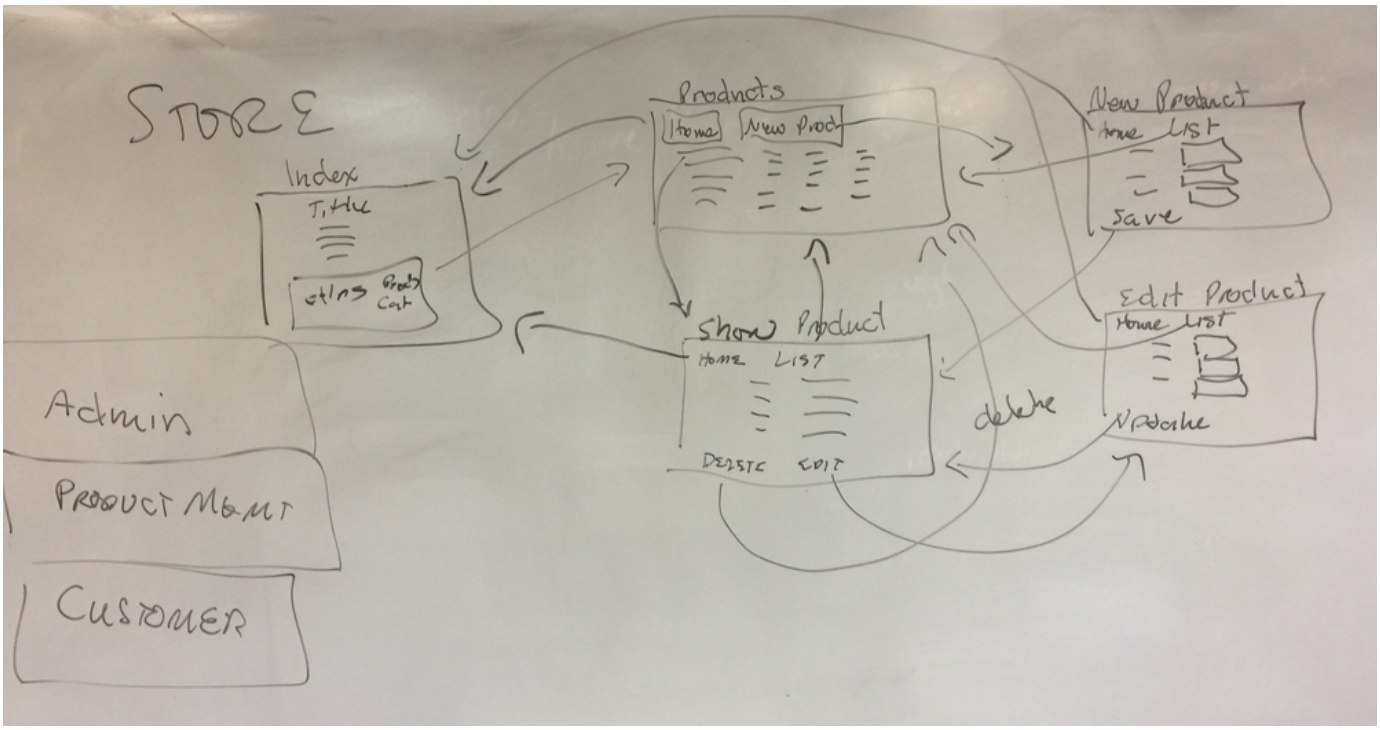
Once you have Balsamiq, create a mockup of the application with the flow described in the diagram below.

Links should be fully functional between pages.

Create a single project in balsamiq with a different mockup for each page.

Save your project to a file and upload that file to thinkwave. This should be a "bmpr" file so I can run it on my machine.

The fields on the show page should not be active. Look for a state property to disable input controls or use labels.



User Interface

Mockup follows the diagram above.

07a Assignment - Balsamiq Tutorial

Follow the tutorial found here: https://support.balsamiq.com/tutorials/firstmockup/

Post the resulting image (.png) file to thinkwave.

08 Assignment - HTML

Implement the mockup in html.

Here are samples of show and update pages.

show.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Schilling Store</title>
</head>
<body>
    <h1>Welcome to the Schilling Store!</h1>
    <hr/>
    <a href="../index.html">Home</a> | <a
href="../product/list.html">List</a>
    <form action="../product/update.html">
        <fieldset>
            <legend>Product Details</legend>
            <table>
                <tr><th>Name</th><td>Product 1</td></tr>
                <tr><th>Description</th><td>This is my product</td></tr>
                <tr><th>Price</th><td>$35.00</td></tr>
                <tr><th>Manufacturer</th><td>Manny 1</td></tr>
                <tr><th>Category</th><td>Catty 1</td></tr>
            </table>
        </fieldset>
        <button type="submit">Edit</button>
    </form>
</body>
</html>
```

update.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Schilling Store</title>
</head>
<body>
    <h1>Welcome to the Schilling Store!</h1>
    <hr/>
    <a href="../index.html">Home</a> | <a
href="../product/list.html">List</a>
    <form action="../product/show.html">
        <fieldset>
            <legend>Product Details</legend>
            <table>
                <tr><th>Name</th><td><input type="text"/></td></tr>
                <tr><th>Description</th><td><input
type="textarea"/></td></tr>
                <tr><th>Price</th><td><input type="text"/></td></tr>
                <tr><th>Manufacturer</th><td><select><option>Manny
1</option><option>Manny2</option></select></td></tr>
                <tr><th>Category</th><td><select><option>Cat
1</option><option>Cat 2</option></select></td></tr>
            </table>
        </fieldset>
        <button type="submit">Update</button>
    </form>
</body>
</html>
```

09 Assignment - Your Own Store

Go back and look at the kind of product you wanted to sell.

What kind of information do you need to capture about that product, that you might not need for other kinds of products.

Create a new project called ***schilling-store-scaffolded-<username>*** where username is your schilling email username.

Create all the domain classes in your project. Add your fields to the Product class. You must have at least one new field that was not in our project before.

Create the tests and make sure you add tests for your new fields. There must be at least two new tests for each field (valid and invalid).

Create scaffolded controllers for the domain classes.

Take screenshots of your passing tests and the controllers on the index.html page.

This assignment is worth 50 points.

10 Assignment - Bootstrap Your Site

In this assignment, we add bootstrap as a css library for our site.

This will give it a modern look and feel.

Download the Bootstrap Template found here

Unzip to some location on your hard drive.

Open the folder as a project in IDEA.

Follow the Bootstrap Tutorial also listed here

Rework all your html to use bootstrap.

File	Modified
create.html	Nov 09, 2017 by Ken Sayers
list.html	Nov 09, 2017 by Ken Sayers
show.html	Nov 09, 2017 by Ken Sayers
update.html	Nov 09, 2017 by Ken Sayers
index.html	Nov 09, 2017 by Ken Sayers

Drag and drop to upload or browse for files

Download All

11 Quiz - Introduction to SQL

This quiz will cover the information on the SQL tutorial up until before Aliases and Joins.

You can take the quiz at this link until the end of the day on Nov 30.

https://goo.gl/forms/SpzdJeYJPvWI1EC23

The questions all pertain to the data inside the tutorial.

Note: you can try your sql statements (even insert, update and delete) on the data in the tutorial.

If you mess up the data, just restore the data with the button on the right side.

You can use the try it yourself in the tutorial to work through the questions here.

NOTE: the quiz questions only work with the first 9 records in the table.

To simulate this in the tutorial add this to your where clauses (CustomerID < 10).

For example, to see the table shown in the quiz you can use this statement:

```
SELECT * from Customers where CustomerID < 10
```

12 Assignment - Manufacturer Report

Following the pattern established for the Category report here, create a report for manufacturers.

Upload the screen shot of the final report.

13 Assignment - UX or Styling - Extra Credit

Update the store in whatever way you like to add your own touch.

Use CSS or HTML or navigation to improve the User Experience.

Share the changes you make and the screenshots of how the changes have affected the UX.

14 Test - Web Applications

Complete the test found here: https://goo.gl/forms/HiUHIejYSkYVtRts1

You can resubmit if you like. I will take your highest score.

Exercise 02 - Classes and Instances

This exercise shows how classes differ from instances.

We also see a little collections love.

Exercise 02 Code

```
class Frame {
    String color = 'Natural'
    Double height = 8
    Double width = 4

    def setHeight(Double height) {
        throw new UnsupportedOperationException("Cannot change height")
    }

    def setWidth(Double width) {
        throw new UnsupportedOperationException("Cannot change width")
    }

    String toString() {
        "$color $height x $width"
    }
}

def frames = []
frames.add new Frame()
frames.add new Frame()

frames.each { frame ->
    println frame
}

// uncomment the following lines one at a time and execute the script
// frames[0].color = 'Red'
// frames[1].height = 9.0

frames.each { frame ->
    println frame
}

// make it possible to customize the frame with a message or title
```

Exercise 03 - Domain Tests

See the attached files. They are tests that should be placed into the ***src/test/groovy/<your package>*** folder of your store project.

You will see that the file should already be there when you copy them. Just overwrite them. The files were created when you created your domain classes.

They should all pass when you are done with your domain classes.

File	Modified
CategorySpec.groovy	Sep 09, 2017 by Ken Sayers

ManufacturerSpec.groovy Sep 09, 2017 by Ken Sayers

ProductOrderSpec.groovy Sep 09, 2017 by Ken Sayers

ProductSpec.groovy Sep 09, 2017 by Ken Sayers

Drag and drop to upload or browse for files

Download All

Unit 2 - Modern Application

15 - Version Control

Next to your name put your github user name.

Teams:

	Team 1	Team 2	Team 3	Team 4
Github Team	Alex-Deryk	Ketu-Sami	William-Emmett	Sean-Parker
Github Repository	Alex-Deryk-01	Ketu-Sami-01	William-Emmett-01	Sean-Parker-01
Member	Alex - fianial	Ketu - Geekman9097	William - Vivamun	Sean - SeanSwayze
Member	Deryk - TripleDebt	Sami - smithsami	Emmett - RandomPerson24	Parker - riddlydiddly

Instructions:

Split into four teams of two each.

The team must be named a combination of your first names like this: jack-jill-joan-feature

Install your student version of github account, from here: https://education.github.com/pack

Using the command line follow the scenario below:

```
git clone https://github.com/SchillingCSP/<github repository for your team 01>
cd into the repo with something like "cd Sean-Parker-01"
echo "# vcs-assignment" >> README.md
git add README.md
git commit -m "first commit"
git push -u origin master
```

Each member of the team creates a branch. The branch should be named for your first name.

Add an initial file to the branch. It should be called ***<youe nane>.txt.*** It must have some text in it. You could use "placeholder text" if you like. (See the echo command above)

Commit the branch and push it to the shared repository.

Each member of the team should then pull and check out the branch from the other team member.

Then each should add their name to the <other member>.txt file

Then each should commit to their local repository.

Next, push to the shared repository.

You probably won't be able to push without any extra steps.

If you can't push, you will get an error trying to push, because you will have to pull first.

Extra steps for second and third pushes.

First, commit your changes

Then pull from the repository.

Merge the changes.

Commit the merge.

Push the commit.

Everyone, should then pull from the repository and see all files and changes to all txt files.

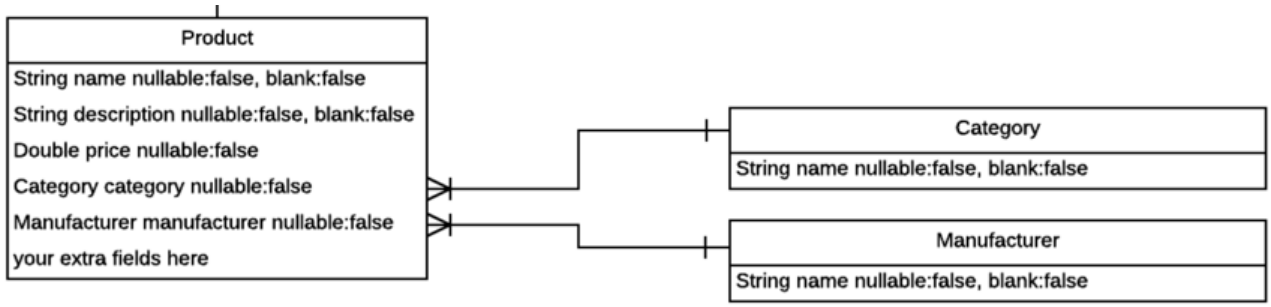
At the end of the assignment, there should be at least the two files in the repository and each should have both name in it.

16 - Team Store

Overview

The four teams defined for the previous assignment will be used for this project.

Each team will select a portion of the project to complete.



There will be a single repository. Clone with this URL: https://github.com/SchillingCSP/team-store.git

The four teams are:

Component	Responsibility	Owner (Team name)
Core	Initialize the grails project. Styling. Bootstrapping.	Sami-Ketu
Product	Product domain, controller and views including a report	William-Emmett
Manufacturer	Manufacturer domain, controller and views including a report	Sean-Parker
Category	Category domain, controller and views including a report	Alex-Deryk

Use branches to handle sharing of code without breaking other teams.

There will be dependencies between the different teams. Product relies on manufacturer and categories for example.

One member of each team should take the domain and the other the controller and views.

This will require coordination in the team using a branch.

The core team should split the work by creation of the grails project separate from the styling and bootstrapping.

You should have several points where you synchronize code between teams with a push and pull.

- After the core team has created the grails project
- After each team has created their domains (category and manufacturer first)
- After each team creates their controllers and views.

Each person should have the latest fully running version of the application on their machine at the end of the project.

This will require a quick demonstration to me on the due date.

Assessment

40 Points total

15 points to everyone for getting the site working.

15 points for each team for completion of their portion of the project.

10 to each individual for having the working system on their machine.

Expectations

Every person must contribute at least one file to the project.

The end result should have the same functionality as the individual stores we have created:

- three domains - Product, Category, Manufacturer
- controllers and views for each of the domains
- some sample data for startup
- some special styling of the html and/or css

NOTES

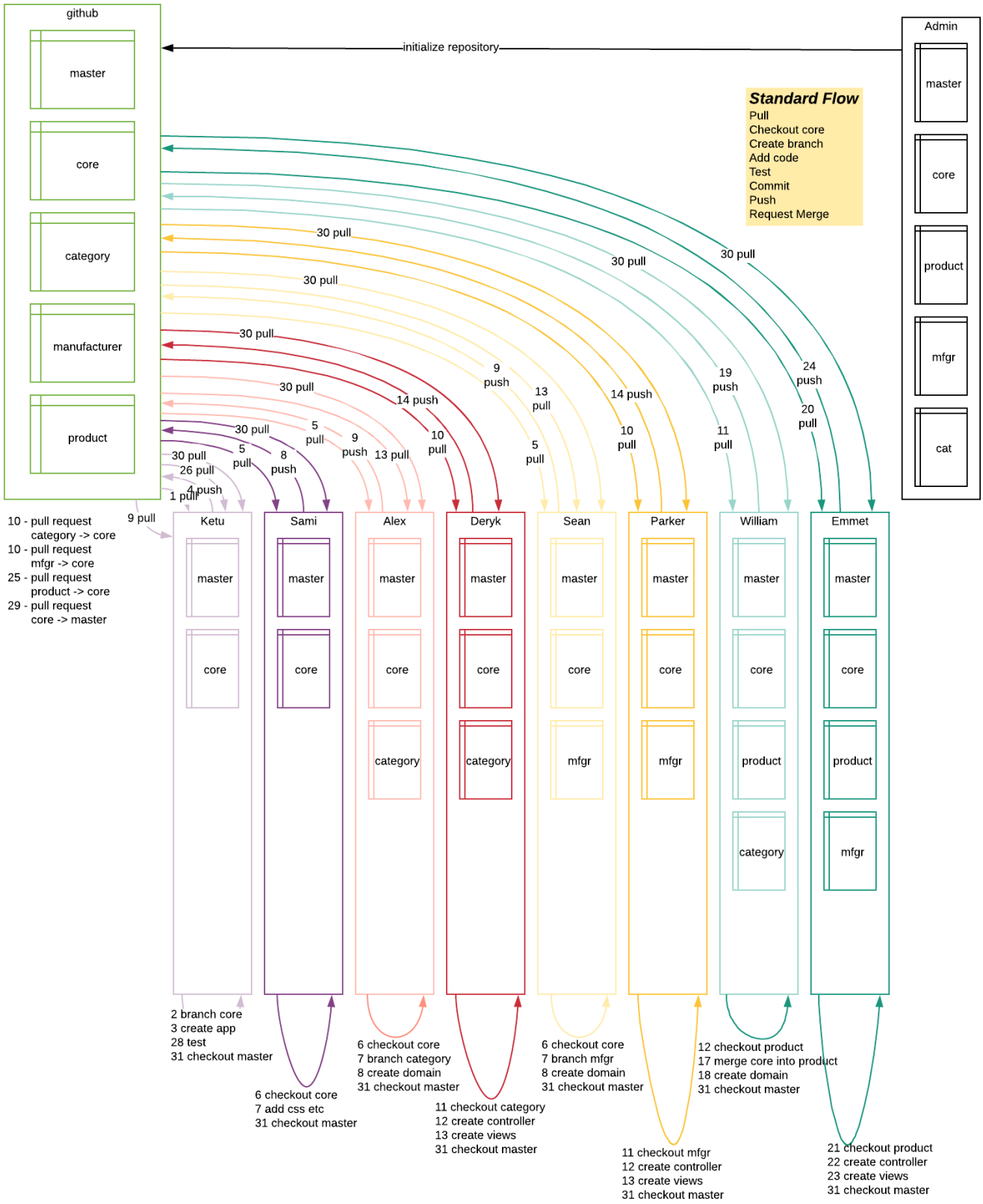
Emmett- Note To William: I added a working domain class to the William-Emmett branch. I finally got IntelliJ to work by using this link: https://www.jetbrains.com/help/idea/creating-grails-application-from-existing-code.html

16a - Team Store Redux

The assignment is two parts.

1. As a team, create the team-store-redux using that repository from git hub. Everyone should end up with the complete app working with all functionality. (25 points)
2. Fill out the table below (use the link and save as a file that can be uploaded to thinkwave) (25 points)

Person	Discord
Sean	EpicSean314#8119
Ketu	Geekman9097#0663
Deryk	derikpie5342hick@gmail.com
Emmett	elooman@schillingschool.org
Alex	fianial#8741
William	DMWCincy#7087



Update the following table with what is in each branch or environment after the associated step. Each person is responsible for the entire table, so create your own child page and fill out the table.

	GitHub					Local							
Step #	**master**	**core**	**category**	**manufacturer**	**product**	**Ketu**	**Sami**	**Alex**	**Deryk**	**Sean**	**Parker**	**William**	**Emmett**

0	readme												
1						readme							
2						readme branch: core							
3						grails-app							
4		grails-app											
5													
6													
7													
8													

Link to google doc.

https://docs.google.com/spreadsheets/d/17-yRIJwrRggcL7QlZdSAjvuTx-5DDk7Ye_oDMZ_sicg/edit?ts=5a9568a0#gid=0

Make a copy by save as.

17 - Services and Postman

This assignment is all about web services and how we can test them using postman.

Background

Web Services Architecture

Web services are used to connect to data or functionality from a different organization or in a different technology.

In our case, we will be writing a mobile application that will use our grails system to manage the data.

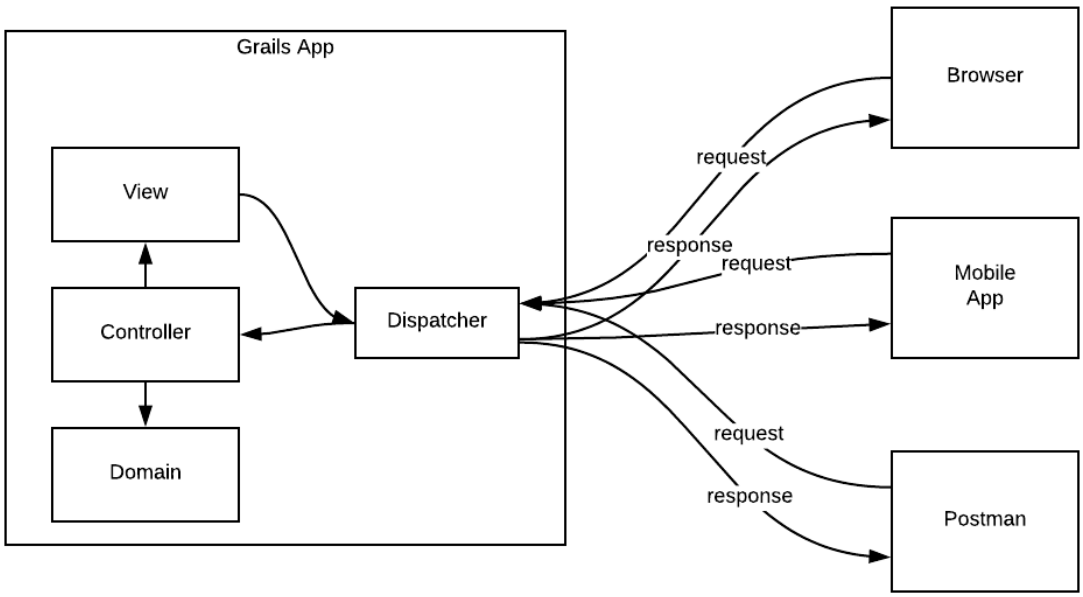
The mobile application is in a totally different technology (javascript).

We will connect the mobile application to our grails through web services.

This diagram shows how we connect a mobile application to our application, where the request goes and how it comes back.

This is shown for the regular grails html ui and for the mobile application.

We will use postman to test our services.



Browser Flow for category list
- briwser sends request **GET *http://localhost:8080/category/index***
- dispatcher calls ***index*** on ***CategoryController***
- controller gets Category.list and responds with ***/views/category/index.gsp***
- browser shows html

Browser Flow for category save
- browser sends request **POST *http://localhost:8080/category/save***
- browser provides body {id:null,name:'Toys}
- dispatcher calls ***save*** on ***CategoryController***
- controller creates category 1 and saves it in the database
- controller redirects to the show method

Mobile App Flow for category list
- app sends request **GET *http://localhost:8080/category/index.json***
- dispatcher calls ***index*** on ***CategoryController***
- controller gets Category list and responds with ***/views/category/index.gson***
- no such view is found so it renders with default format and returns json [{id:1,name:'One'},...]
- App creates list and displays it with category information

Mobil App Flow for category save
- same as browser except with .json on request and different view to show json results

Postman will simulate what the mobile app requests and receives

What is in a request and response?

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

- A Request-line
- Zero or more header (General|Request|Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

Request

Item	Description	Example
URI	Universal Resource Identifier	localhost:8080/category/index

Method	The request **method** indicates the method to be performed on the resource identified by the given **Request-URI**	GET, PUT, POST, DELETE
Headers	The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.	Content-Type=application/json
Body	GET does not require a body. A POST will include a body.	{id:1, name:'One'}

Example GET Request Message

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Example POST Request Message

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

After receiving and interpreting a request message, a server responds with an HTTP response message:

- A Status-line
- Zero or more header (General|Response|Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

Item	Description	Example
HTTP Version	A server supporting HTTP version 1.1 will return the following version information: `HTTP-Version = HTTP/1.1`	HTTP/1.1
Status Code	The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit: **1xx: Informational** It means the request was received and the process is continuing. **2xx: Success** It means the action was successfully received, understood, and accepted. **3xx: Redirection** It means further action must be taken in order to complete the request. **4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled. **5xx: Server Error** It means the server failed to fulfill an apparently valid request.	200: success 201: created 500: internal server error

Headers	The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.	Content-Type application/json;charset=UTF-8 Date Sun, 11 Mar 2018 22:45:56 GMT Transfer-Encoding chunked X-Application-Context application:development

Body	The result of making the request. This is html for browsers and json or xml for web services.	`<html>`
		`<body>`
		`<h1>Hello, World!</h1>`
		`</body>`
		`</html>`
		`json example`
		`[`
		`{`
		`"id": 1,`
		`"product": {`
		`"id": 1`
		`},`
		`"quantity": 2,`
		`"number": "3"`
		`},`
		`{`
		`"id": 2,`
		`"product": {`
		`"id": 1`
		`},`
		`"quantity": 2,`
		`"number": "3"`
		`}`
		`]`

Assignment

You should download postman.

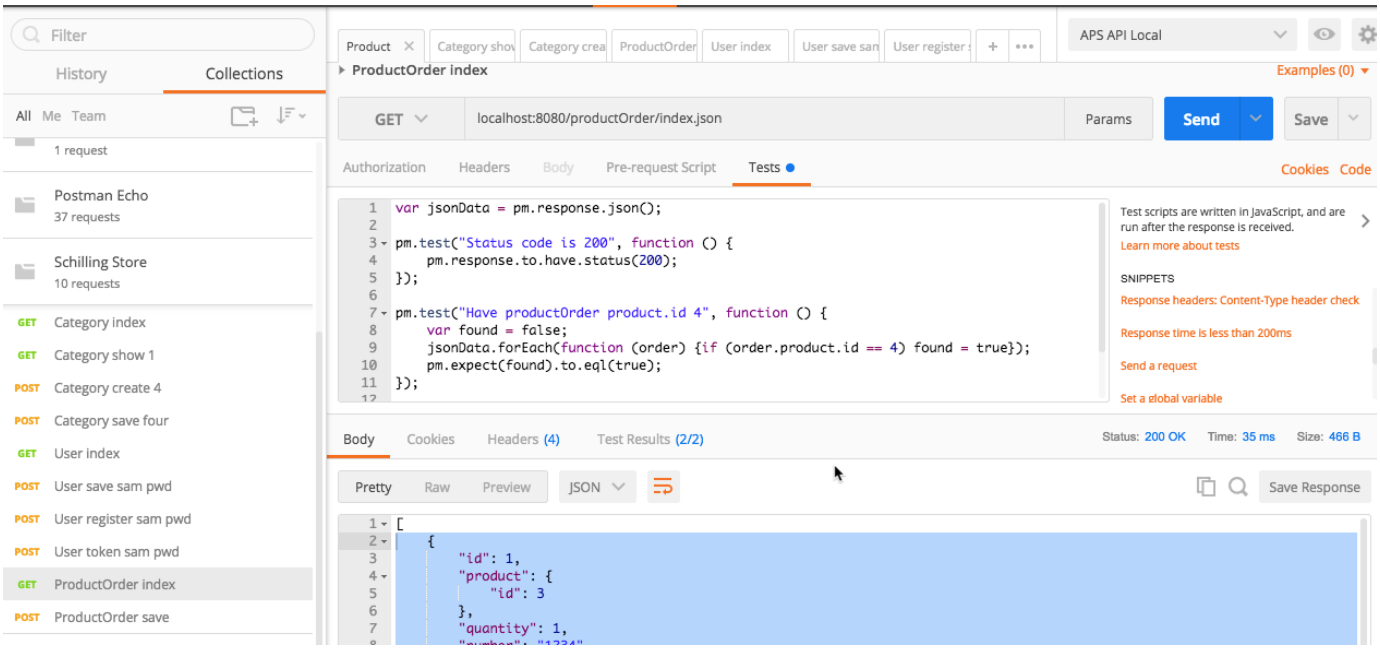
We will use git to share our postman scripts with each other.

The team-store-redux project has ProductOrder added to it. They should exist in the Core branch by the time you read this.

If you check out Core, you will get postman tests in src/test/postman.

If you import this file into postman and run the app, then you should be able to run these tests in postman from the runner.

Here is a sample postman request for ProductOrder



For this assignment, you will create postman tests for your parts of the team store and share them with the rest of the teams.

Create tests to list the objects, show an object, create a new object and save an object.

Each test should include an assertion for the response code and some expected data in the response.

The assignment is complete when merged into Core and can be run by all others in the class.

Unit 3 - Mobile App

18a - Mobile App Mockup

Using Balsamiq, create a mockup of your store app.

It should include at least three pages as that is the minimum viable product.

Main page which can have an initial list of products. This should include a filter.

Detail page for a product.

Purchase page for a single or multiple products.

Attached is my mockup of the app. It is just a beginning. You can use this as a start for your mockup or create one from scratch.

Your mockup should have navigation between the pages.

Here is a link to some instruction on creating mobile app mockups: https://support.balsamiq.com/tutorials/mobileapplication/

File	Modified
Kens Mobile Mockup.bmpr	Apr 07, 2018 by Ken Sayers

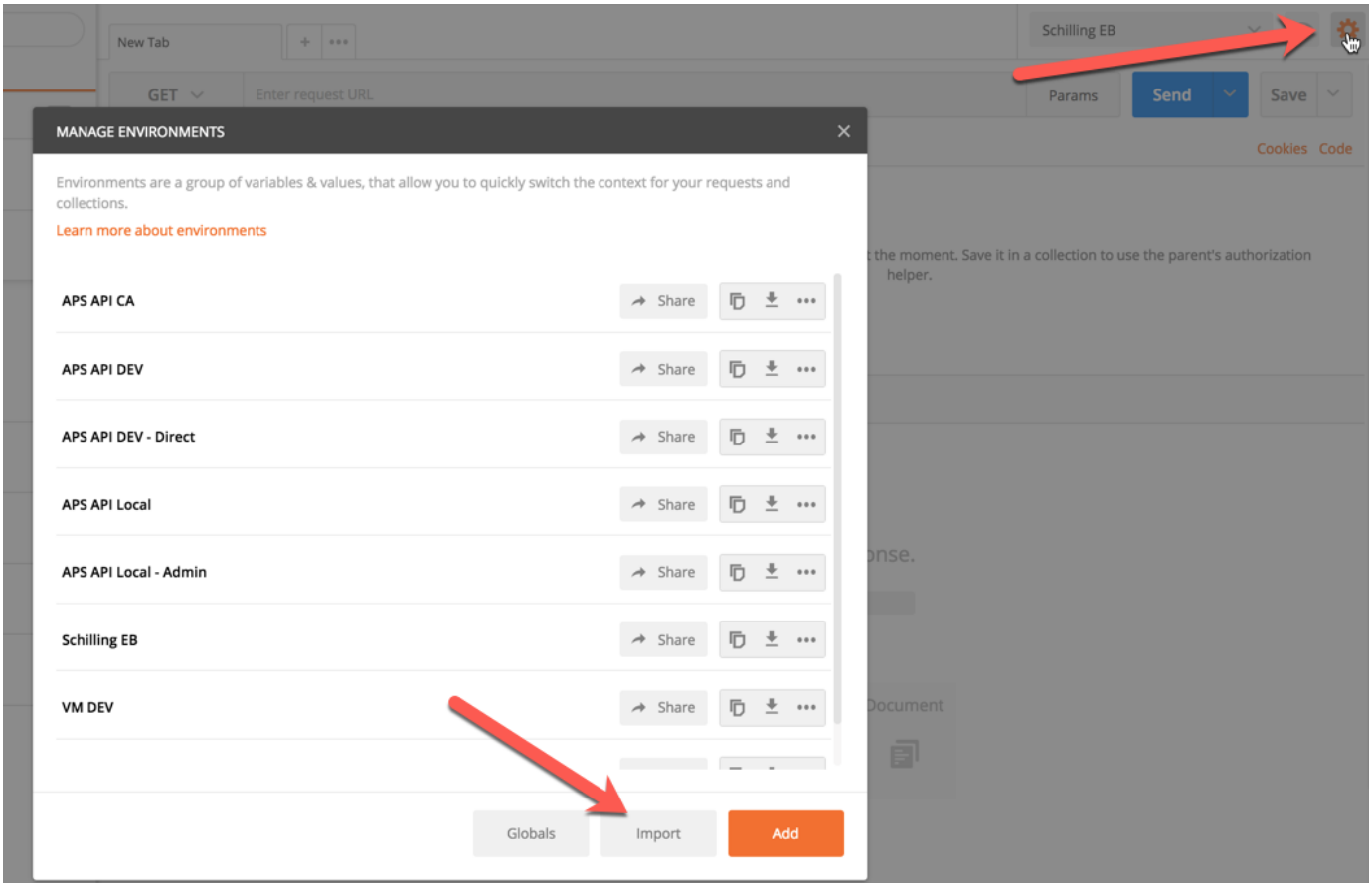
Drag and drop to upload or browse for files

18b - Adjust Postman Tests to Use AWS Server

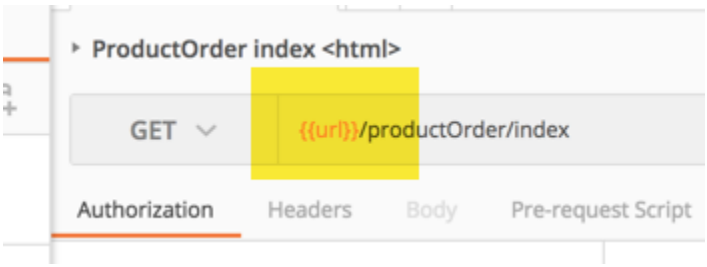
We will add environments to our Postman tests so we can point to our local environment or to the amazon webservices.

Take the two attached environments and import them into your postman.

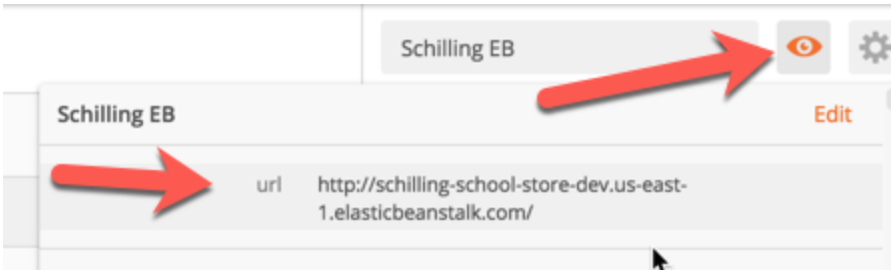
The first environment uses ***localhost:8080/*** to point to the application. The AWS (Amazon Web Services) url is ***http://schilling-school-store-dev.us-east-1.elasticbeanstalk.com/***. This will take you to a shared application that is managed in AWS. We will go through how this is deployed so you know. If we need to update the services to support our remote application, we'll have to redeploy.



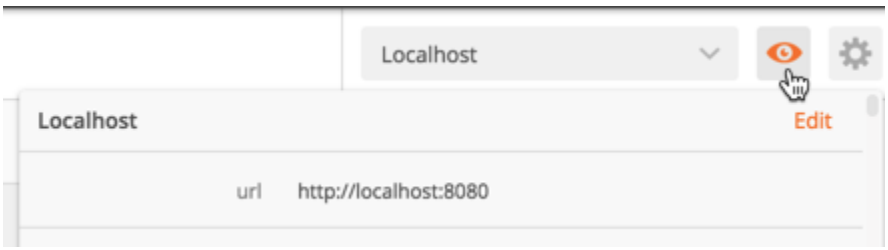
Once imported, update your requests to use "{{url}}" instead of "localhost:8080".



This matches a variable in the environment.



url is different for localhost



Then try out all your tests.

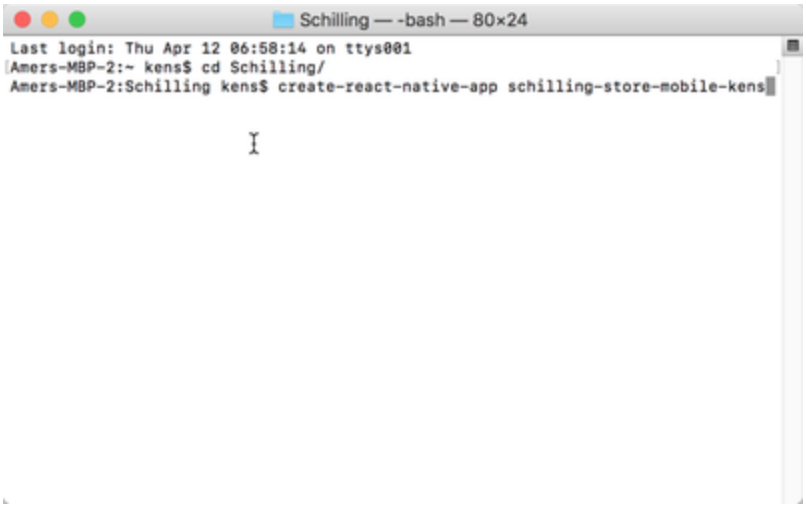
Commit and push your tests, request a merge into Core

Here are the environment files.

18c - Mobile Store Skeleton

In this project, we'll create our application again and put all the flow from our mockup in.

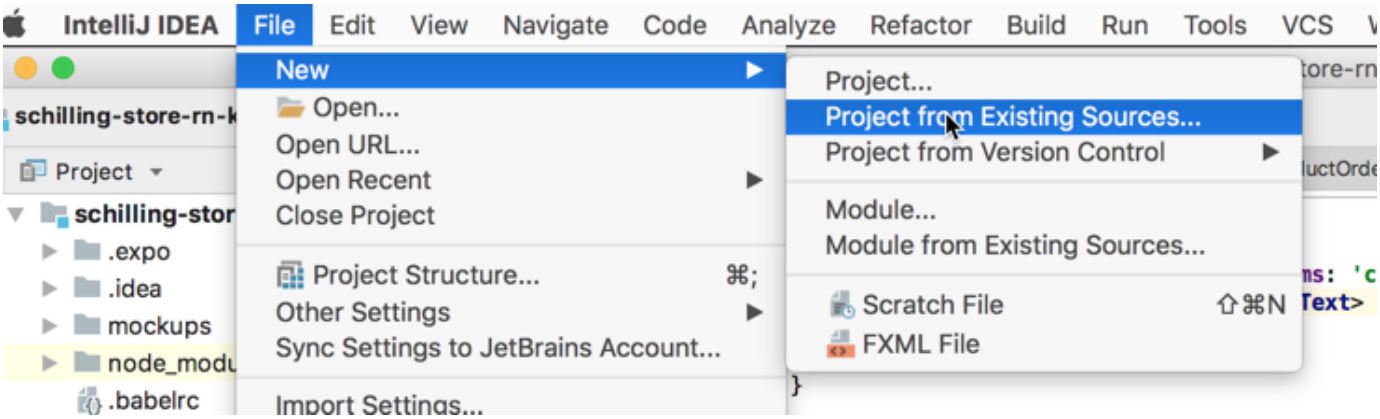
Start from a command or terminal window and create the react native application.

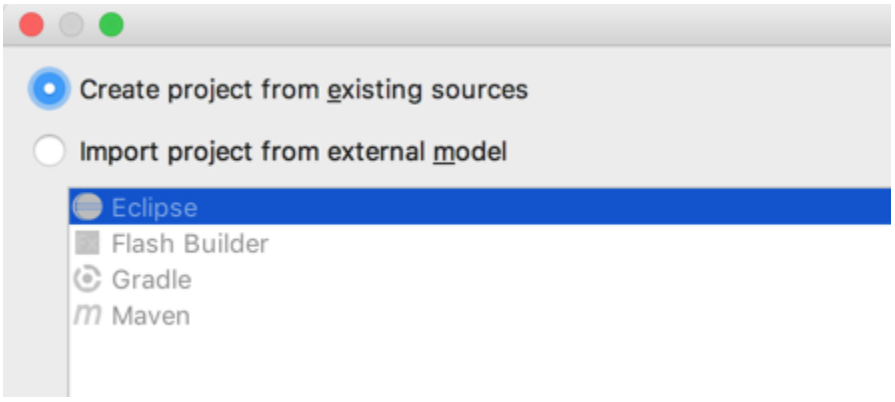


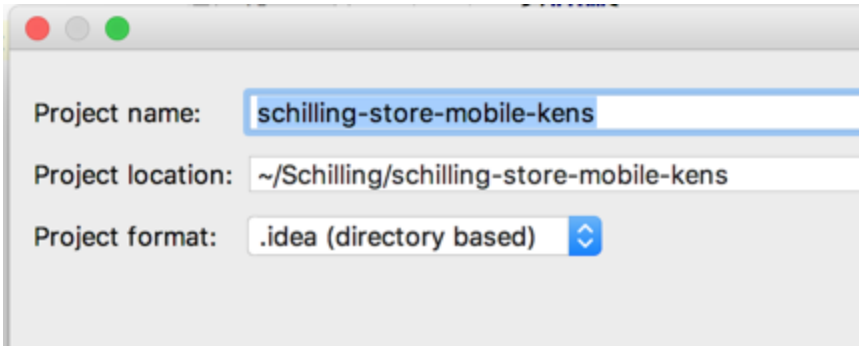
start the application as mentioned in previous exercises with npm start and the sysctl commands

once you have the application working in expo, stop it using ctrl-c.

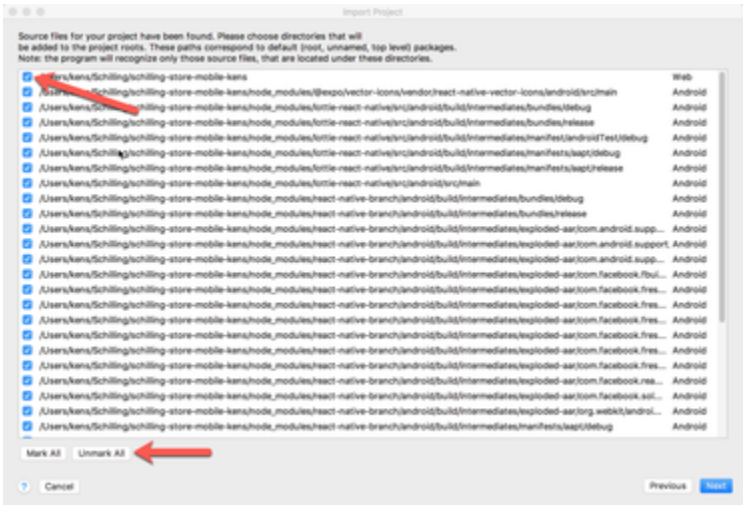
open the application in intellij by just opening existing sources





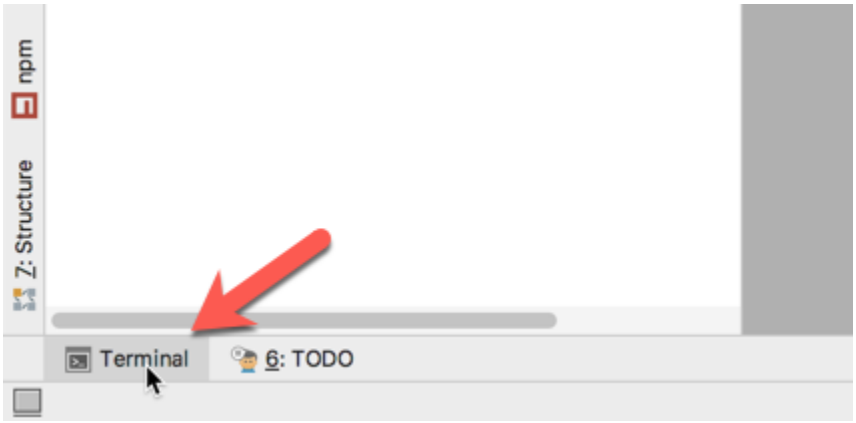


You may get a million directories to use as the project roots. Unmark all and then just select the first.

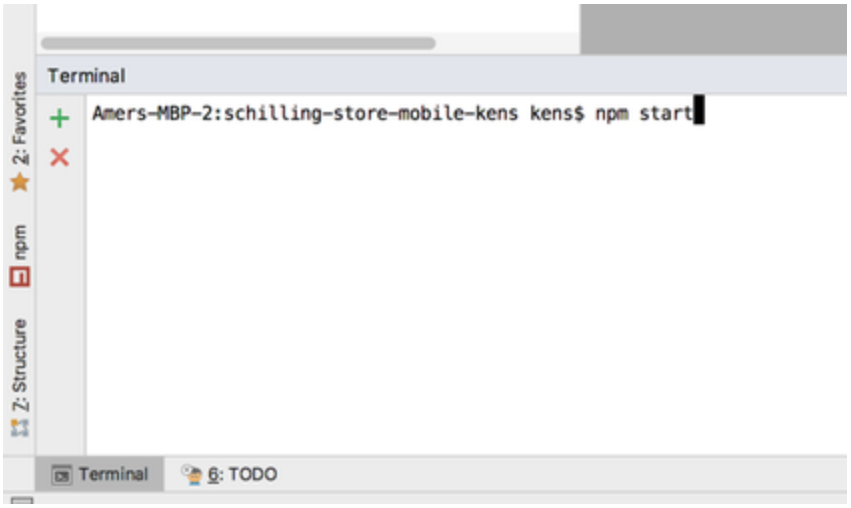


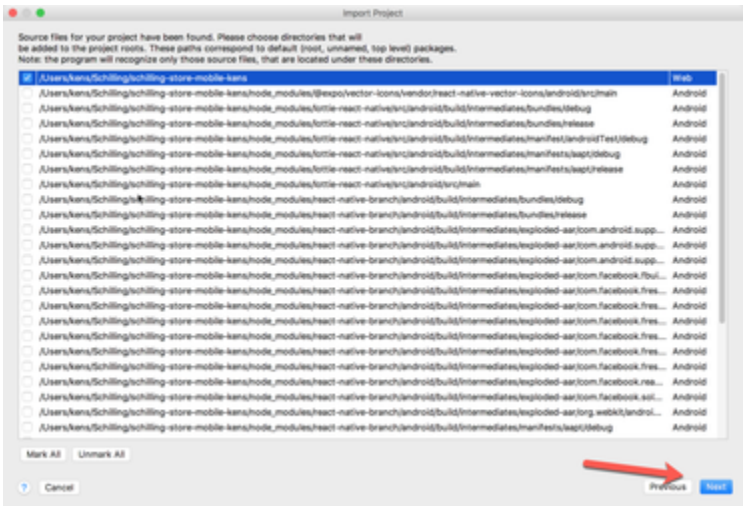
Finish

Open terminal in IntelliJ IDEA



run the npm start command





Now we can edit the application in IntelliJ and see the changes on our phone.

Open the App.js file and change the text to see the change on your phone. You may need to refresh. Shake your phone, then refresh. Now everytime you update a file, it should automatically refresh.

We will be using react-navigation to manage our screen flow. Here are the instructions for getting started: https://reactnavigation.org/docs/getting-started.html

You can use a separate command window or the terminal in IntelliJ. You may need to stop your app first with ctrl-c.

execute the command: npm install --save react-navigation

Now follow the hello navigation tutorial: https://reactnavigation.org/docs/hello-react-navigation.html

From there we'll develop the application with our pages and navigation to match our mockups.

Below is a first cut at the app with navigation. It has representative controls and flows. This could be the basis for your app. Just replace the code in App.js with this.

App.js

The final version of this assignment should follow your mockup as much as you can. Some of you went crazy on your mockups and there may be aspects that are harder to do or that I haven't tried yet myself.

Imagination and effort will win you points on this assignment. Use the react-native site and google to answer questions. Be very careful with your changes as you may break things. Use local history to go back to known working versions.

18d - Adding Categories, Manufacturers and Sort to the App

Table of Contents

Add Categories and Manufacturers

The code below amends the previous App.js to add support for Categories, Manufacturers and a sort by name, category or manufacturer.

Fixes and Cart

This code adds the cart capability to the application. No changes were made to the services. Note the call to get product/index.json now has a ?max=100 appended to it. The service defaulted to max of 10 items and this allows us to override the default.

Adding Icons

Adding icons is a little tricky. You will need to install react-native-vector-icons with "npm install react-native-vector-icons --save" then "npm link".

You can look at this code to see where I am using them. Look for Icon.xxx.

See instructions on this page: https://github.com/oblador/react-native-vector-icons

Payment Input Fields

Another installment. This time it includes code to collect payment information on the purchase screen.

Styling

Some styling. Look into the search screen.

Styling

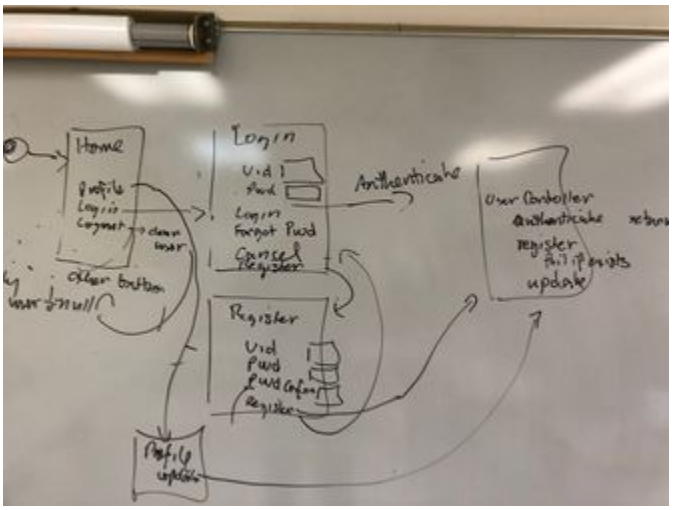
18e - Adding Login

Now that we have a pretty functional application, we would like to keep track of who is using it.

That requires a few things to happen:

- capture the credentials
- keep track of who is logged in
- maintain info about the user across sessions
- authenticate the user against an id and password
- optionally add authorization

How will the application flow?



The services have to be updated to include user management. This means our grails app has to change.

We will have two environments available. One with the new user (and reviews) management and the old one without it.

The new url is: http://schillingschoolpoc-test.us-east-1.elasticbeanstalk.com/

Make a new app, or if you like, use a branch to manage the differences in your application.

The final system should support:

- registration (new user)
- login (make sure the user exists and the password matches)
- logout
- don't allow purchase without login? this is your option, you could allow guest checkout.
- new screens
    - home with links to new screens
    - register
    - profile
    - login

You should create a second application. Follow these steps:

```
create-react-native-app schilling-store-mobile-kens-auth
open in the project in intellij
go to the terminal
npm install --save react-navigation
copy your app.js file from your other app
```

Initial flow code

Next, we'll add the ability to call the new server.

Add a variable at the top with:

let baseURL = "...";

This will allow us to replace the "magic value" throughout the program and make the change in one place.

So, go through and replace all the urls with baseURL + '...'

Here is some sample code:

There are a couple other places to change as well.

Test the app and make sure it still works.

Now use the new url. You will have to fix code to add to cart. When you create the order, you have to include the user. This code hard codes it to 1. You should use your logged in user id. something like ***currentUser.id*.**

Last, we start using the services from the application. We need to be able to create new users on register, authenticate users and pre fill info on the cart.

Here is the last spot of code, that is not quite complete.

We get an error if we don't login with the correct credentials.

For those of you on Android, this code works on my android device.

Here is some code to fix the login issue. This allows you to handle the bad authentication which comes back with a 500 status.

App Requirements

MoSCoW -

Must Have:

list products

view a product

purchase a product

Should Have:

account/login/profile

search

look for a product

sort by price/mfr/cat

Could Have:

recommendations

update wish list

update cart

list categories/mfrs

filter by price/mfr/cat

Won't Have:

advertisements

React Native Setup

Install Expo on your device and you can open my app.

Expo Link: https://expo.io/--/to-exp/exp%3A%2F%2F192.168.0.138%3A19000

```
Open in Expo:
https://expo.io/--/to-exp/exp%3A%2F%2F192.168.0.138%3A19000

Download Expo for Android at http://bit.ly/2bZq5ew or iOS at http://apple.co/2c6HMtp
The link to my app will be broken then ext time I start it up.
```

How-to articles

Add how-to article

Title	Creator	Modified
Clone Repository to disk from command line	Ketu Sayers	Jan 18, 2018
Implement Catgory Report	Ken Sayers	Dec 10, 2017
Import a Project from a Zip File	Ken Sayers	Dec 06, 2017
Bootstrapping	Ken Sayers	Sep 28, 2017
Create a Controller	Ken Sayers	Sep 26, 2017
Take a Screenshot	Ken Sayers	Sep 19, 2017
Running Tests	Ken Sayers	Sep 12, 2017
Enable Image Upload for Products	Ken Sayers	Sep 10, 2017
Installing Grails on windows	Ketu Sayers	Sep 07, 2017
Working with Grails Domain Classes	Ken Sayers	Sep 06, 2017
Create a Grails Domain Class	Ken Sayers	Sep 06, 2017
Create a Grails Project	Ken Sayers	Sep 06, 2017

Bootstrapping

Bootstrapping refers to pre configuration of an application during startup.

Grails supports bootstrapping with the ***Bootstrap.groovy*** in ***grails-app/init***

Step-by-step guide

1. Go to the Bootstrap.groovy in the grails-app/init directory.
2. Input your code to setup your instances. Below is a sample for the store.

Bootstrap.groovy

```
package schilling.store.kens

class BootStrap {

    def init = { servletContext ->
        Category catOne = new
Category(name:"One").save(failOnError:true)
        Category catTwo = new
Category(name:"Two").save(failOnError:true)
        Category catNuts = new
Category(name:"Nuts").save(failOnError:true)

        Manufacturer manOne = new
Manufacturer(name:"Manuone").save(failOnError:true)
        Manufacturer manTwo = new
Manufacturer(name:"Manutwo").save(faileOnError:true)
        Manufacturer man03 = new
Manufacturer(name:"Manu03").save(faileOnError:true)
        Manufacturer man04 = new
Manufacturer(name:"Manu04").save(faileOnError:true)
        Manufacturer man05 = new
Manufacturer(name:"Manu05").save(faileOnError:true)
        Manufacturer man06 = new
Manufacturer(name:"Manu06").save(faileOnError:true)
        Manufacturer man07 = new
Manufacturer(name:"Manu07").save(faileOnError:true)
        Manufacturer man08 = new
Manufacturer(name:"Manu08").save(faileOnError:true)
        Manufacturer man09 = new
Manufacturer(name:"Manu09").save(faileOnError:true)
        Manufacturer man10 = new
Manufacturer(name:"Manu10").save(faileOnError:true)
        Manufacturer man11 = new
Manufacturer(name:"Manu11").save(faileOnError:true)
        Manufacturer man12 = new
Manufacturer(name:"Manu12").save(faileOnError:true)
        Manufacturer man13 = new
Manufacturer(name:"Manu13").save(faileOnError:true)
        Manufacturer man14 = new
```

```
Manufacturer(name:"Manu14").save(faileOnError:true)
        Manufacturer man15 = new
Manufacturer(name:"Manu15").save(faileOnError:true)
        Manufacturer man16 = new
Manufacturer(name:"Manu16").save(faileOnError:true)
        Manufacturer man17 = new
Manufacturer(name:"Manu17").save(faileOnError:true)
        Manufacturer manNutty = new
Manufacturer(name:"Nutty").save(faileOnError:true)

        new Product(name: "POne",description: "Described",category:
catOne, manufacturer: manOne, price: 10.10).save(failOneError:true)
        new Product(name: "PTwo",description: "Described",category:
catTwo, manufacturer: manTwo, price: 20.20).save(failOneError:true)

        byte[] macadamiaNutsImageBytes =
this.class.getClassLoader().getResourceAsStream("macadamia_nuts.jpeg").b
ytes
        byte[] brazilNutsImageBytes =
this.class.getClassLoader().getResourceAsStream("brazil_nuts.jpeg").byte
s
        new Product(name: "Macadamia Nuts",description: "Macadamias
1lb",category: catNuts, manufacturer: manNutty, price: 10.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
2lb",category: catNuts, manufacturer: manNutty, price: 20.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
3lb",category: catNuts, manufacturer: manNutty, price: 30.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
4lb",category: catNuts, manufacturer: manNutty, price: 40.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
5lb",category: catNuts, manufacturer: manNutty, price: 50.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
6lb",category: catNuts, manufacturer: manNutty, price: 60.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
7lb",category: catNuts, manufacturer: manNutty, price: 70.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
```

```
8lb",category: catNuts, manufacturer: manNutty, price: 80.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
9lb",category: catNuts, manufacturer: manNutty, price: 90.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
10lb",category: catNuts, manufacturer: manNutty, price: 100.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Macadamia Nuts",description: "Macadamias
11lb",category: catNuts, manufacturer: manNutty, price: 110.10,
imageBytes: macadamiaNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
1lb",category: catNuts, manufacturer: manNutty, price: 10.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
2lb",category: catNuts, manufacturer: manNutty, price: 20.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
3lb",category: catNuts, manufacturer: manNutty, price: 30.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
4lb",category: catNuts, manufacturer: manNutty, price: 40.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
5lb",category: catNuts, manufacturer: manNutty, price: 50.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
6lb",category: catNuts, manufacturer: manNutty, price: 60.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
7lb",category: catNuts, manufacturer: manNutty, price: 70.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
8lb",category: catNuts, manufacturer: manNutty, price: 80.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
9lb",category: catNuts, manufacturer: manNutty, price: 90.10,
imageBytes: brazilNutsImageBytes, imageContentType:
```

```
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
10lb",category: catNuts, manufacturer: manNutty, price: 100.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
        new Product(name: "Brazil Nuts",description: "Brazils
11lb",category: catNuts, manufacturer: manNutty, price: 110.10,
imageBytes: brazilNutsImageBytes, imageContentType:
'image/jpeg').save(failOneError:true)
    }
    def destroy = {
```

```
        }
    }
```

Clone Repository to disk from command line

In our latest assignment, we've been insructed to create a repository on disk. However, doing as listed in the assignment will result in an error along the lines of "repository already exists" This is how instead of trying te create a remote repo, you can instead clone it to work on.

Step-by-step guide

1. use the cd (or equivalent navigation) command to move your current bash directory into the location you want to project: `$ cd Documents/CSP18/Version\ control`
2. use the 'git clone' command to copy the repo to disk: `$ git clone` https://github.com/SchillingCSP/vcs-exercise-01.git
3. check out your branch: `$ git checkout <branch-name>`
4. Proceed with the assignment as listed.

Related articles

- Clone Repository to disk from command line
- Implement Catgory Report
- Import a Project from a Zip File
- Bootstrapping
- Create a Controller

Create a Controller

Controllers broker the communication between the domain and the views.

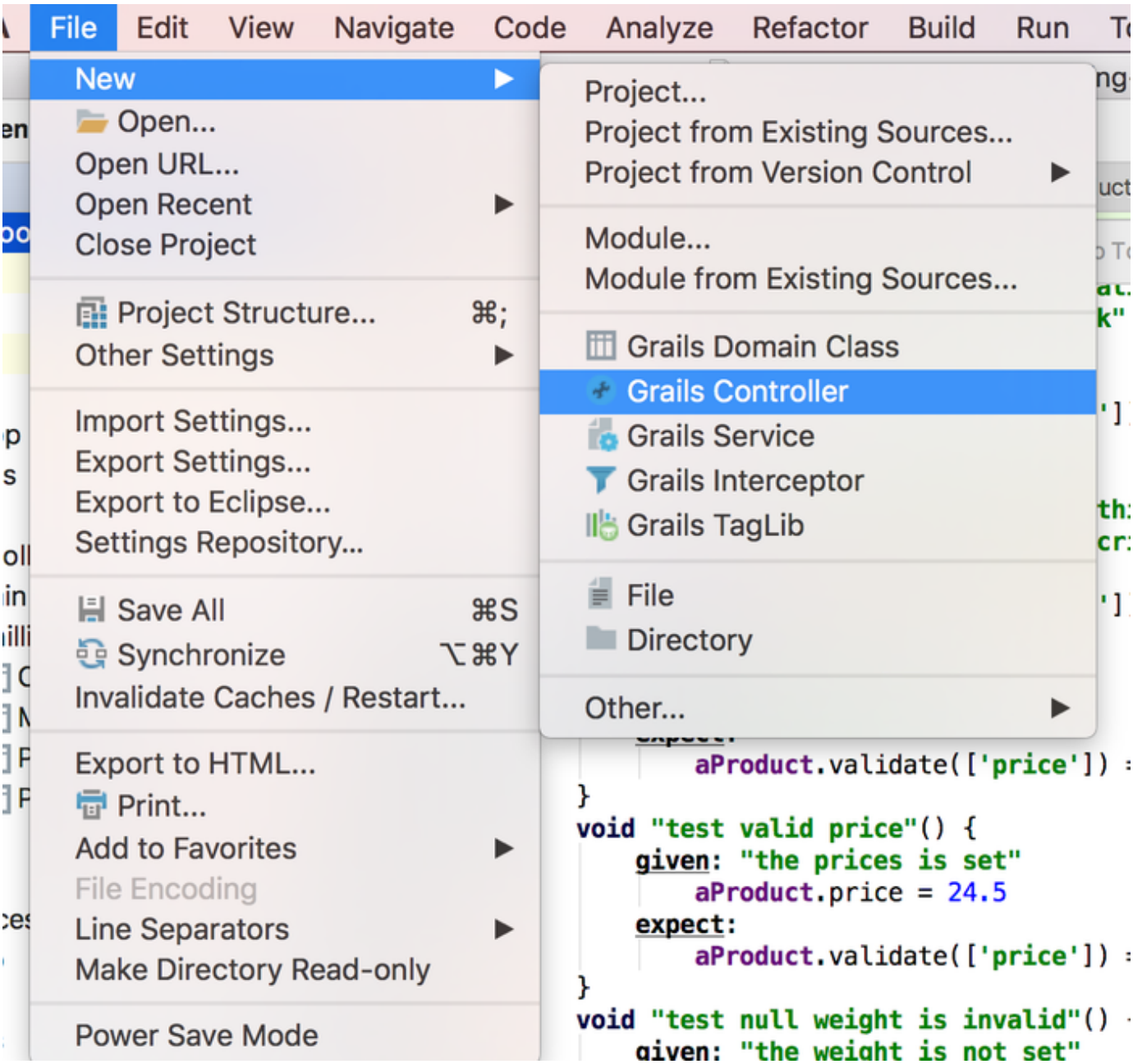
After the domain is created, the next step is to create a controller and use scaffolding to get the system up and running.

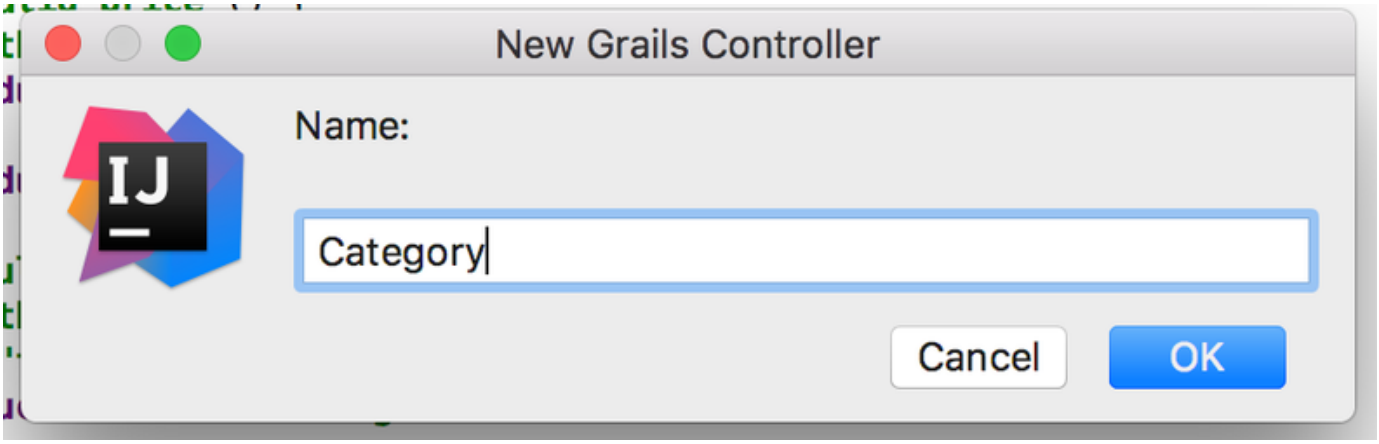
Scaffolding makes it possible to use the conventional implementation of the controller as a starting point.

After that, you will generate the controller.

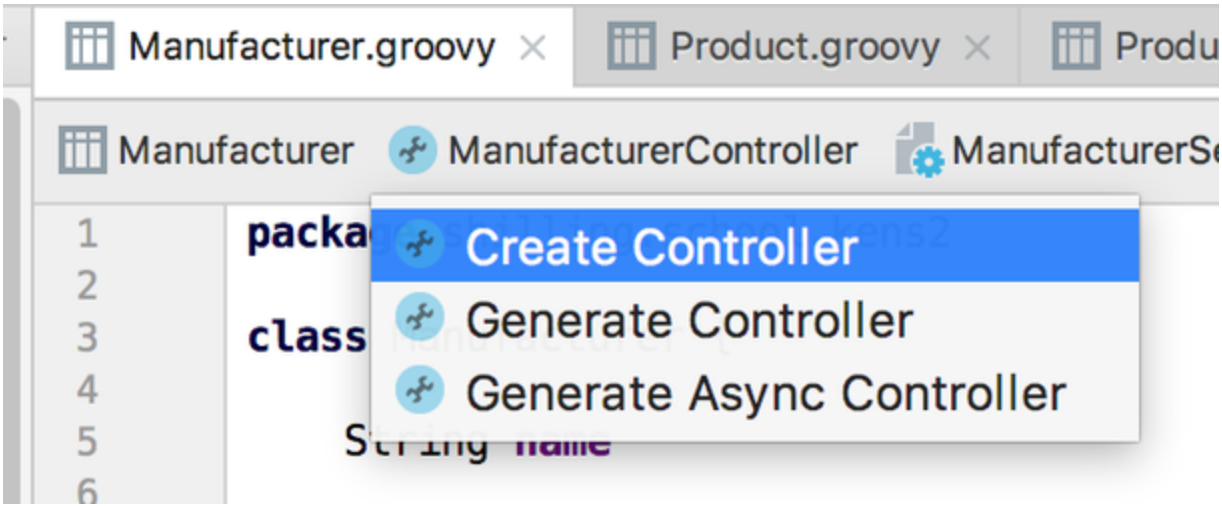
Step-by-step guide

- Create the controller

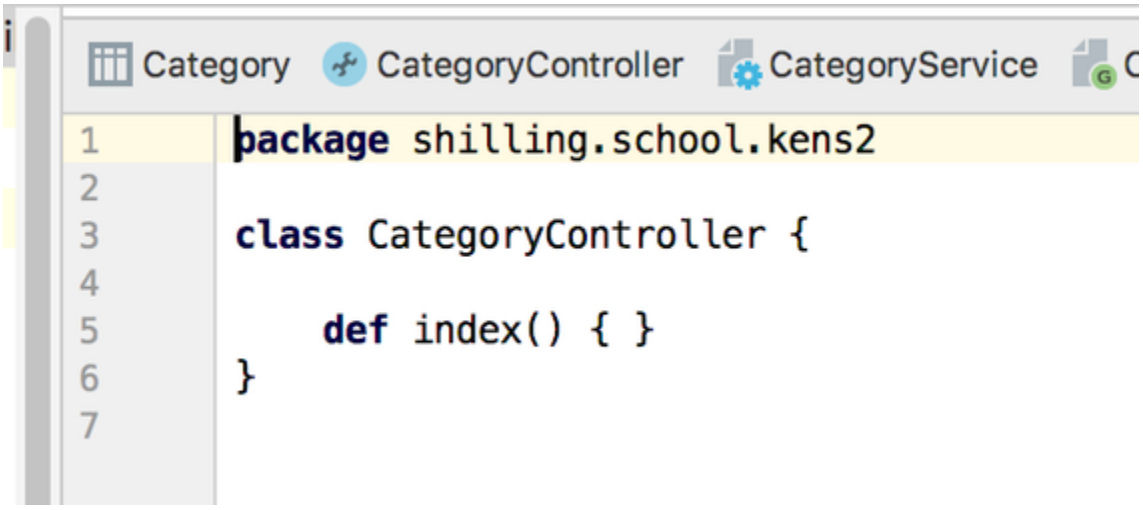




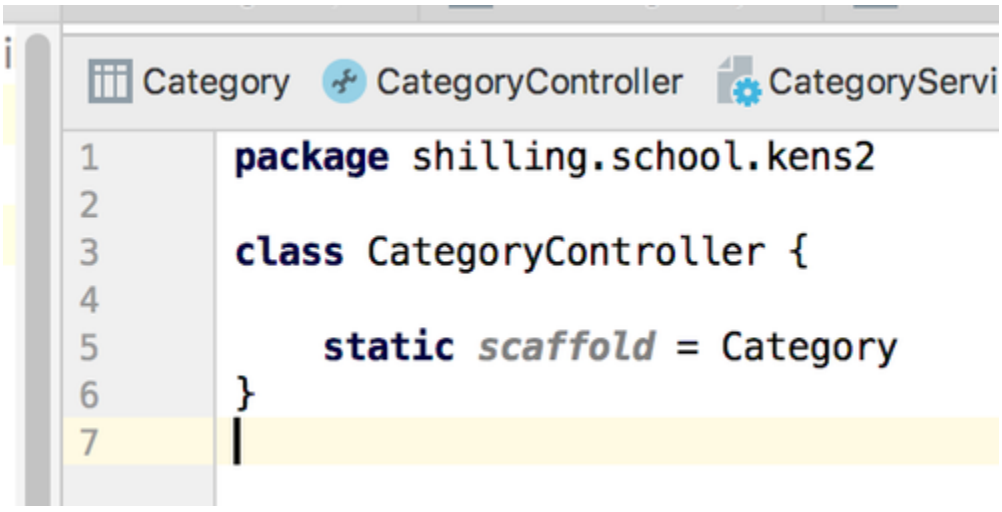
- You can also create the controller from the menu at the top of the domain class editor.



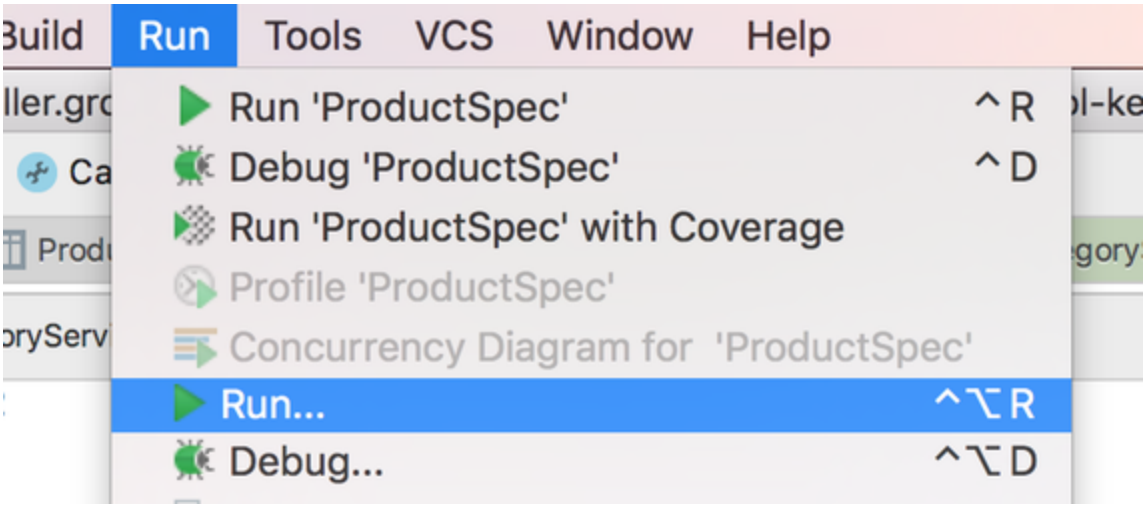
- The controller is opened in the editor. If not, go to the ***grails-app/controllers*** and open the CategoryController.groovy file. You will see this:

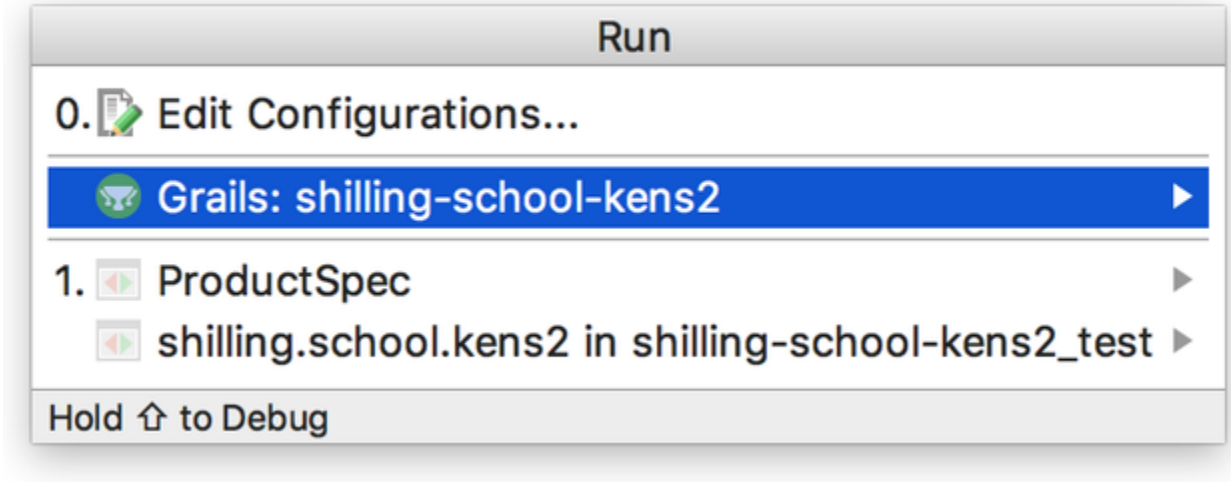


- Make it a scaffolded controller. Change the code to the following:

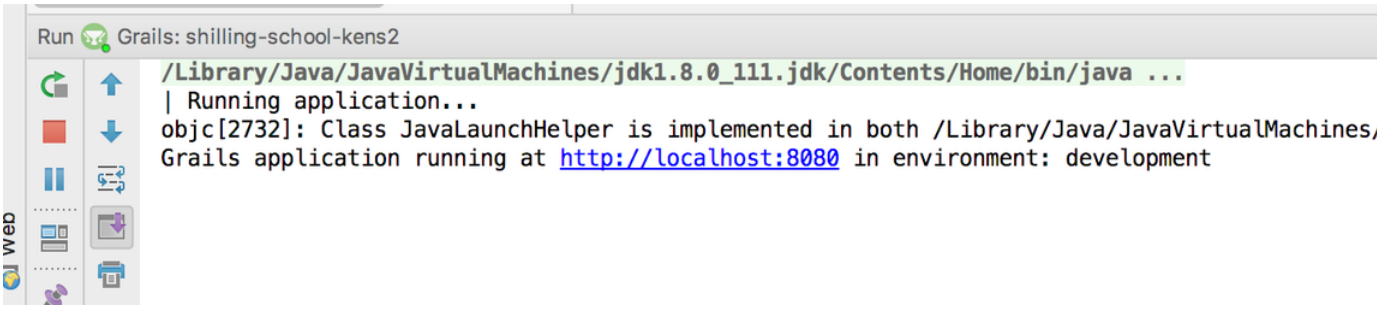


- Now you can start your application and exercise the controller.
- Run the application.

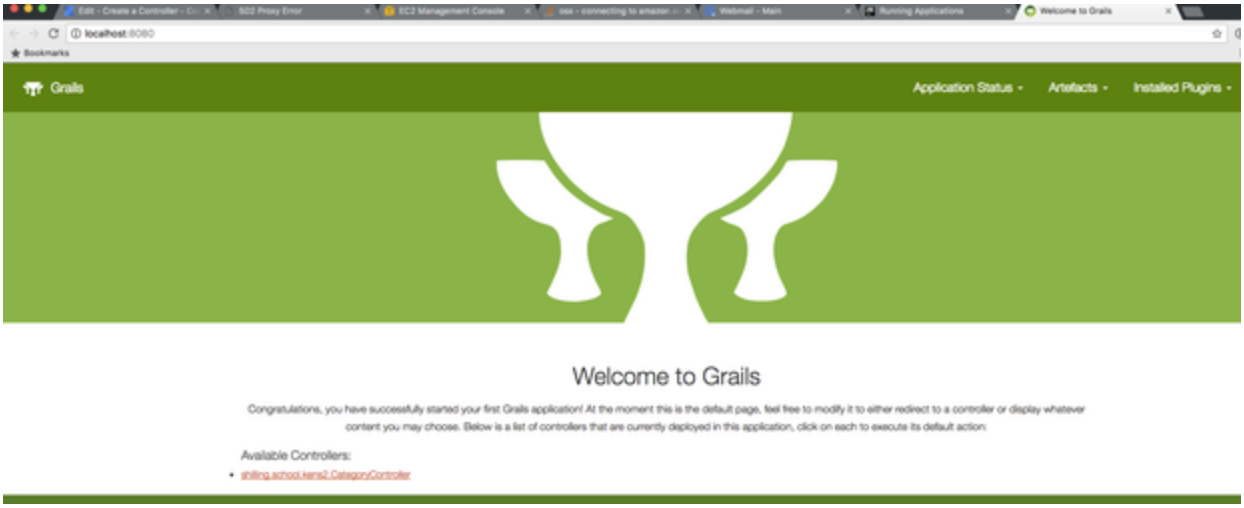




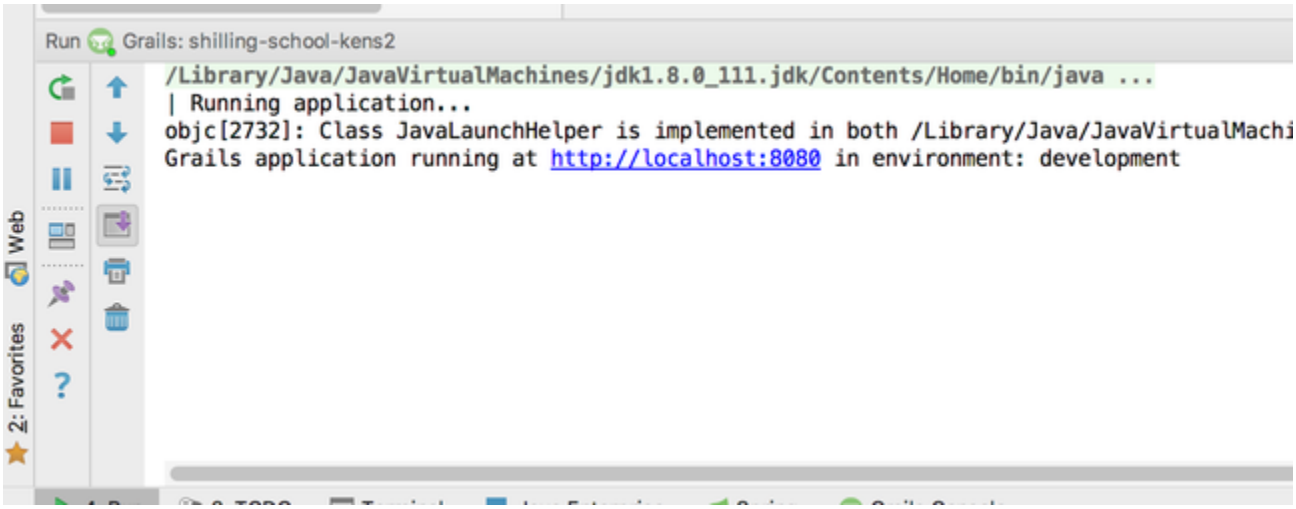
- You should see something similar to this in the console at the bottom.



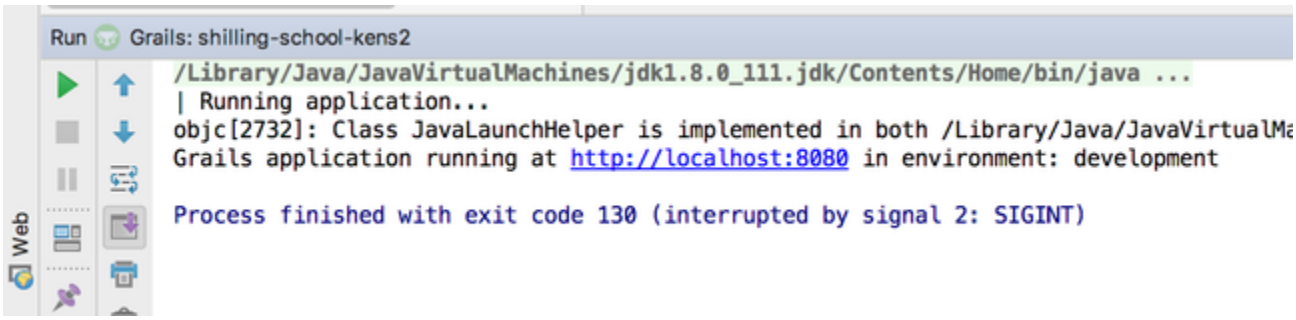
- And you may also see the browser with the following. If not, open your browswer to the url in the console. Usually http://localhost:8080



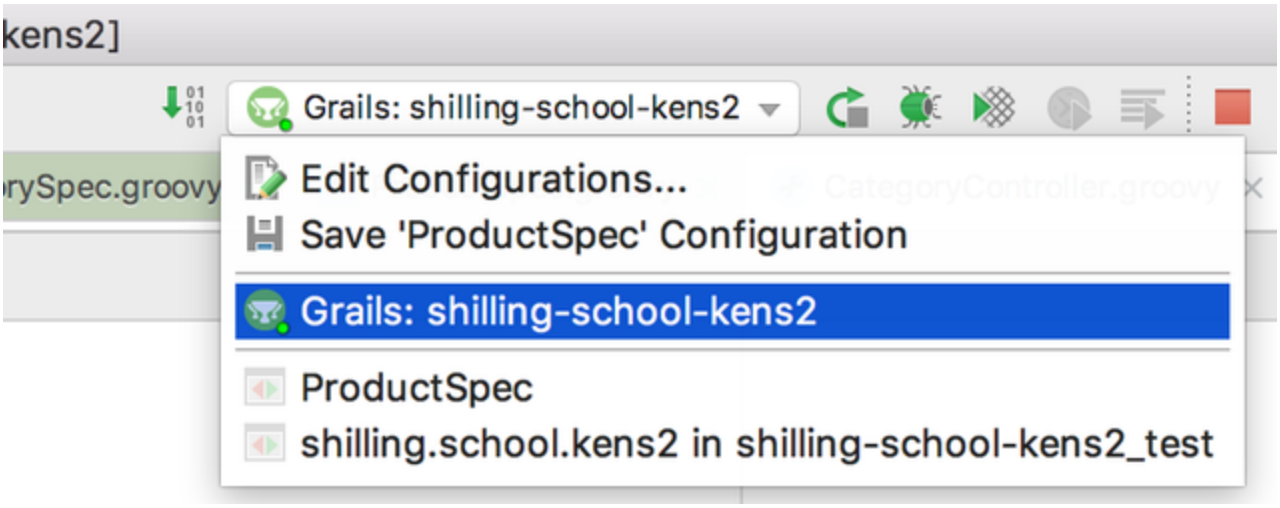
- You can see the controllers you have created in the list at the bottom.
- Click on the link and you will see the link.
- Try out to see what you can do with the system.
- When you want to stop the system use the red button next to the console



- The button will o grey and you will get another message in the console.



- To restart the application press the green button.
- If you have done a bunch of things in between, this may start the last thing you did instead of the application.
- If so, use the drop down at the top to choose what to run.

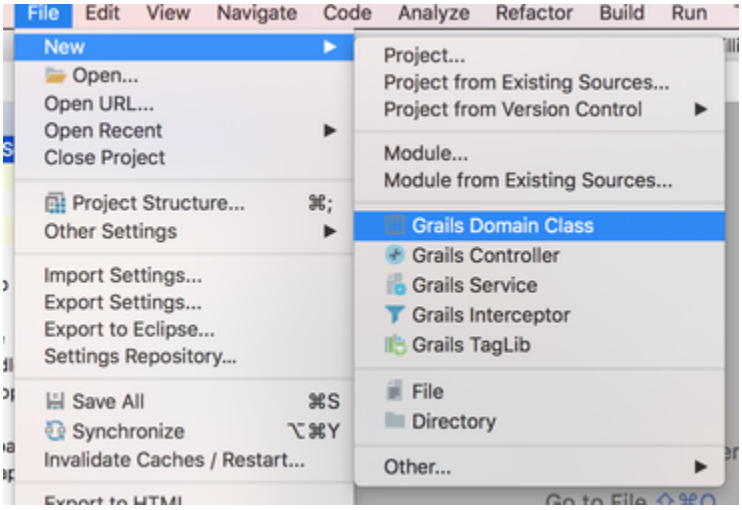


- Choose the one with ***Grails: <your project name>***

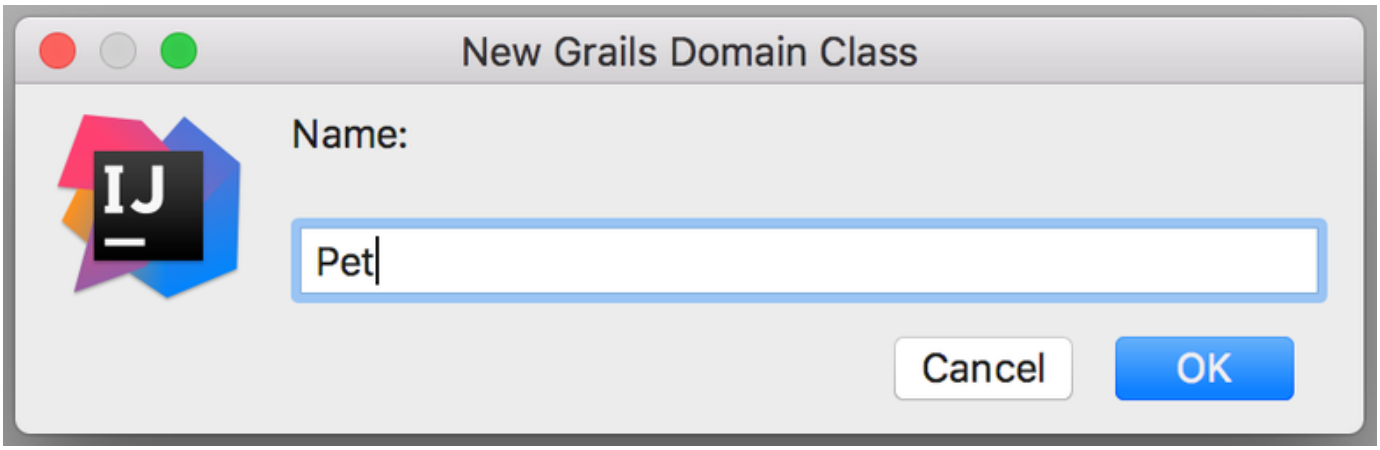
Create a Grails Domain Class

Step-by-step guide

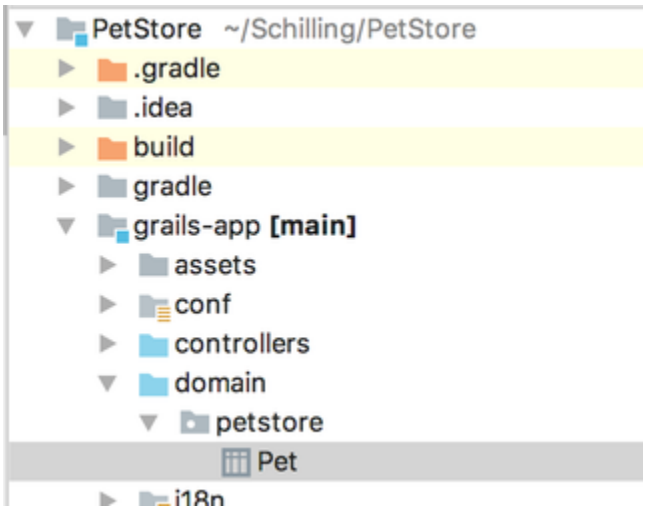
- Open IntelliJ IDEA
- Open your Project
- Create the Domain Class



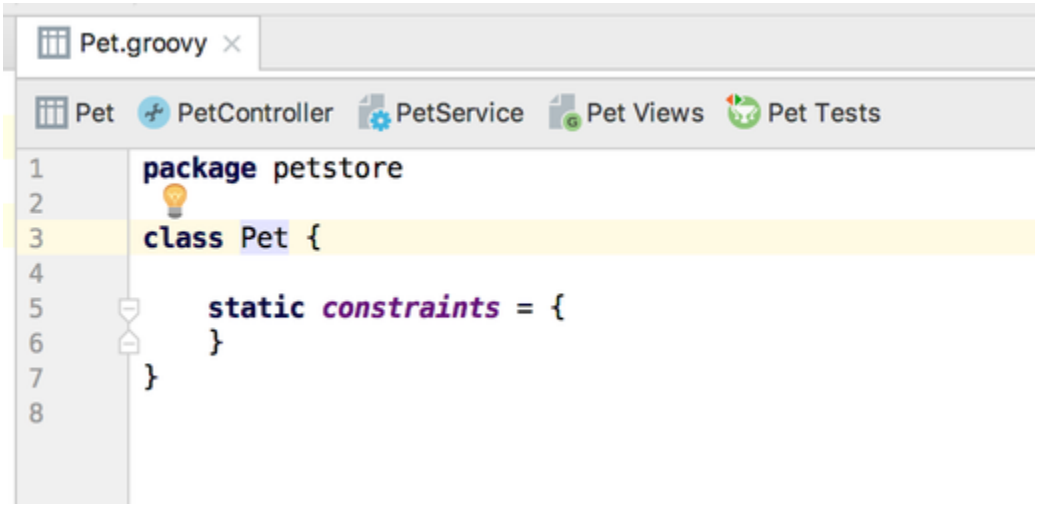
- Give it a Name



View Your Domain Class. It will be located in ***grails-app/domain.***



- Double click on the file and it will show in the editor.



- Finished

Concepts to Learn

- Domain Class
- Grails Object Relational Mapping (GORM)
- Constraints
- Relationships

The domain class has a few sections we'll be discussing.

Details about domain classes in grails can be found in the documentation here:

http://docs.grails.org/latest/guide/GORM.html

Related articles

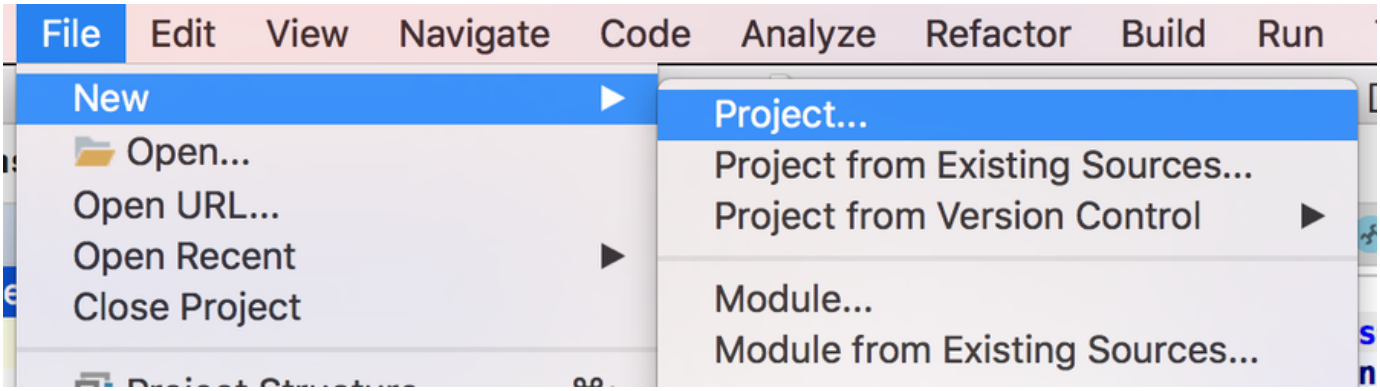
- Clone Repository to disk from command line
- Implement Catgory Report
- Import a Project from a Zip File
- Bootstrapping
- Create a Controller

Create a Grails Project

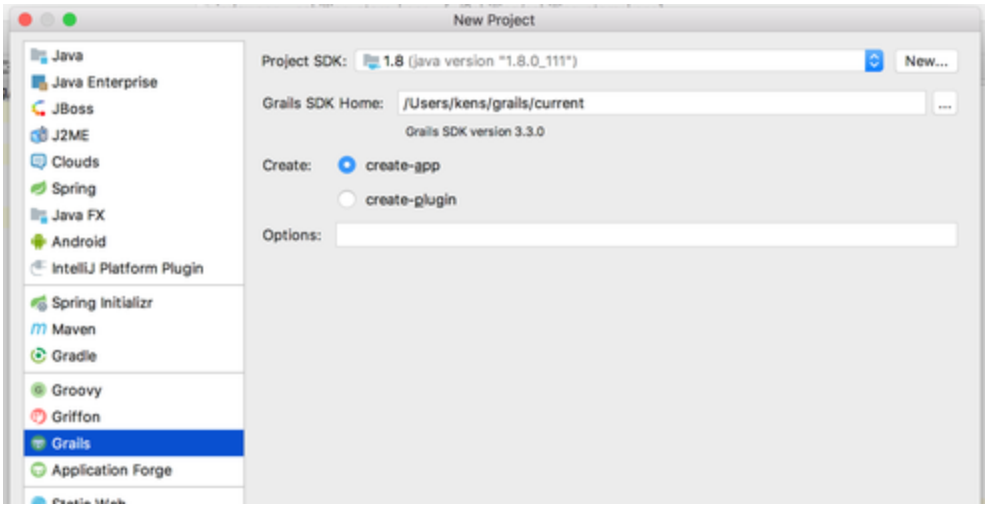
In this How-To, we create a new project in Grails.

Step-by-step guide

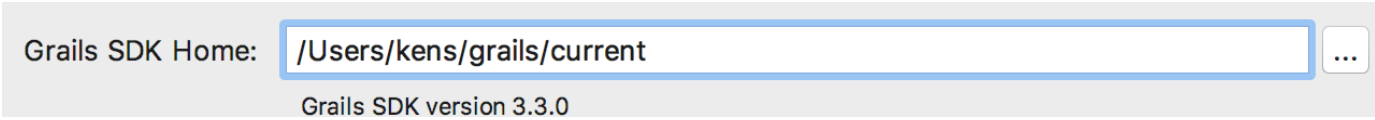
- Open IntelliJ IDEA
- Create a new Project



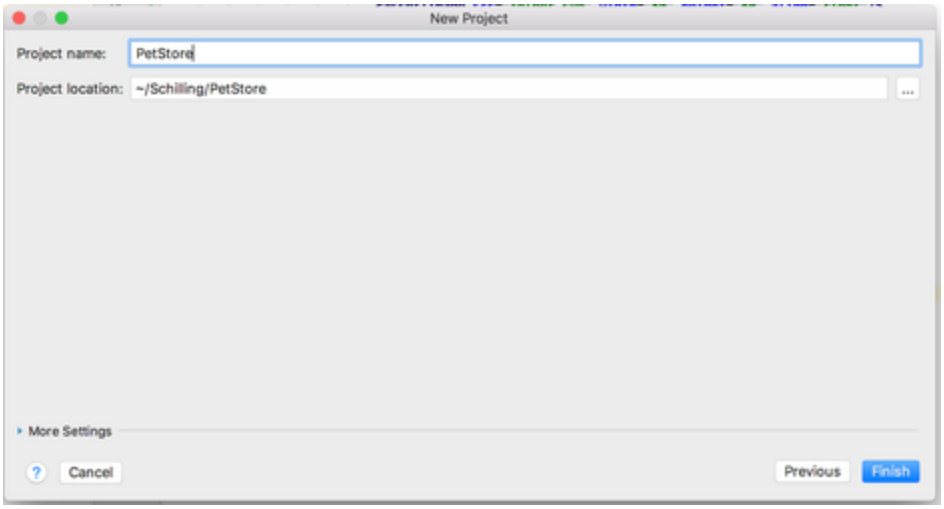
- Make it Grails Project



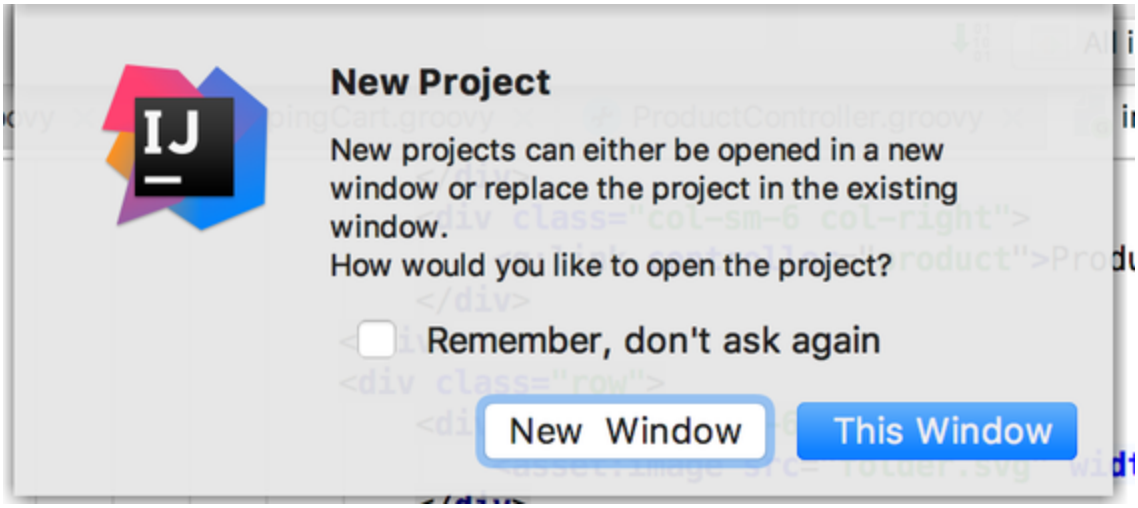
- Make sure you point to your Grails SDK (Software Development Kit). On a mac you can do "which grails" in a terminal window. On Windows it will be where you unzipped it.



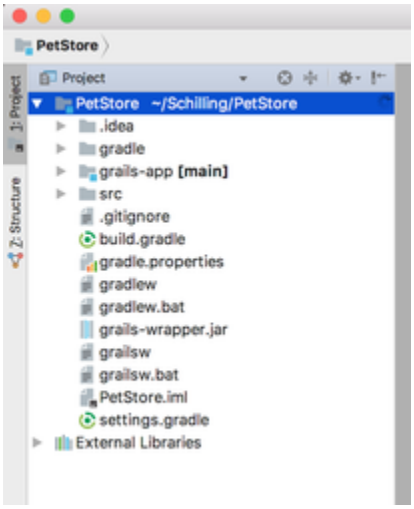
- Give it a Name



- Open in a New Window



- After a short time, you should see a project window with the project structure. Open the top level twisty:



- Finish

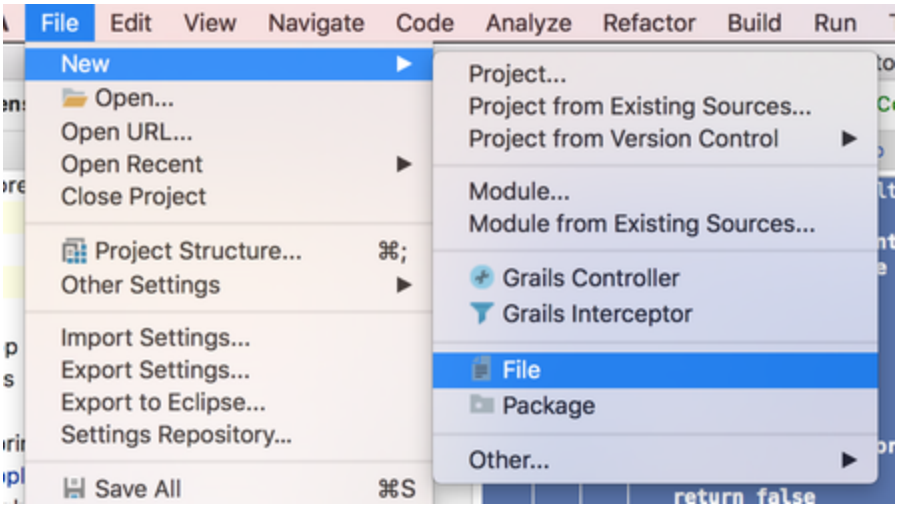
Enable Image Upload for Products

Here we show how to setup the application so we can upload images for our products.

Step-by-step guide

Create a Command

- Create a command for uploading the image. Place this file into the controllers directory. There is no grails command for this, so just create a new file in the directory using the ***newfile***



- Copy this code into the file. Be wary of the package. It must manage your directory structure.

/grails-app/controllers/schilling/store/kens/ProductImageCommand

Increase the allowed file size

- make the changes to application.yml file. Note, there should alread be a grails / controllers section. Just add the upload to it.

/grails-app/conf/application.yml

- my final file looks like this:

Setup the Domain

- The domain gets two new instance variables, some constraints and some mapping

/grails-app/domain/schilling/store/kens/Product.groovy

Setup the Views

- We will update each of the views to ignore these new fields
- Adjust the list of fields to match your domain class
- first index.gsp

/grails-app/views/product/index.gsp

- next create.gsp

/grails-app/views/product/create.gsp

- next edit.gsp

/grails-app/views/product/edit.gsp

- next show.gsp

/grails-app/views/product/show.gsp

- create editImage.gsp

/grails-app/views/product/editImage.gsp

Setup the Cotnroller

- update the ProductController.groovy class with three new methods.

/grails-app/controllers/schilling/school/kens/ProductController.groovy

Add Messages to i18n

- i18n stands for Internationalization
- This is how we handle multiple languages and currencies in grails and java
- Update the messages.properties file (and other languages if you like)

/grails-app/i18n/message.properties

Setup Bootstrap

- Bootstrap.groovy is a great place to put startup code
- Especially useul in development.
- We'll setup a bunch of domain objects.
- First put your images into /src/main/resources
- Then code like the following can load the images into your products

/grails-app/init/schilling/store/kens/BootStrap.groovy

Forwarding Schilling Emails

With the release of the IntelliJ License link, I felt perhaps it was time I posted this, for those who don't know.

Since it's been reccomended that we use our schilling email accounts for software licenses with this class, schilling emails have just become essentially required. However, some of us find the UI and webmail itself boring, so here's how to set up a forwarder to your regular email.

1. Go to the Schilling webmail portal
2. Log in with your credentials. Default password should be Schilling123 if you've never used it before, but don't quote me on that. Ask in the office if it's not.
3. If it asks, select Roundcube.
4. In the top right corner, next to log out, click your email address.
5. Select "Forwarders" from the drop down menu. It's the eight option on my screen.
6. Click the "Add Forwarder" button on the left, near the top of the page.
7. In the box on the page you're brought to, type in the email you want to forward to.
8. Add Forwarder with the button below the entry box.
9. Click "Go back" on the success screen, then follow steps 4 and 5 to confirm your forwarder is in the list.

Congratulations, you've just set up an auto-forward. Any emails will have the original sender information still listed as the sender when they reach your main mailbox, and will obey any spam filter on your email.

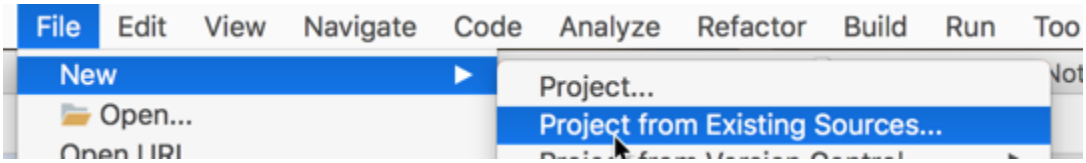
Import a Project from a Zip File

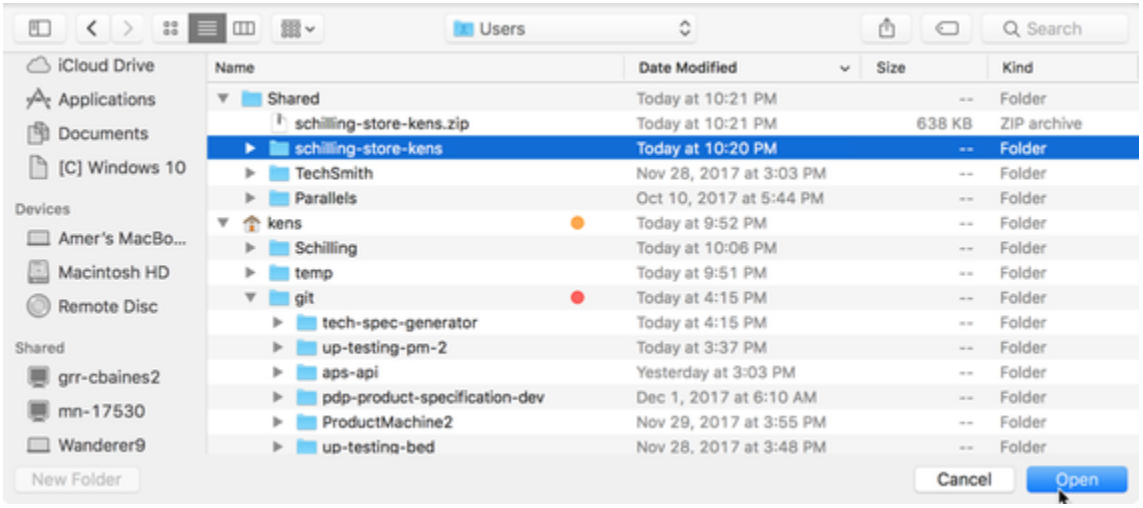
This guide helps you load a project into idea from a zip.

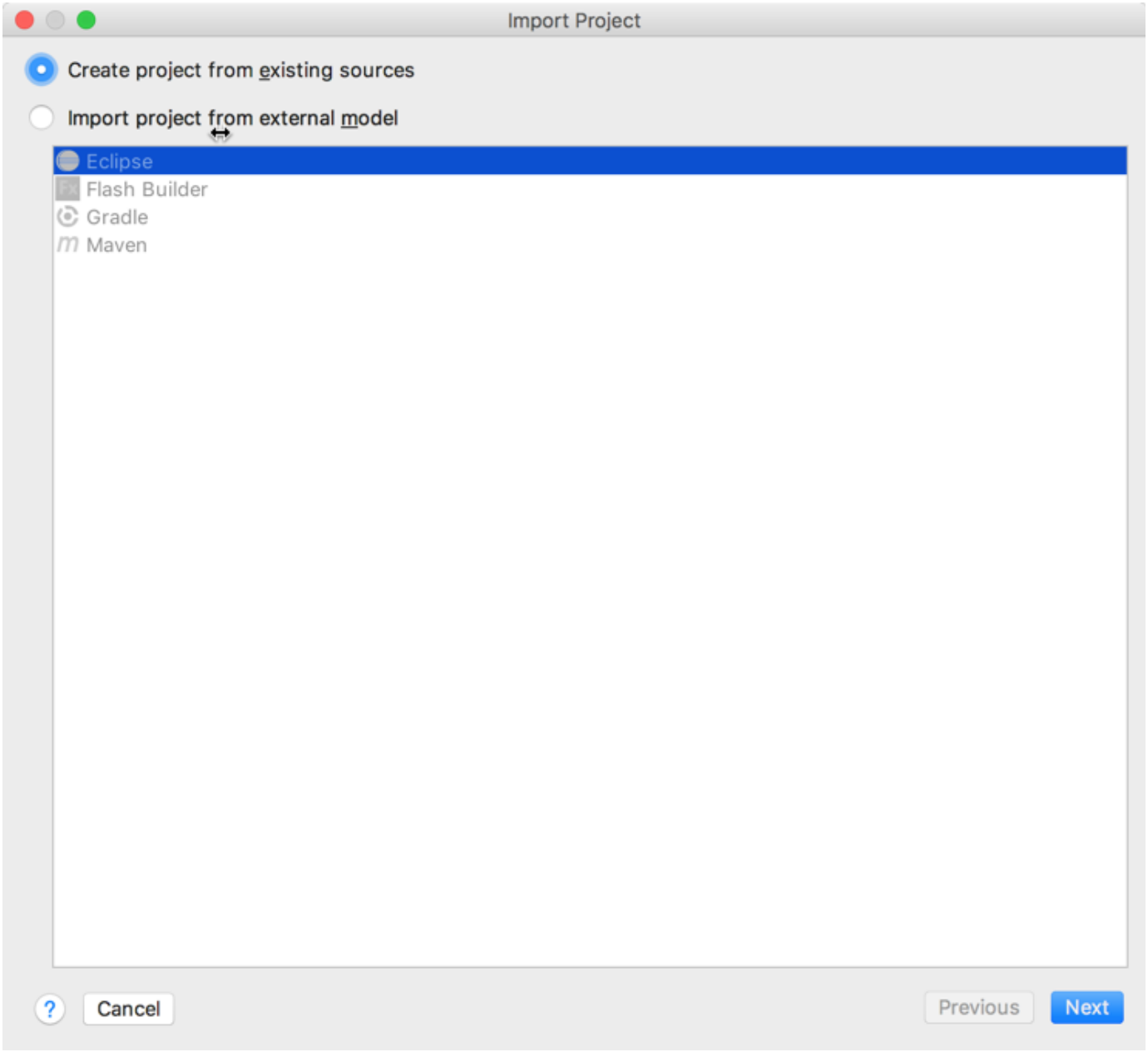
Step-by-step guide

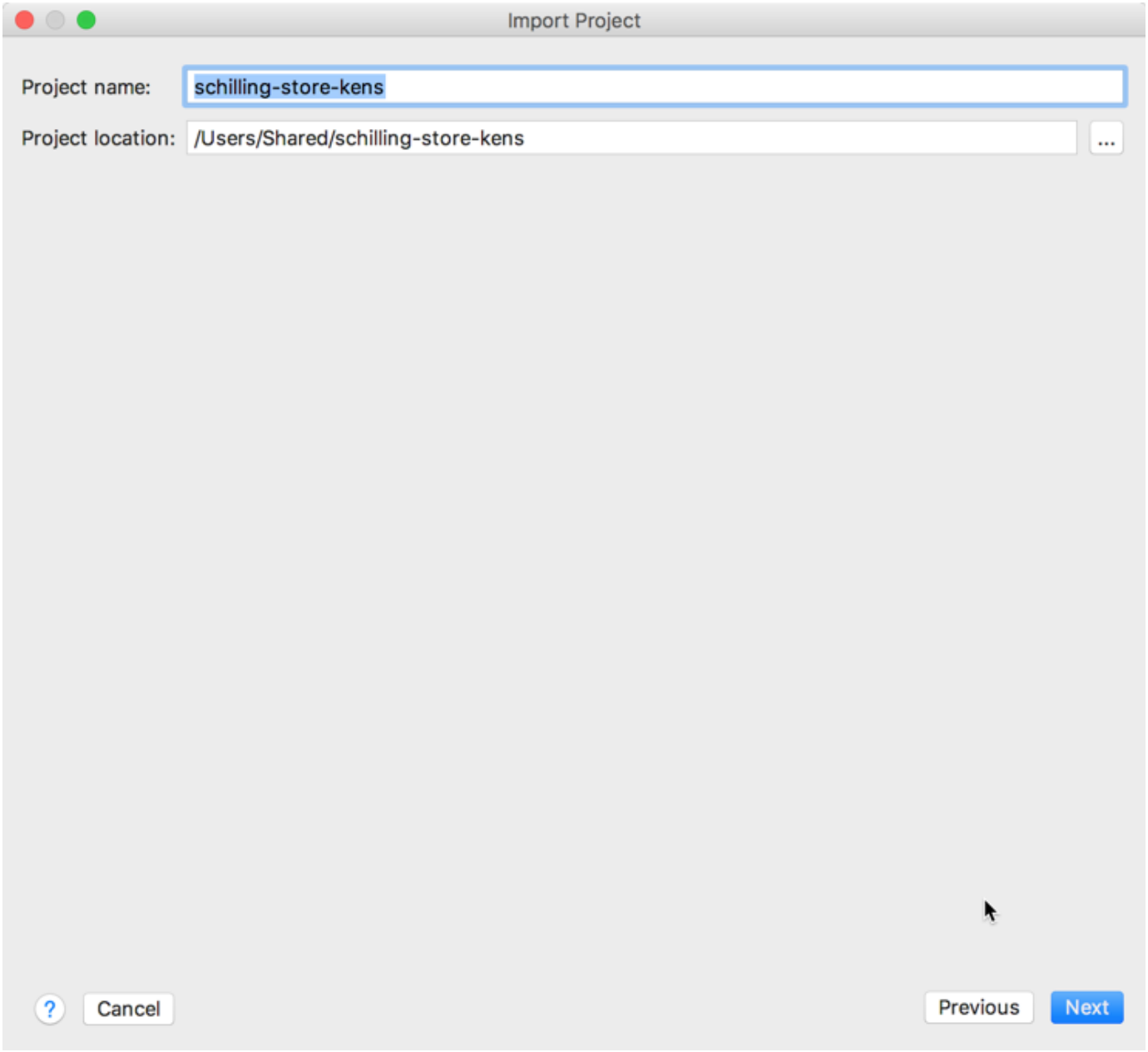
Follow these steps

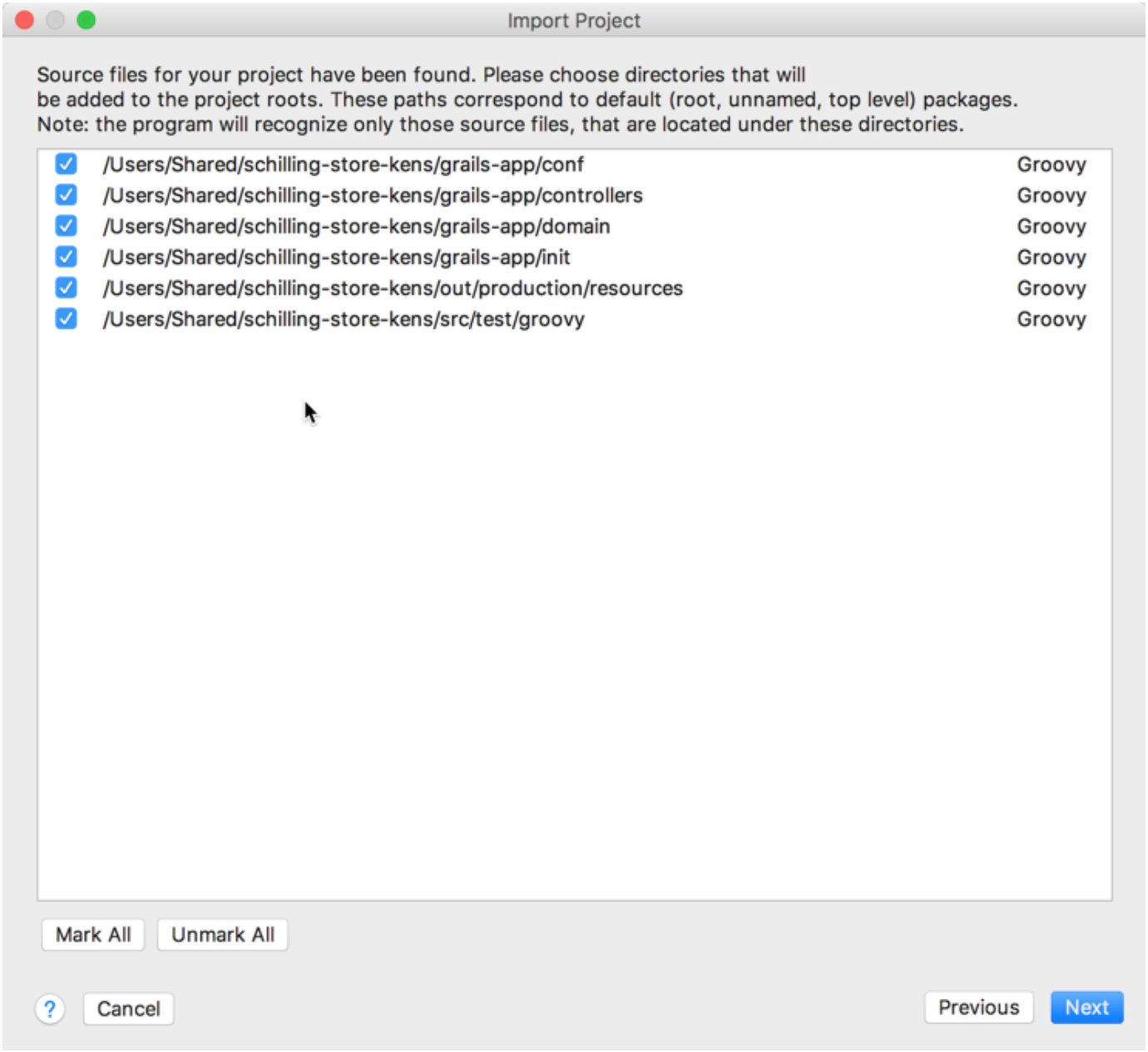
1. Download the zip file from the resources page
2. Unzip the file to a directory on your disk like C:\Development on windows or /Users/yourname/Development on mac.
3. In intellij, create a new project from existing sources

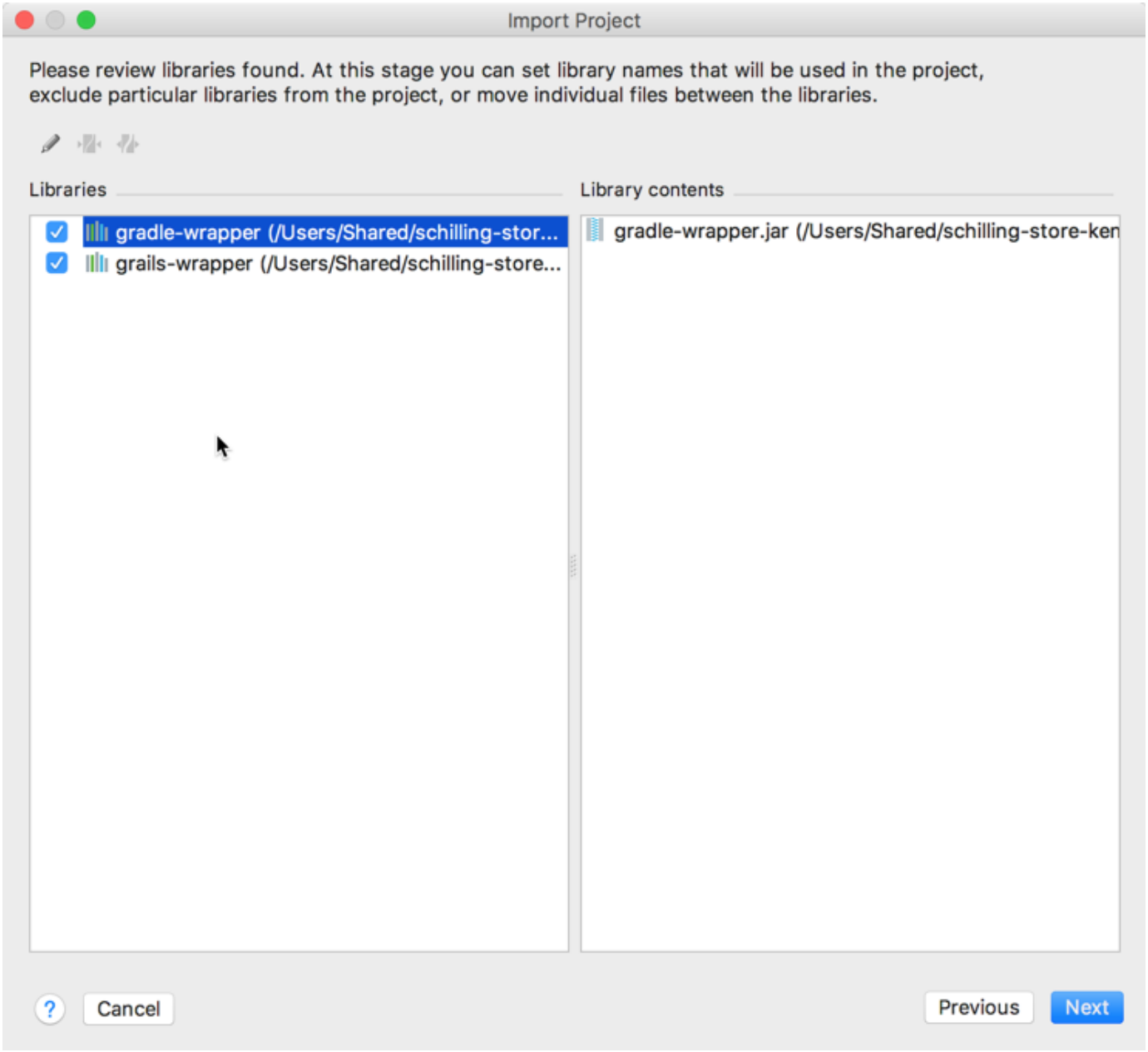


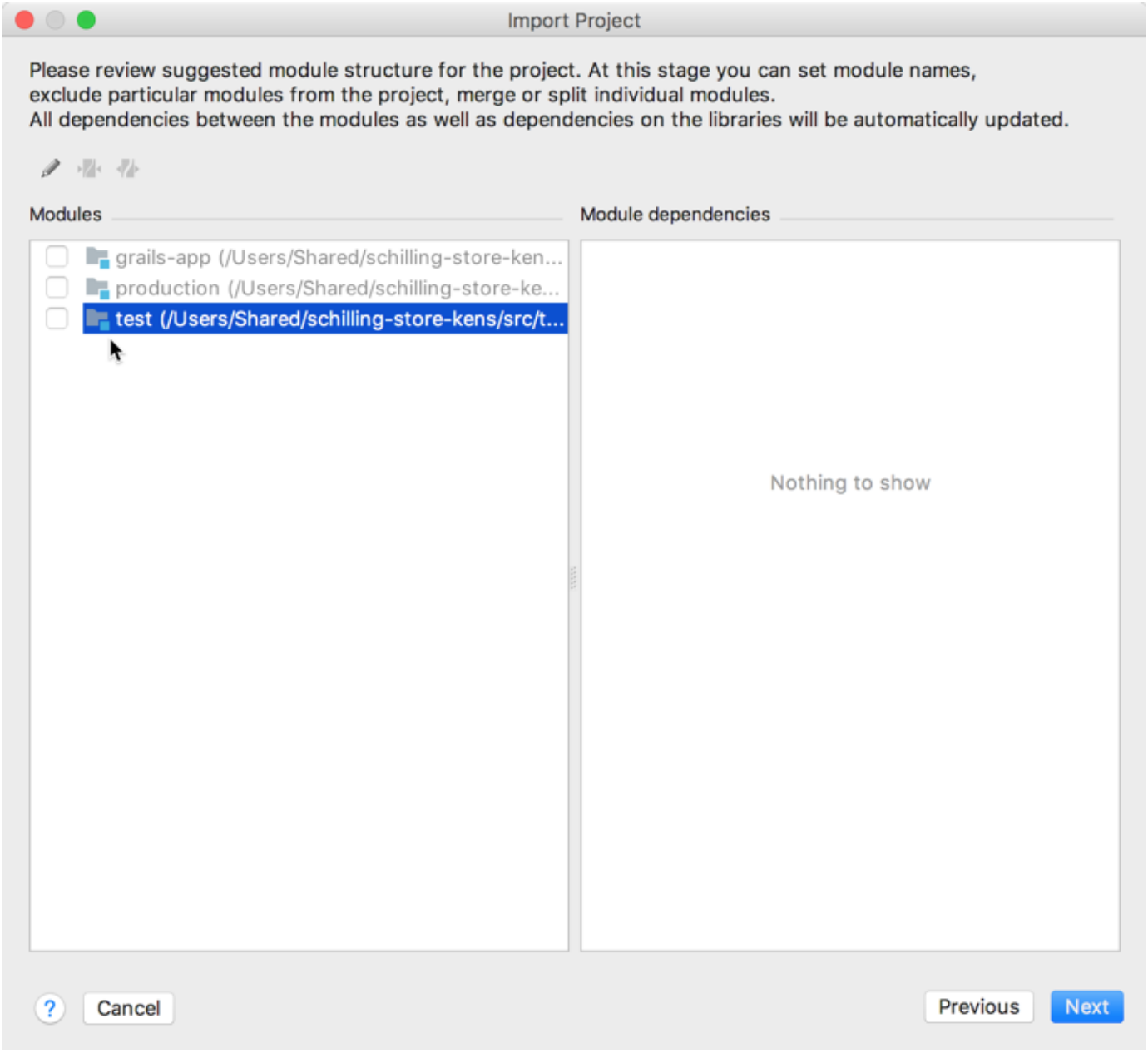


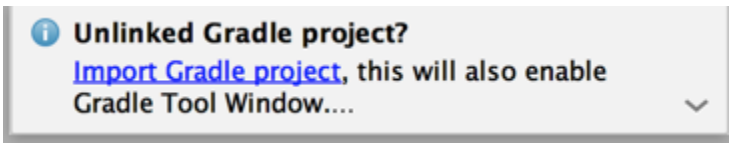

Import Project
Create project from existing sources
Import project from external model
Eclipse
Flash Builder
Gradle
Maven
Cancel
Previous
Next


Import Project
Project name:
schilling-store-kens
Project location:
/Users/Shared/schilling-store-kens
...
Cancel
Previous
Next


Import Project
Source files for your project have been found. Please choose directories that will
be added to the project roots. These paths correspond to default (root, unnamed, top level) packages.
Note: the program will recognize only those source files, that are located under these directories.
/Users/Shared/schilling-store-kens/grails-app/conf
Groovy
/Users/Shared/schilling-store-kens/grails-app/controllers
Groovy
/Users/Shared/schilling-store-kens/grails-app/domain
Groovy
/Users/Shared/schilling-store-kens/grails-app/init
Groovy
/Users/Shared/schilling-store-kens/out/production/resources
Groovy
/Users/Shared/schilling-store-kens/src/test/groovy
Groovy
Mark All
Unmark All
Cancel
Previous
Next


Import Project
Please review libraries found. At this stage you can set library names that will be used in the project, exclude particular libraries from the project, or move individual files between the libraries.
Libraries
Library contents
gradle-wrapper (/Users/Shared/schilling-stor...
grails-wrapper (/Users/Shared/schilling-store...
gradle-wrapper.jar (/Users/Shared/schilling-store-ken
Cancel
Previous
Next





Installing Grails on windows

This is for William, or others who want to read and decipher this. Will make this a better page when I have time.

Step-by-step guide

1. Go here: https://github.com/flofreud/posh-gvm/blob/master/README.md
2. When it complains about home, use this: https://www.mkyong.com/java/how-to-set-java_home-on-windows-10/

> Mostly for William. Ask him for clarification.

Related articles

- Clone Repository to disk from command line
- Implement Catgory Report
- Import a Project from a Zip File
- Bootstrapping
- Create a Controller

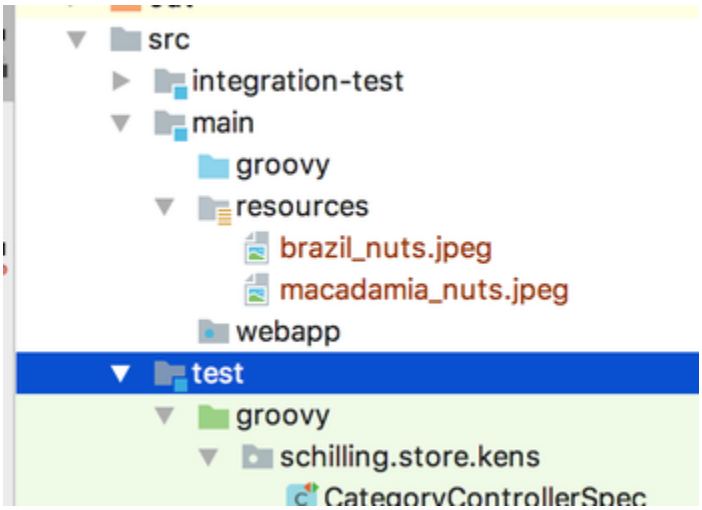
Running Tests

In grails, you can run the tests easily.

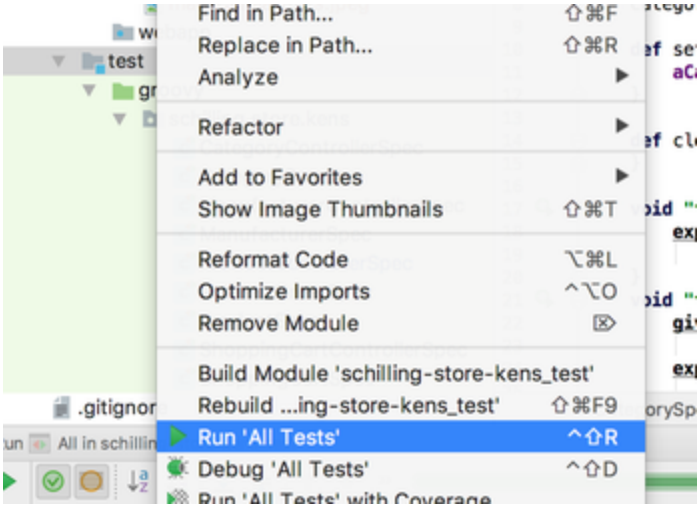
Step-by-step guide

The first time you run the tests:

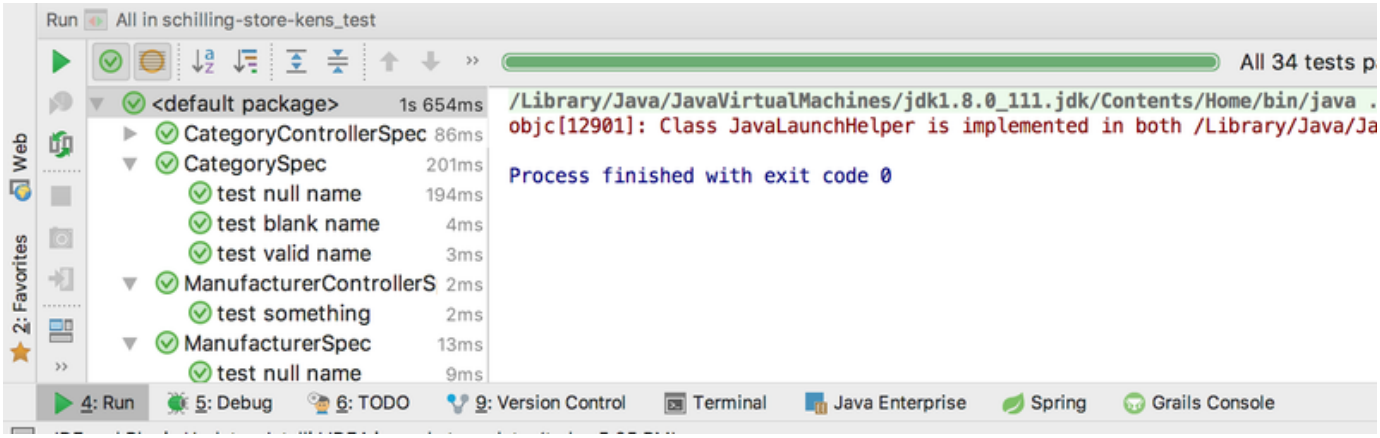
- Select the ***src/test*** folder



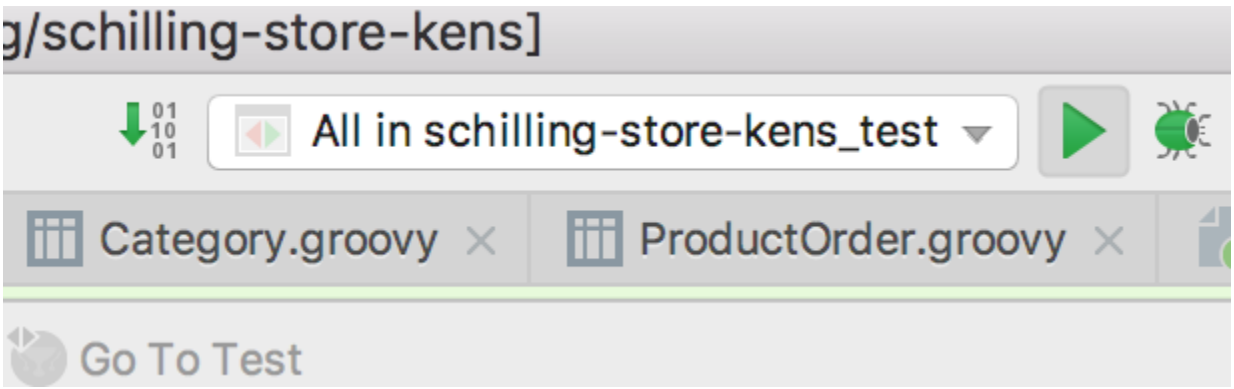
- Next right click and select ***Run All Tests***



- You will see the tests run in the bottom section.



You can run the tests again using the little run box at the top right:



Take a Screenshot

You will often want to save part of your screen. When creating documentation or troubleshooting problems screenshots come in handy.

Here are links to how to do that on windows and the mac.

Taking Screenshots on Windows 10

Taking Screenshots on the Mac

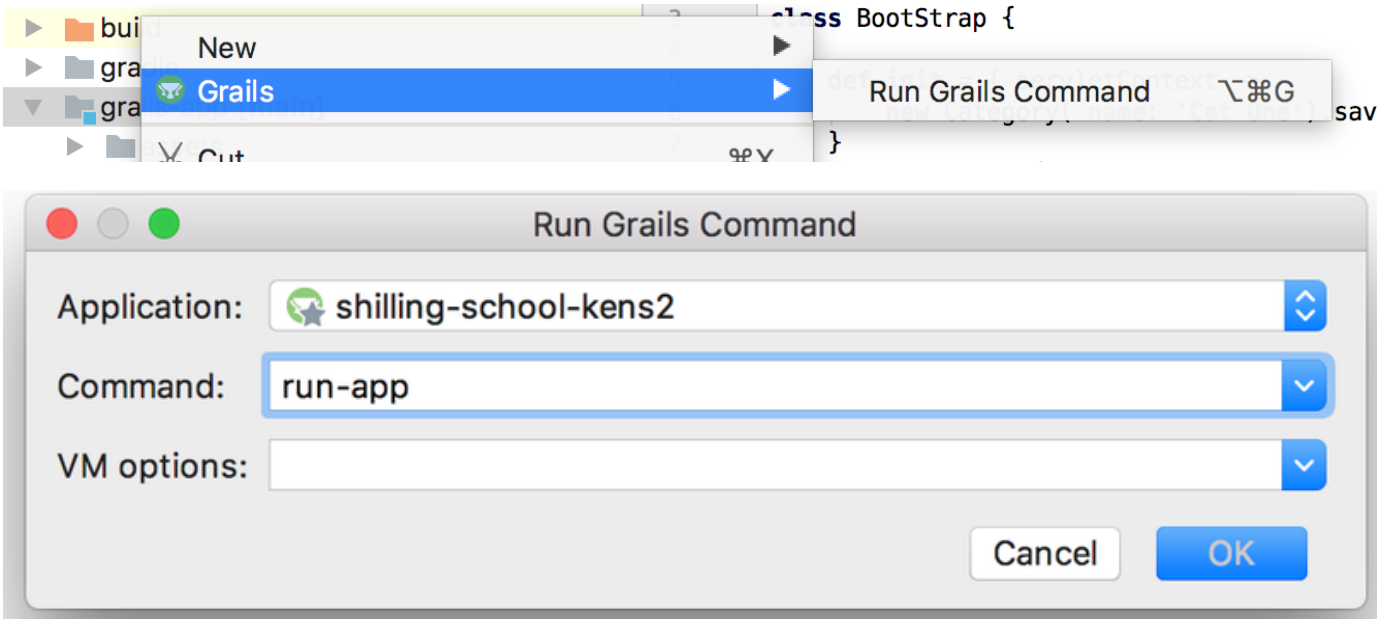
Using the Application UI Debug Console and Database Console to View and Update Data

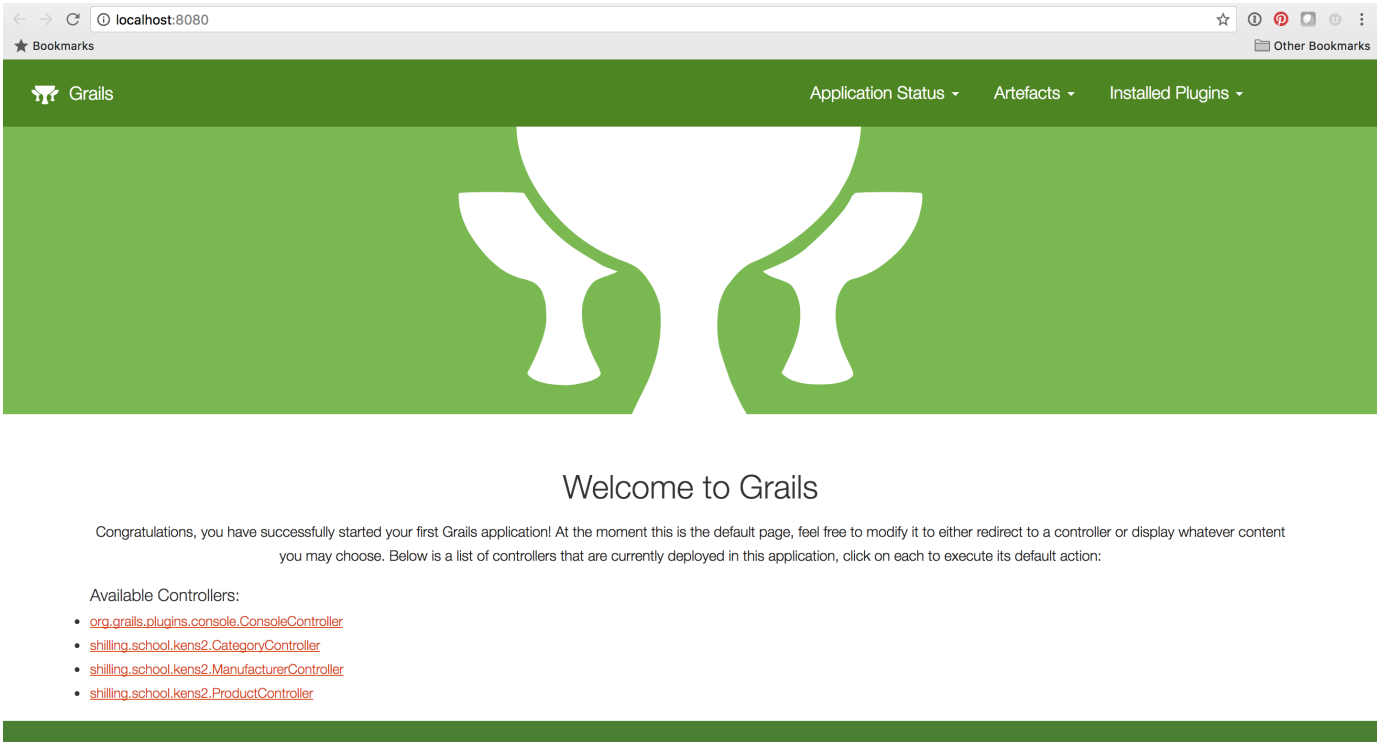
This page helps you get setup to work with the grails application using the user interface, the grails application console and the database console.

Working with the User Interface

Run the application by IDEA or the command line.

Using IDEA:





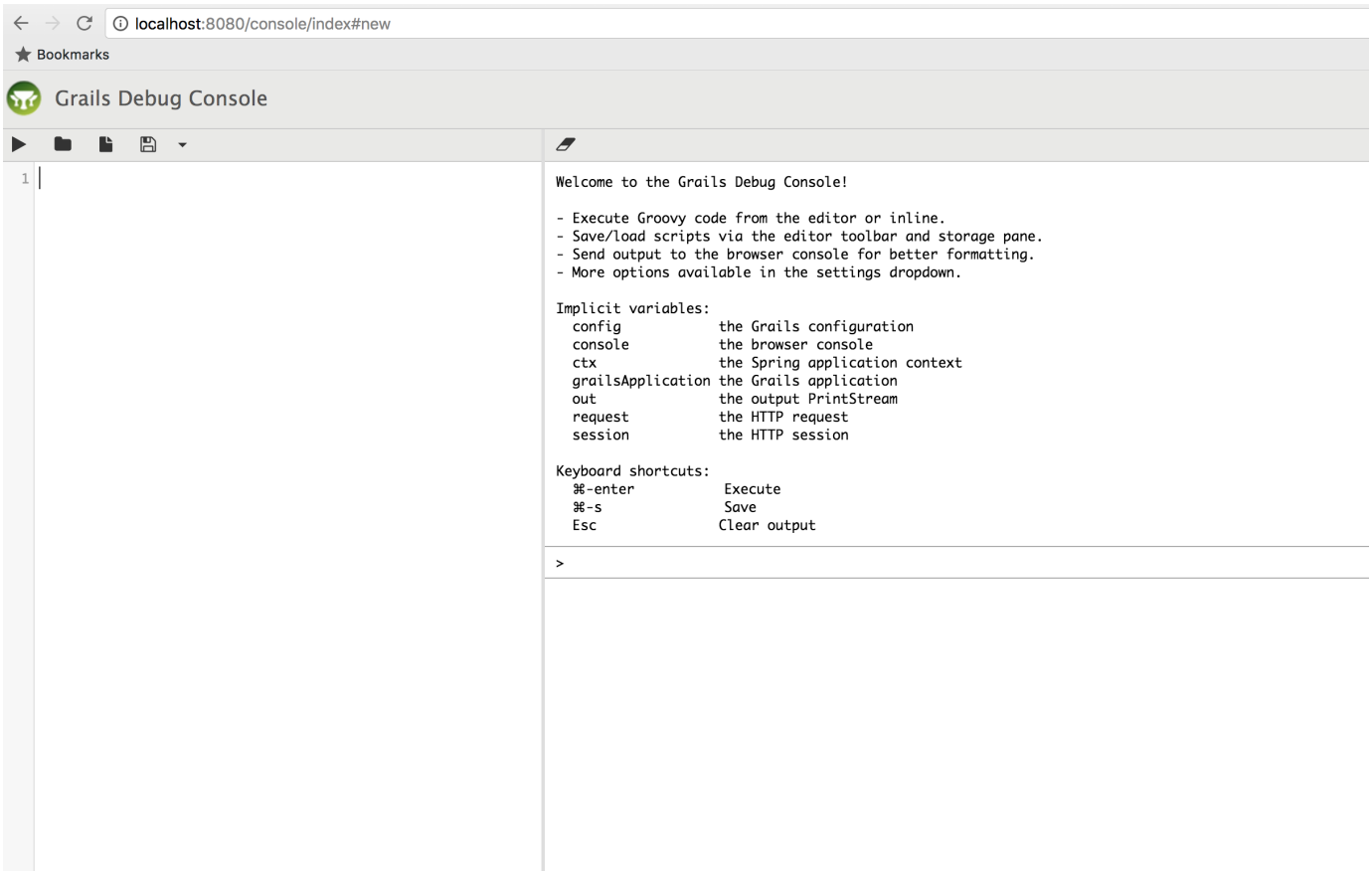
Use the controllers to manage the categories. Create new category, edit existing ones etc.

Use the Console

If you add this line to the ***build.gradle*** file in the root directory of your application, when you run the application you will see the ConsoleController in the list of controllers.

```
dependencies {
    compile "org.springframework.boot:spring-boot-starter-logging"
    compile "org.springframework.boot:spring-boot-autoconfigure"
    compile "org.grails:grails-core"
    compile "org.springframework.boot:spring-boot-starter-actuator"
    compile "org.springframework.boot:spring-boot-starter-tomcat"
    compile "org.grails:grails-web-boot"
    compile "org.grails:grails-logging"
    compile "org.grails:grails-plugin-rest"
    compile "org.grails:grails-plugin-databinding"
    compile "org.grails:grails-plugin-i18n"
    compile "org.grails:grails-plugin-services"
    compile "org.grails:grails-plugin-url-mappings"
    compile "org.grails:grails-plugin-interceptors"
    compile "org.grails.plugins:cache"
    compile "org.grails.plugins:async"
    compile "org.grails.plugins:scaffolding"
    compile "org.grails.plugins:events"
    compile "org.grails.plugins:hibernate5"
    compile "org.hibernate:hibernate-core:5.1.5.Final"
    compile "org.grails.plugins:gsp"
    console "org.grails:grails-console"
    profile "org.grails.profiles:web"
    runtime "org.glassfish.web:el-impl:2.1.2-b03"
    runtime "com.h2database:h2"
    runtime "org.apache.tomcat:tomcat-jdbc"
    runtime "com.bertramlabs.plugins:asset-pipeline-grails:2.14.2"
// add the following line
    runtime 'org.grails.plugins:grails-console:2.1.1'
    testCompile "org.grails:grails-gorm-testing-support"
    testCompile "org.grails.plugins:geb"
    testCompile "org.grails:grails-web-testing-support"
    testRuntime
"org.seleniumhq.selenium:selenium-htmlunit-driver:2.47.1"
    testRuntime "net.sourceforge.htmlunit:htmlunit:2.18"
}
```

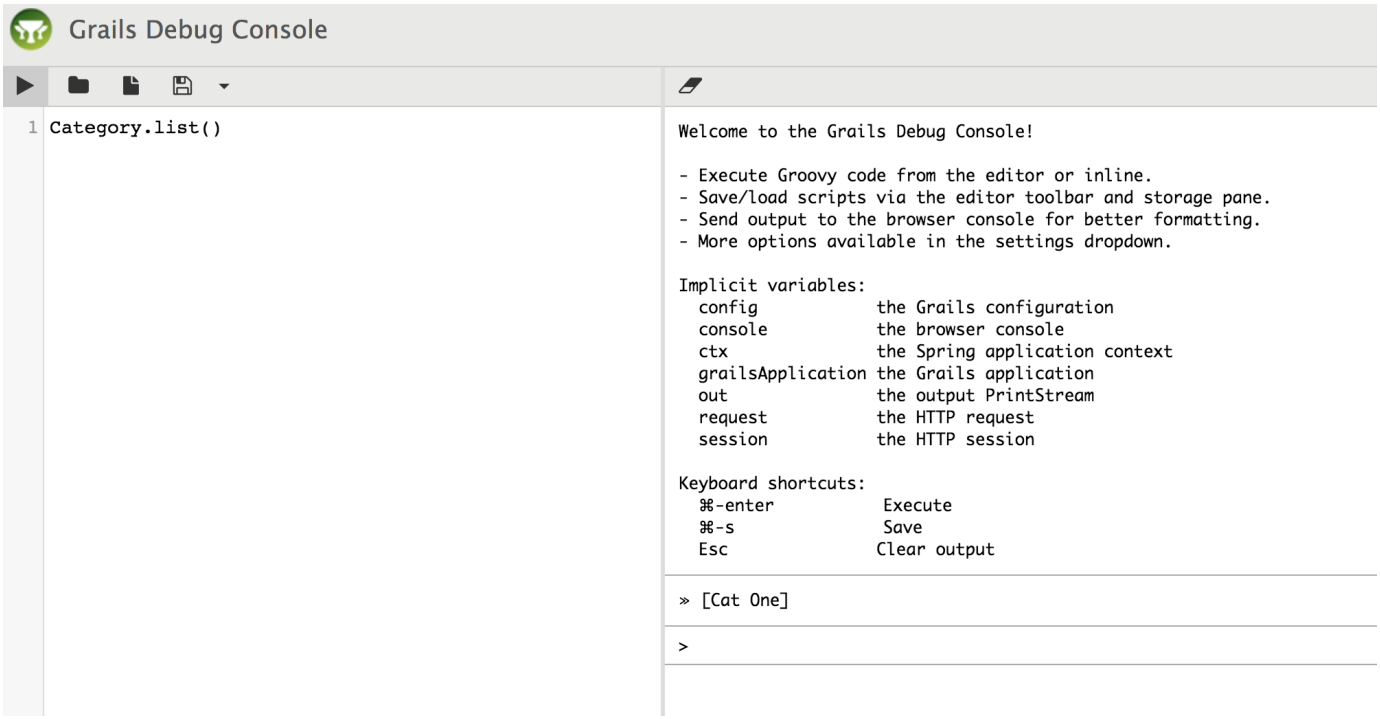
Open the console by clicking on the link to the ConsoleController.



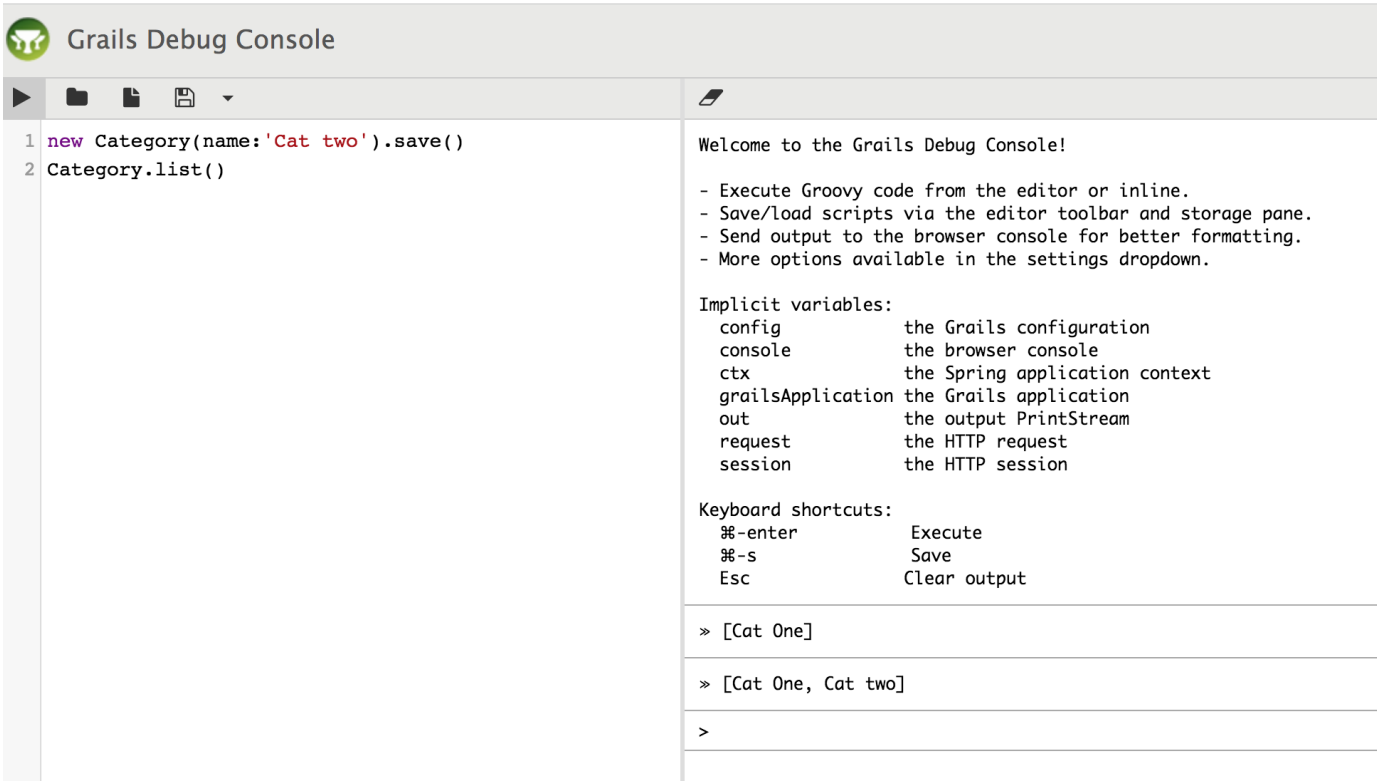
Use the following code to:

To do this	Run this
Create a category	new Category(name:'Category One').save()
List the categories	Category.list()
Get a specific category by ID	Category.get(2)
Get a specific category by name	Category.findByName('Category One')
Update a category	def cat = Category.get(2) cat.name = 'Cat Too' cat.save()

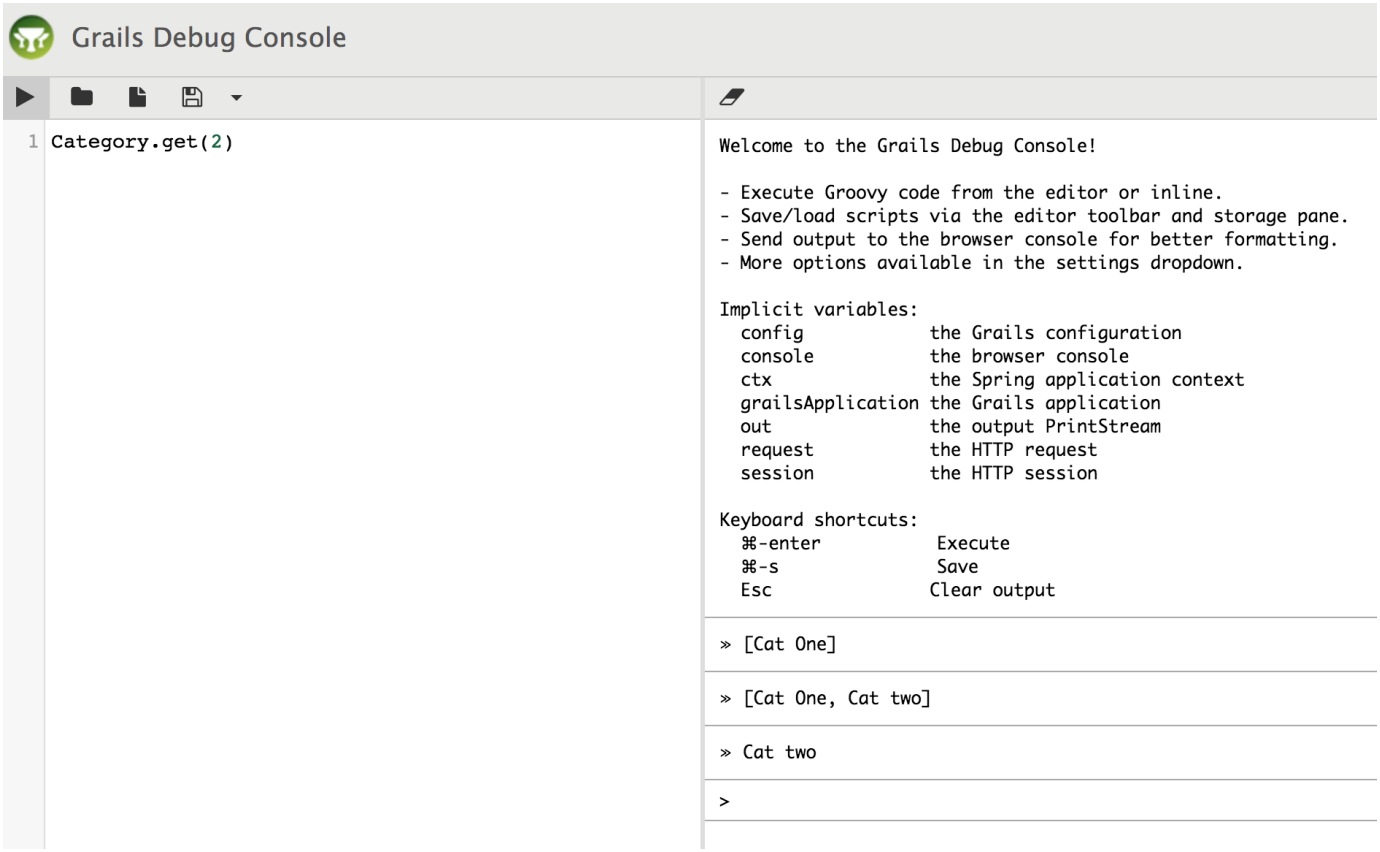
Create a category



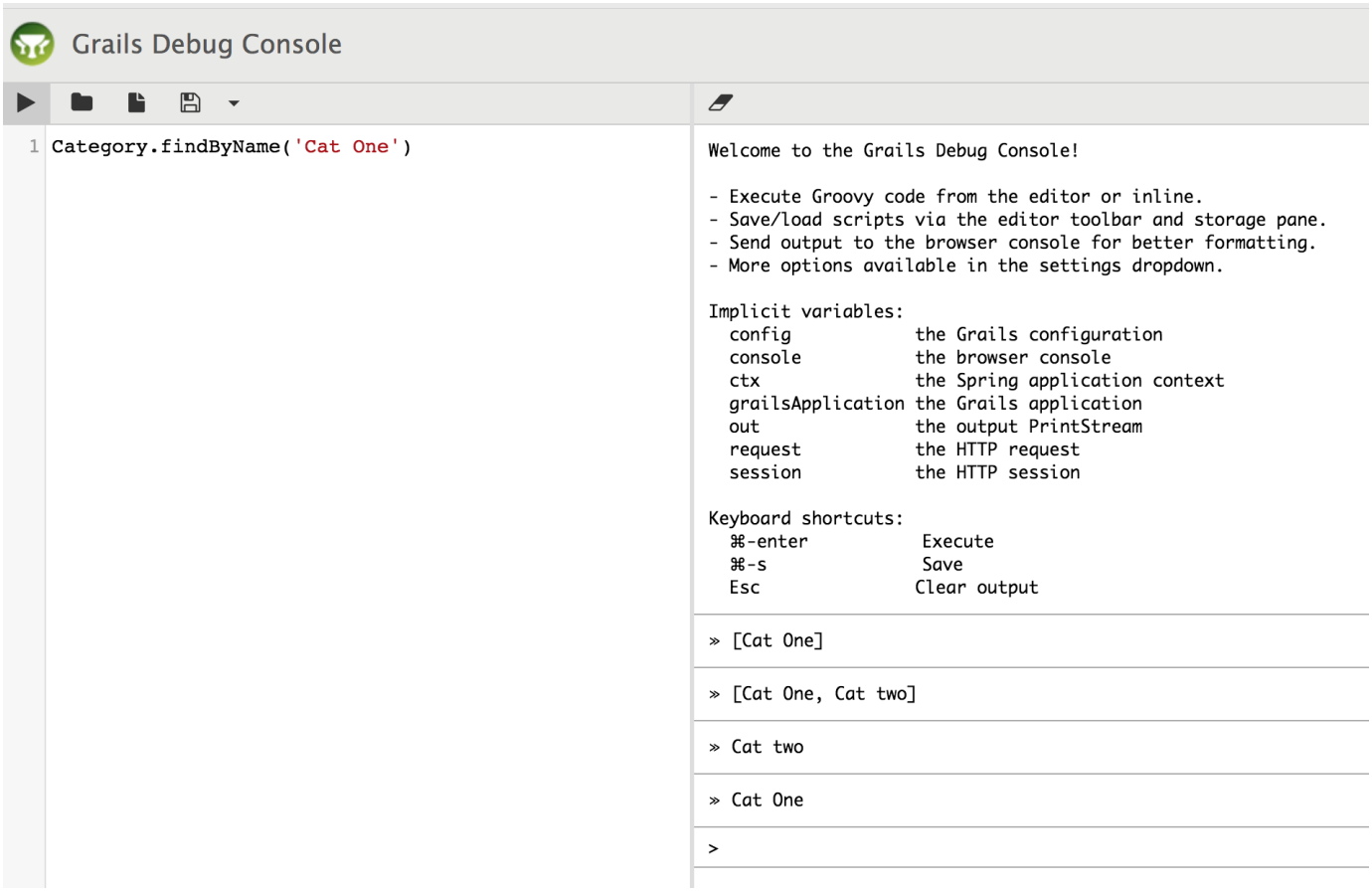
List the categories



Get a specific category by ID



Get a specific category by name



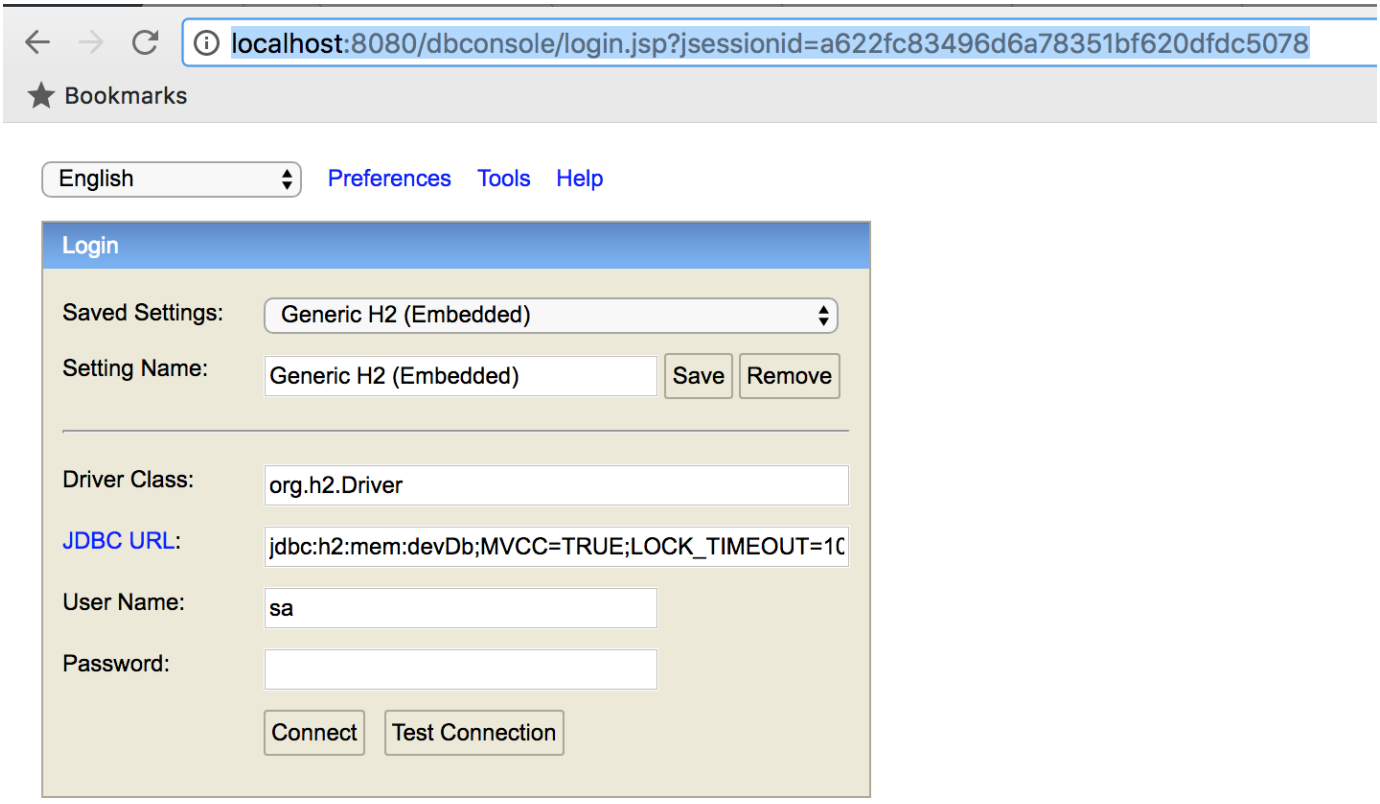
Update a category

```
1 def cat = Category.get(2)
2 cat.name = 'Cat Too'
3 cat.save()
4 Category.list()
```

```
Welcome to the Grails Debug Console!

- Execute Groovy code from the editor or inline.
- Save/load scripts via the editor toolbar and storage pane.
- Send output to the browser console for better formatting.
- More options available in the settings dropdown.

Implicit variables:
  config             the Grails configuration
  console            the browser console
  ctx                the Spring application context
  grailsApplication  the Grails application
  out                the output PrintStream
  request            the HTTP request
  session            the HTTP session

Keyboard shortcuts:
  ⌘-enter            Execute
  ⌘-s                Save
  Esc                Clear output

» [Cat One]

» [Cat One, Cat two]

» Cat two

» Cat One

» [Cat One, Cat Too]

>
```

Interact with the Database using the Database Console

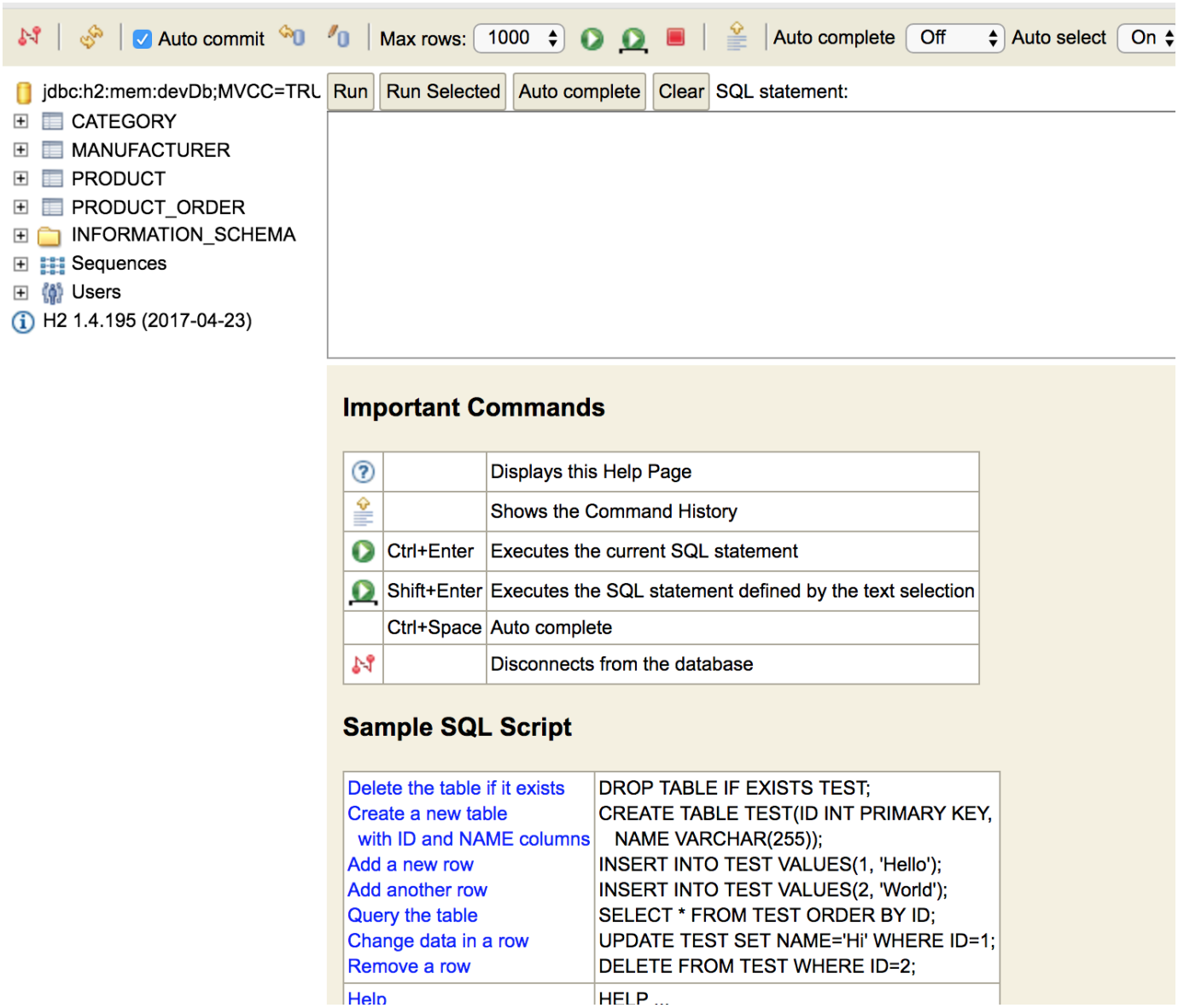
Open the following url: http://localhost:8080/dbconsole



Enter this in the JDBC URL field: ***jdbc:h2:mem:devDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE***

Enter ***sa*** for the user name and leave the password blank.

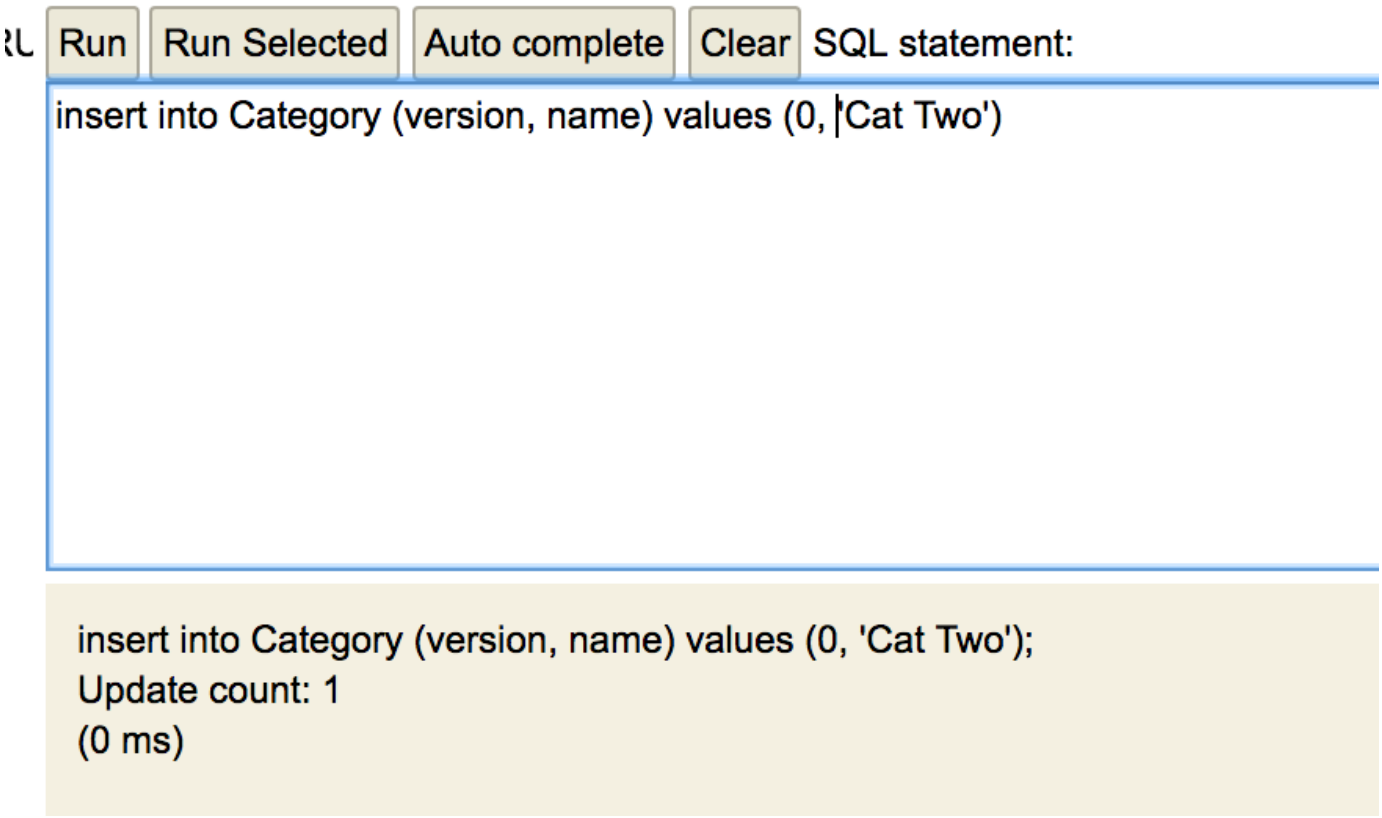
Click the ***Connect*** button



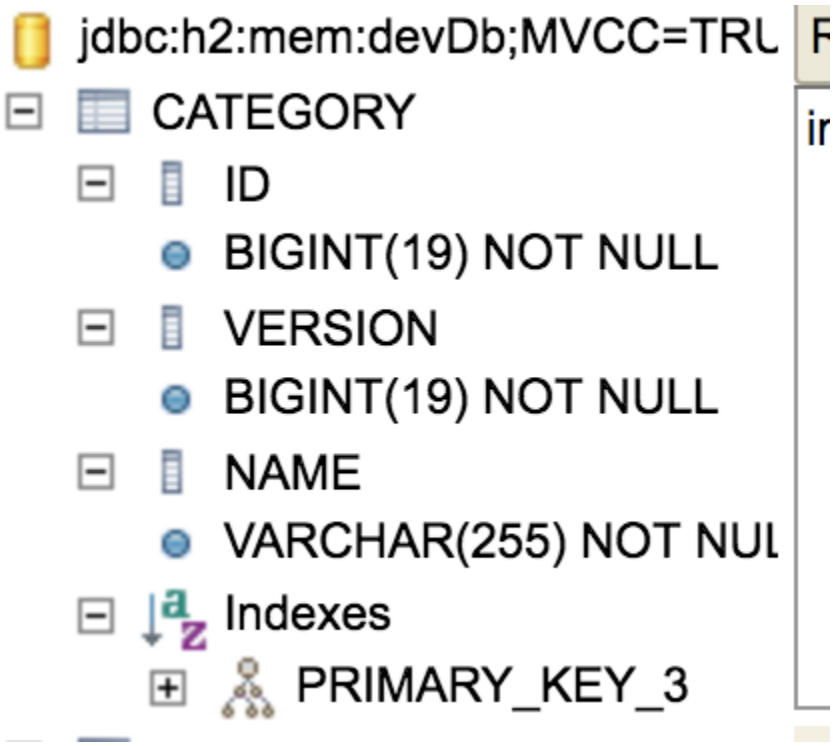
Use the following SQL code to:

To do this	Run this
Create a category	insert into Category (version, name) values (0, 'Cat Two')
List the categories	select * from Category
Get a specific category by ID	select * from Category where ID = 2
Get a specific category by name	select * from Category where NAME = 'Cat One'
Update a category	update Category set name = 'Cat Too' where ID = 2

Create a Category

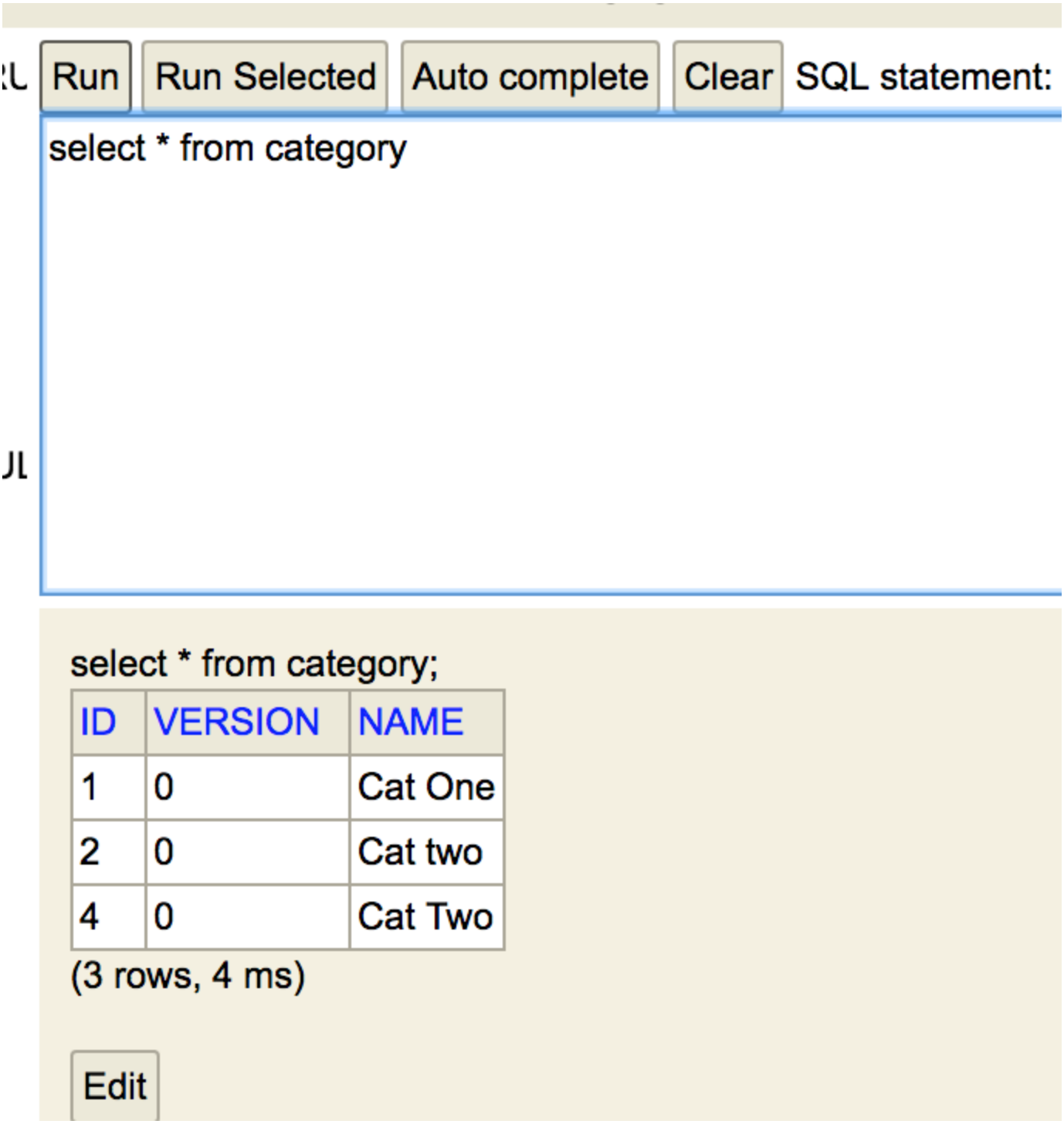


Why do we have to put in version? Take a look at the table:

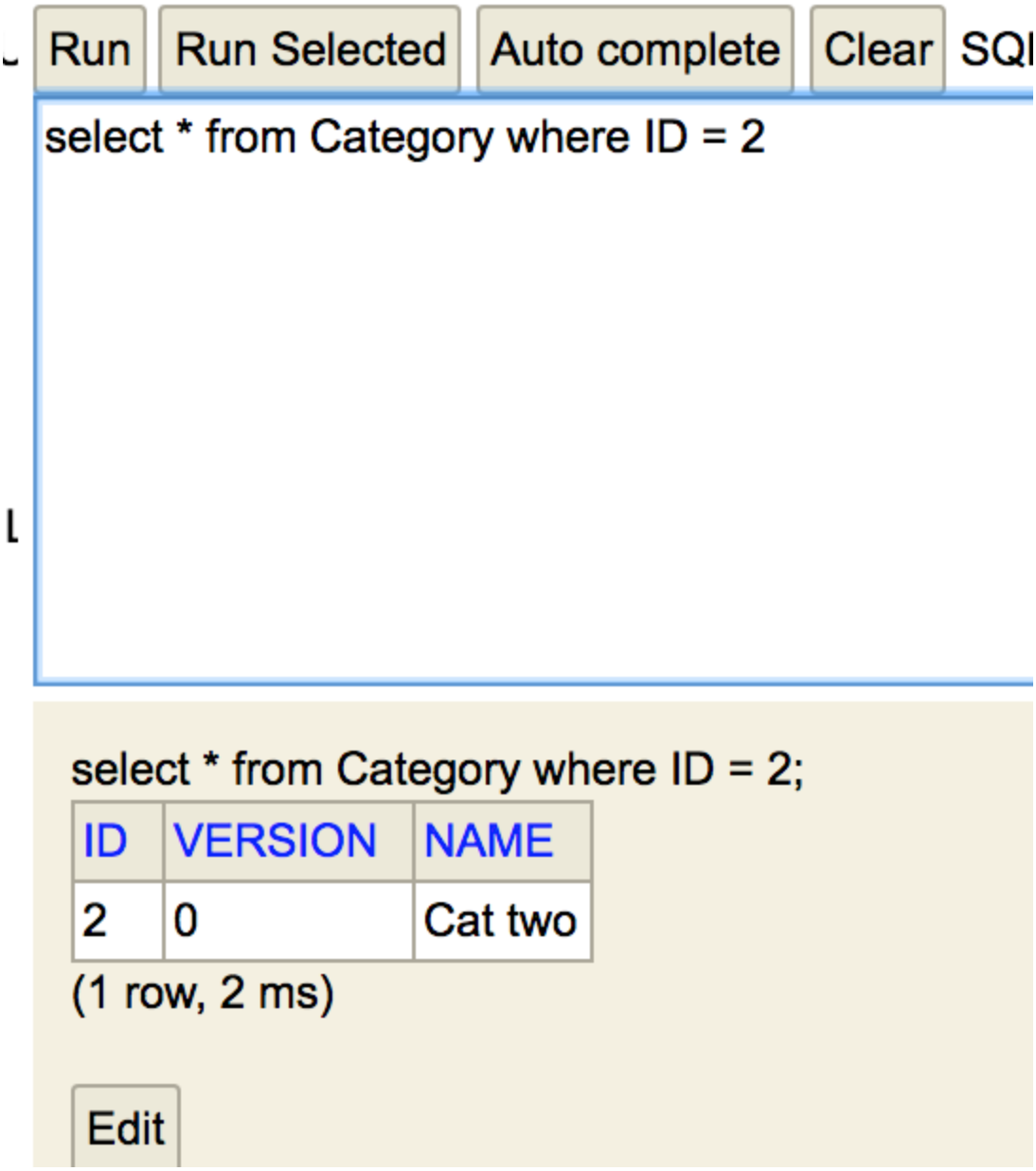


There is an extra column that Grails put in there that we need to account for. Don't ask why, we will learn later about what that does for us.

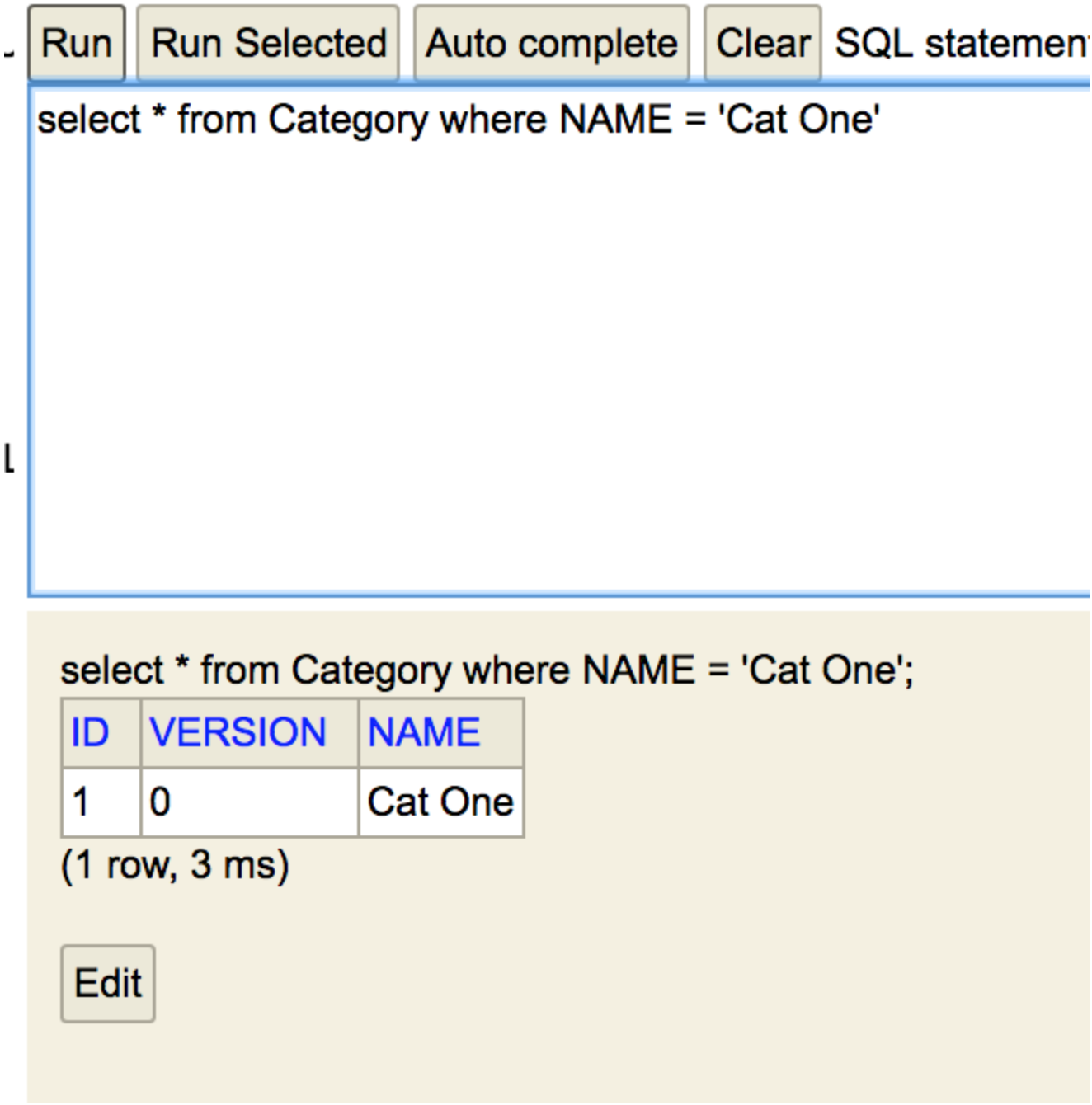
List the Categories



Get a Specific Category by ID



Get a Specific Category by Name

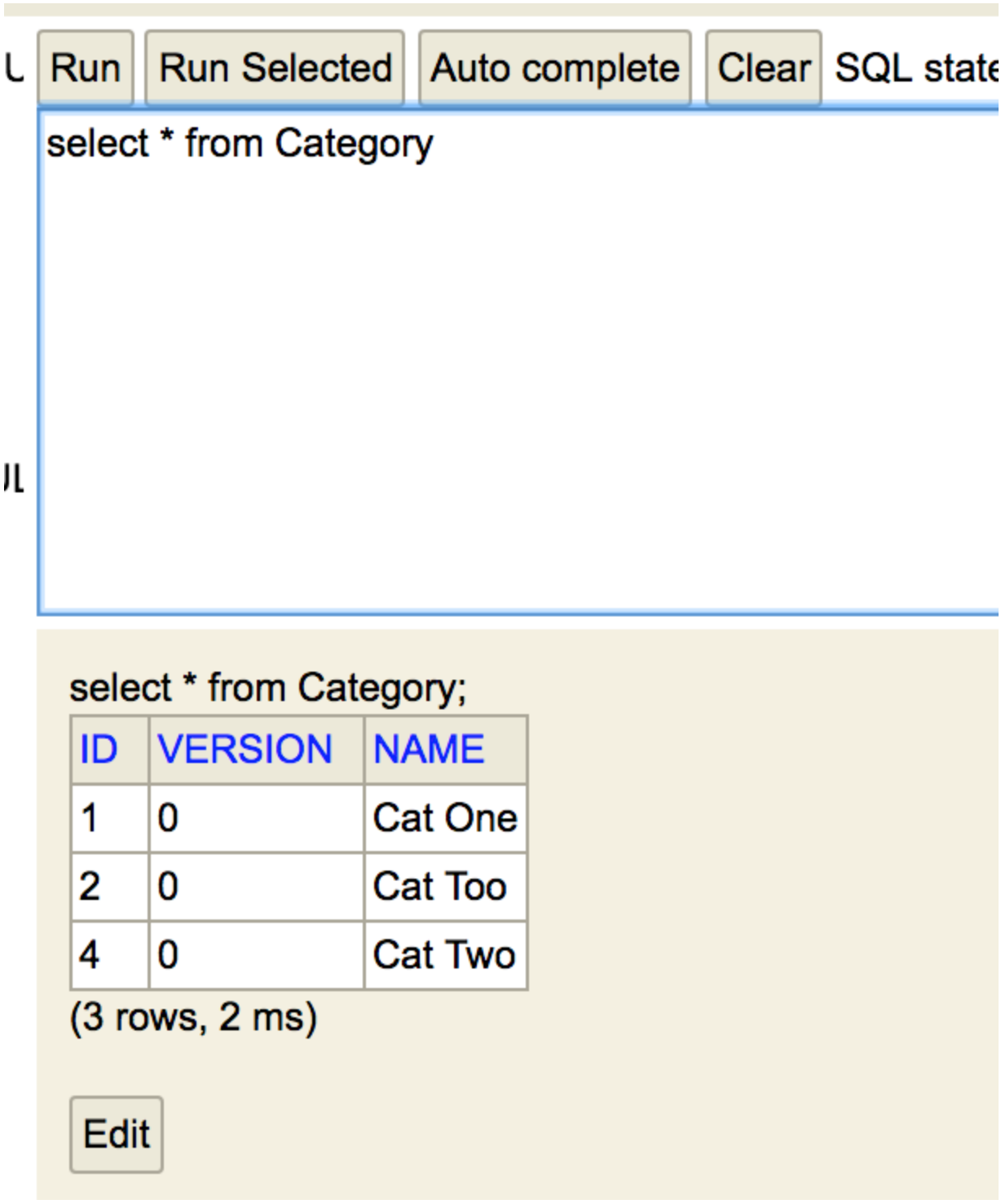


Update a Category

Run | Run Selected | Auto complete | Clear | SQL statement:

```
update Category set name = 'Cat Too' where ID = 2
```

```
update Category set name = 'Cat Too' where ID = 2;
Update count: 1
(3 ms)
```



Working with Grails Domain Classes

Domain classes are used to hold the persistent information. An example of a domain class in the pet store is Pet.

Domain classes have instance variables, relationships, constraints and methods.

Step-by-step guide

Define Instance Variables

1. For each variable, add a line with the type and name of the variable
2. Initialize it if you like

Define constraints

1. For each variable define it's constraints
2. Use this documentation: http://docs.grails.org/latest/ref/Constraints/Usage.html

Define relationships:

1. For each relationship, define a variable or use the hsMany or hasOne block
2. See the "Domain Classes section of the doc: http://docs.grails.org/latest/guide/GORM.html

```
class Author {

    String name

    static hasMany = [books: Book]
}
```

Here is a sample:

```
package petstore

class Pet {

    String species
    String name
    String description
    Integer age
    Integer weight

    static constraints = {
        name nullable: false, blank: false
        species nullable: false
        description nullable: false
        age nullable: false, min: 0
    }

    String toString() {
        "$name $species $age"
    }
}
```

Implement Catgory Report

This how to will cover creating a report of categories.

Our report should:

- be accessible from a link on the category index/list page
- show the categories and their associated products
- have a link back to the list page

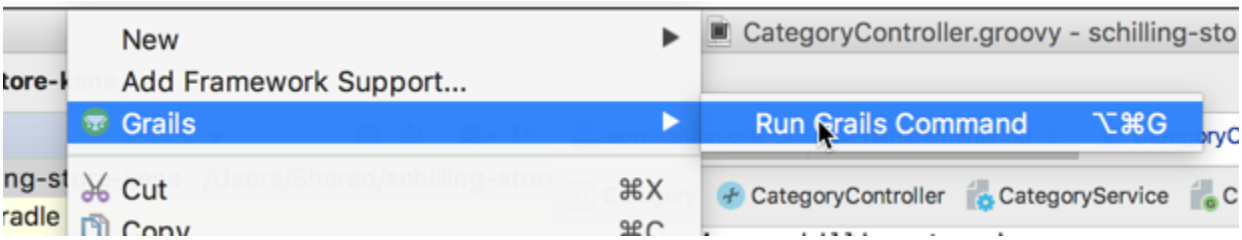
Step-by-step guide

This is how we do it

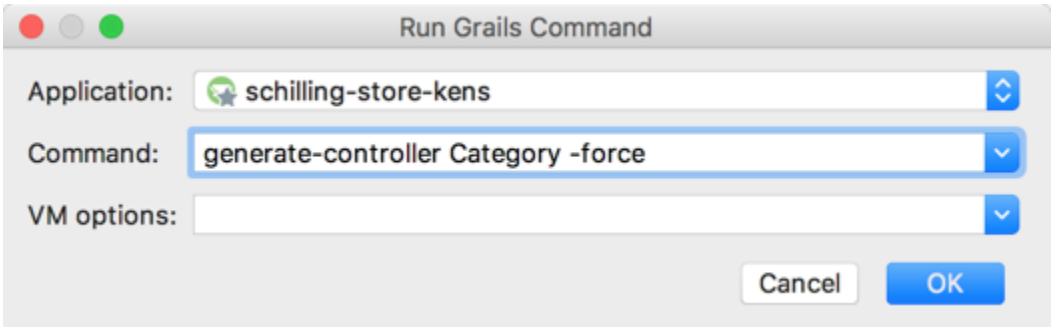
Generate the category controller. This will remove the scaffold line and replace it with the code that represents what was happening.

```
package schilling.store.kens

class CategoryController {

    static scaffold = Category
}
```

Use grails run command:



Execute the command: ***generate-controller Category -force***



This will generate the code for the controller and open it in the editor. The Controller should look something like this:

CategoryController

```
package schilling.store.kens

import static org.springframework.http.HttpStatus.*
import grails.transaction.Transactional

@Transactional(readOnly = true)
class CategoryController {
```

```
    static allowedMethods = [save: "POST", update: "PUT", delete:
"DELETE"]

    def index(Integer max) {
        params.max = Math.min(max ?: 10, 100)
        respond Category.list(params), model:[categoryCount:
Category.count()]
    }

    def show(Category category) {
        respond category
    }

    def create() {
        respond new Category(params)
    }

    @Transactional
    def save(Category category) {
        if (category == null) {
            transactionStatus.setRollbackOnly()
            notFound()
            return
        }

        if (category.hasErrors()) {
            transactionStatus.setRollbackOnly()
            respond category.errors, view:'create'
            return
        }

        category.save flush:true

        request.withFormat {
            form multipartForm {
                flash.message = message(code: 'default.created.message',
args: [message(code: 'category.label', default: 'Category'),
category.id])
                redirect category
            }
            '*' { respond category, [status: CREATED] }
        }
    }

    def edit(Category category) {
        respond category
    }

    @Transactional
```

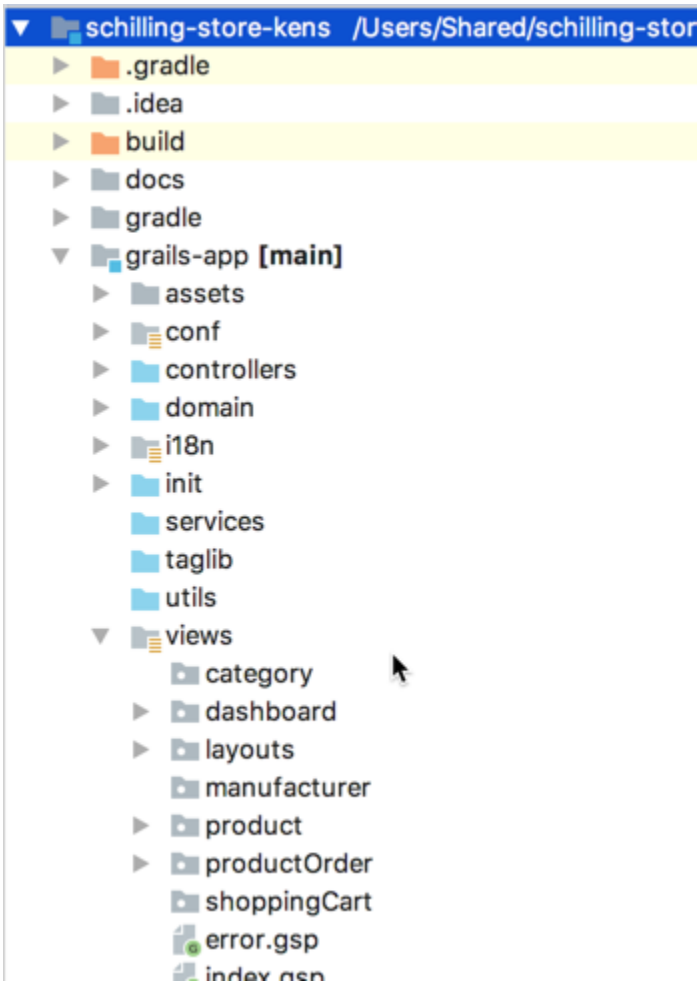
```
    def update(Category category) {
        if (category == null) {
            transactionStatus.setRollbackOnly()
            notFound()
            return
        }

        if (category.hasErrors()) {
            transactionStatus.setRollbackOnly()
            respond category.errors, view:'edit'
            return
        }

        category.save flush:true

        request.withFormat {
            form multipartForm {
                flash.message = message(code: 'default.updated.message',
args: [message(code: 'category.label', default: 'Category'),
category.id])
                redirect category
            }
            '*'{ respond category, [status: OK] }
        }
    }

    @Transactional
    def delete(Category category) {

        if (category == null) {
            transactionStatus.setRollbackOnly()
            notFound()
            return
        }

        category.delete flush:true

        request.withFormat {
            form multipartForm {
                flash.message = message(code: 'default.deleted.message',
args: [message(code: 'category.label', default: 'Category'),
category.id])
                redirect action:"index", method:"GET"
            }
            '*'{ render status: NO_CONTENT }
        }
    }

    protected void notFound() {
        request.withFormat {
```

```
            form multipartForm {
                flash.message = message(code:
'default.not.found.message', args: [message(code: 'category.label',
default: 'Category'), params.id])
                redirect action: "index", method: "GET"
            }
            '*'{ render status: NOT_FOUND }
        }
```

```
        }
    }
```

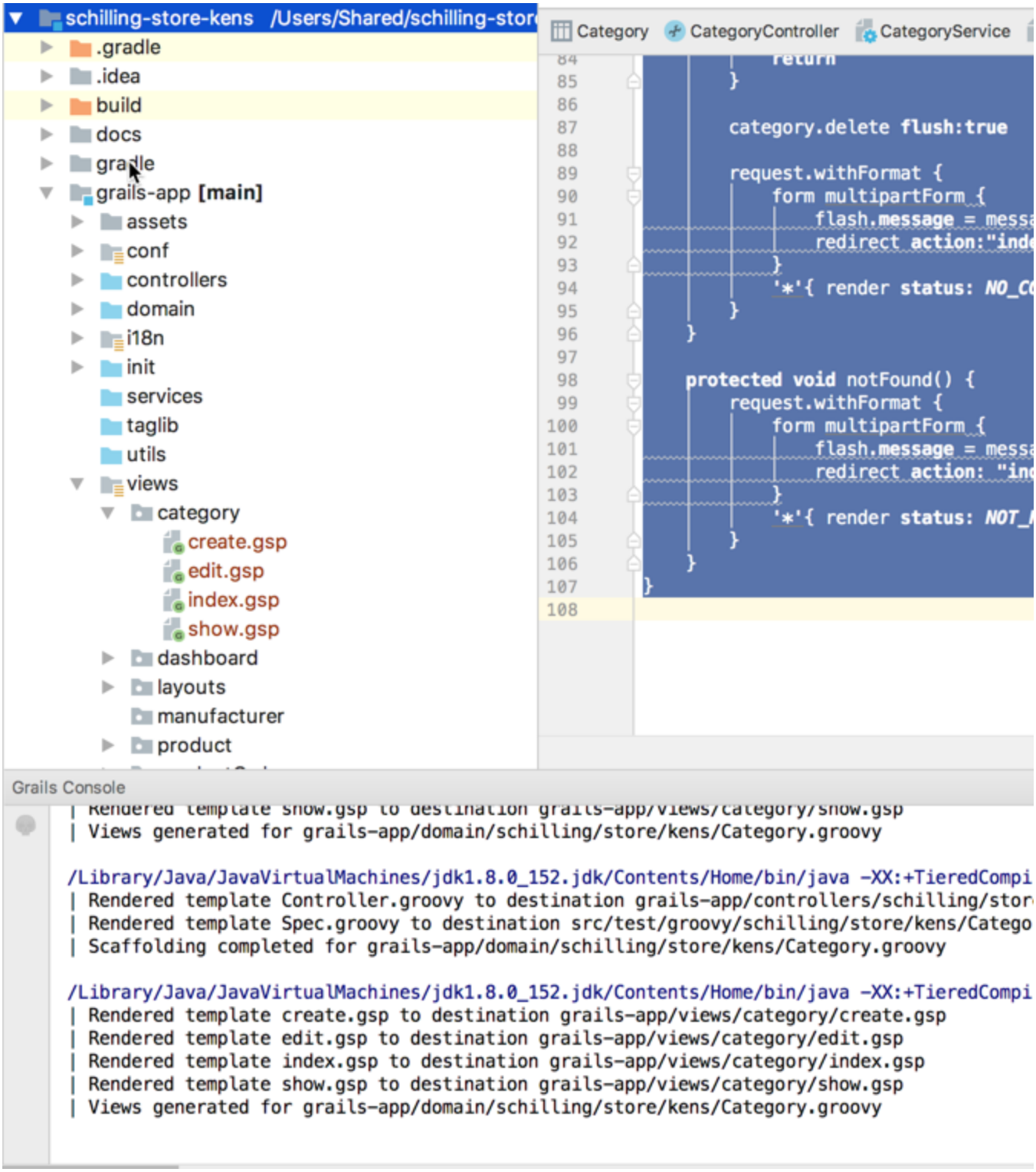
You will notice that there is a method for each of the actions we do in the user interface. index, show, update, delete. These show in the URL as you move around in the app. localhost:8080/category/index...

Next we'll generate the views. If you look at the views directory, you will see a subdirectory called ***category*** that has no files in it.



Once you execute the following code, you will see that there are a number of new files there.

generate-views Category -force



We now have a .gsp file for each of the methods in our controller. create, edit, index, show

In order to create our report, we'll add a new file based on the index.gsp to the views directory and a new method to our controller.

First, the new method in the controller. It should look like this:

```
    def report() {
        respond Category.list(), model:[categoryCount: Category.count()]
    }
```

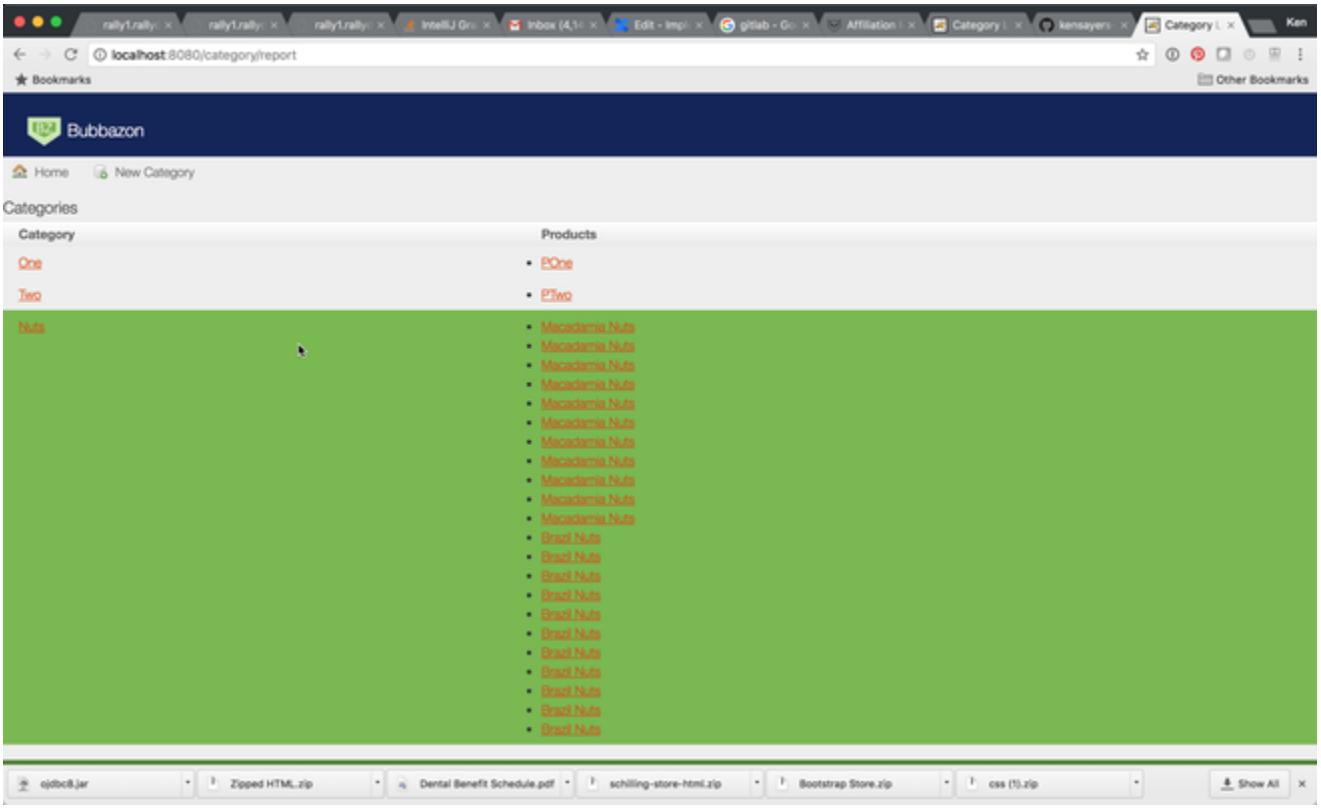
Now add a new file in the ***views/category*** directory named ***report.gsp***

The code should look like this:

```
<!DOCTYPE html>
<html>
    <head>
        <meta name="layout" content="main" />
        <g:set var="entityName" value="${message(code: 'category.label',
default: 'Category')}" />
        <title><g:message code="default.list.label" args="[entityName]"
/></title>
    </head>
    <body>
        <a href="#list-category" class="skip" tabindex="-1"><g:message
code="default.link.skip.label" default="Skip to content…"/></a>
        <div class="nav" role="navigation">
            <ul>
                <li><a class="home" href="${createLink(uri:
'/')}"><g:message code="default.home.label"/></a></li>
                <li><g:link class="create" action="create"><g:message
code="default.new.label" args="[entityName]" /></g:link></li>
            </ul>
        </div>
        <div id="list-category">
            <h1>Categories</h1>
            <table>
                <tr><th>Category</th><th>Products</th></tr>
                <g:each var="category" in="${categoryList}">
                    <tr>
                        <td>
                            <g:link class="show" action="show"
resource="${category}">${category.name}</g:link>
                        </td>
                        <td>
                            <ul>
                                <g:each var="product"
in="${schilling.store.kens.Product.findAllByCategory(category)}">
                                    <li><g:link class="show"
action="show" resource="${product}">${product.name}</g:link></li>
                                </g:each>
                            </ul>
                        </td>
                    </tr>
                </g:each>
            </table>
        </div>
    </body>
</html>
```

Now, if you run the app, you can put in this url and get to the report: ***localhost:8080/category/report***

You will see something lke this:



Next we need to add the link from the index page.

Open the ***views/category/index.gsp*** file in the editor and add this line:

```
<body>
    <a href="#list-category" class="skip" tabindex="-1"><g:message code="default.link.skip.label" default="Skip to content&hellip;"/><
    <div class="nav" role="navigation">
        <ul>
            <li><a class="home" href="${createLink(uri: '/')}"><g:message code="default.home.label"/></a></li>
            <li><g:link class="create" action="create"><g:message code="default.new.label" args="[entityName]" /></g:link></li>
            <li><g:link class="report" action="report">Report</g:link></li>
        </ul>
    </div>
    <div id="list-category" class="content scaffold-list" role="main">
```

Related articles

- Clone Repository to disk from command line
- Implement Catgory Report
- Import a Project from a Zip File
- Bootstrapping
- Create a Controller

Product requirements

Add Product requirements

Title
No content found.

Resources

Category	Item	Comment
Grails Programming	Grails Documentation	Documents the grails framework and echo system.
Groovy Programming	Groovy Tutorial	Tutorial to learn how to program in groovy.
Groovy Programming	Online Groovy Web Console	Online tool for trying out groovy code.
HTML	HTML Tutorial	Start to finish walkthrough of all HTML elements.
HTML	HTML Reference	Information about all the HTML facilities.
HTML	HTML Examples	Examples of uses of HTML.
CSS	CSS Tutorial	Start to finish walkthrough of CSS.
CSS	CSS Reference	Information about all the CSS capabilities and features.
CSS	CSS Examples	Examples that illustrate uses of CSS
Bootstrap CSS	Bootstrap Tutorial	Start to finish walkthrough of Bootstrap
Bootstrap Template	Boostrap Template Zipfile	Unzip this and then open the project in IDEA.
Mockups	Balsamiq Tutorial	Tutorial on using Balsamiq
Mockups	Balsamiq Documentation	Reference Documentation for Balsamic
Mockups	Another Balsamiq Tutorial	A great tutorial from lynda.com.
SQL	SQL Tutorial	A great introduction to SQL with interactive exercises.
Podcast	Under the Radar	Great podcast about all the aspects of developing for iOS.
MVC	MVC Tutorial Video	A quick video explaining mvc in grails
Web Server	What a web application does	Short video explaining what a web app does and little about how.
Web Concepts	Web Concepts Video	Discussing some of the main concepts to help contextualize what we are building.
API	Introduction to APIs	Short discussion of what an API is and why they are so widely used.
API	Google Maps API Tutorial	How to call the Google Maps API.
API	Google API Basics	Shows how to get an API key.
API	Autocomplete API	Enabling the auto completion and using it.
Version Control	Git Reference	The official reference for git
Version Control	Interactive Git Tutorial	Excellent introduction to using git from the command line.

Version Control	GIT Tutorials Part 1 Part 2 Part 3 Part 4	Four part tutorial on git with installation and hands on.
Version Control	Pull Requests	Reference on how to do a pull request.
Version Control	Merging	Reference on how to merge branches

File	Modified
startbootstrap-heroic-features-gh-pages.zip	Oct 30, 2017 by Ken Sayers
CreateData.txt	Nov 30, 2017 by Ken Sayers
schilling-store-kens.zip	Dec 06, 2017 by Ken Sayers

Drag and drop to upload or browse for files

Download All

Roadmap

- Use Cases
  - Administrator
    - Manage Products
    - Manage Manufacturers
    - Manage Categories
    - Manage Orders
    - Manage Customers
    - Bill Customers
  - Customer
    - Manage Orders
    - Manage Account
    - Track Orders
- Other Requirements
- Architecture Roadmap
- On the Radar

Use Cases

Administrator

Manage Products

- [x] Create Products
- [x] Show Products
- [x] Update Products

- [x] Delete Products

Manage Manufacturers

- [x] Create Manufacturers
- [x] Show Manufacturers
- [x] Update Manufacturers
- [x] Delete Manufacturers

Manage Categories

- [x] Create Categories
- [x] Show Categories
- [x] Update Categories
- [x] Delete Categories

Manage Orders

- [x] Create Orders
- [x] Show Orders
- [x] Update Orders
- [x] Delete Orders
- [ ] Report of Orders

Manage Customers

- [ ] Create Customers
- [ ] Show Customers
- [ ] Update Customers
- [ ] Delete Customers

Bill Customers

- [ ] Capture Payment Information
- [ ] Execute Transaction

Customer

Manage Orders

- [ ] Create Orders
- [ ] Show Orders
- [ ] Update Orders
- [ ] Delete Orders
- [ ] Report of Orders

Manage Account

- [ ] Create Account
- [ ] Show Account
- [ ] Update Account
- [ ] Delete Account
- [ ] Login
- [ ] Logout

Track Orders

- [ ] Show status

Other Requirements

- [x] Bootstrap the Data
- [x] Setup the Database
- [ ] Style the Store
- [ ] Make Website Responsive

Architecture Roadmap

- [ ] Create Simple Web App
- [x] Enable API
- [ ] Test API
- [ ] Create Single Page Application - Just for Customers to Use
- [ ] Create Mobile Application - Just for Customers to Use

On the Radar

Version Control

- [ ] Sign up for GitHub
- [ ] Clone the Shared Project
- [ ] Create a Branch
- [ ] Contribute your File
- [ ] Request a Merge
- [ ] Pull from HEAD

APIs

- [ ] Google Maps API Tutorial
- [ ] Create Grails App that uses Maps API
- [ ] Testing APIs with Postman
- [ ] Test Google Maps API with Postman
- [ ] Test Grails service with Postman

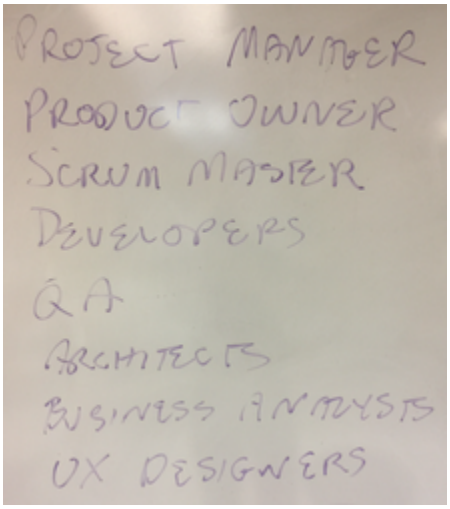
User API

- [ ] Implement User API
- [ ] Test User API with Postman

Mobile App

- [ ] Register
- [ ] Login
- [ ] Product List
- [ ] ...

Scrum



Setup

Setup Balsamiq

Balsamiq is software for mocking up user interfaces.

Here is the information we got from the company. Follow the instructions in the FAQ linked below.

> **Balsamiq License**
> Greets to Cincinnati from San Francisco!
>
> Thanks for helping to rid the world of bad software! We're honored that you'd like to use Balsamiq Mockups in your program.
>
> We've issued you a trial license for you and your students to use on your home and school computers during the duration of your class.
>
> Please keep in mind that this trial license should only be shared with other Faculty Members and Students, and should not be posted publicly.
>
> You can also use the license to register Mockups 3 for Desktop on computer lab machines.
>
> Product: Mockups 3 for Desktop
>
> LICENSE INFORMATION:
> License Name: Computer Science Principles 2017
> License Key: eJzzzU/OLi0odswsqnHOzy0oLUktUghOzkzNS05VCCjKzEvOLMhJLVYwMjA0rzE0NTI1sjQ0MgCBmpAaQwC00RRL
>
> License End Date: Jun 01, 2018
>
> Be sure to copy and paste the License Name and License Key exactly as shown above.
>
> Here is an FAQ to get you started using Balsamiq in the classroom: https://support.balsamiq.com/desktop/classroom
>
> And a free resource that may be useful to you: a user-contributed collection of ready-to-use UI components and design patterns built using Balsamiq Mockups: https://mockupstogo.mybalsamiq.com/projects

Maybe your students will contribute there someday.

Happy wireframing from all of us to all of you.
Valerie

Valerie Liberty
Wow Division, https://balsamiq.com
Support: http://support.balsamiq.com
Sales: sales@balsamiq.com or +1 (415) 367-3531
We're good people, and we care.

Setup Grails

Grails is used as the framework for our application.

It provides Model View Controller by convention.

Follow the instructions here to get grails onto your machine.

You may want to use sdkman as recommended in the grails documentation. Here are the instructions to do that: http://sdkman.io/install.html

Installing Grails

Using IDEA for Grails

Grails installation through sdkman

Setup IntelliJ IDEA

IntelliJ IDEA is an Integrated Development Environment. We use it to create, run, debug and share our code.

To install IntelliJ IDEA follow this link: https://account.jetbrains.com/a/dl6pfg3l

You will need to register. I recommend you do not use your personal email address as you may want to later use IDEA and won't be able to use the email again.

We'll use Grails as our framework. If you haven't set that up do that now. Setup Grails

Once Grails is setup, you can learn how to work with it inside Intellij here: https://www.jetbrains.com/help/idea/getting-started-with-grails-3.html

Setup Lucidchart

Lucidchart is an online diagramming tool. It is useful for drawing diagrams for architecture, classes and other purposes.

We'll use the free version.

Go to this link: https://www.lucidchart.com/

And use the free button near the bottom.

As always, use your schilling email so you can later sign up with your email outside class.

Setup Status

Please fill out the table so I can see the current status of everyone and we can get us all to the same point.

For your operating system choose from Windows or Mac.

For Status choose from Not Started, Need Help and Complete

		Status	
Student	**Operating System**	**IDEA**	**Grails**
William	Windows 10	Complete	Complete
Alex	Windows 10	Complete	Complete
Daniel			
Deryk			
Peter			
Bryn			
Emmett	Mac OS X 10.11.6	Complete	Complete
Ketu	Mac OS X 10.12.6	Complete	Complete
Parker			
Sami	Windows	Complete	I think complete?
Sean			
Ryan			