CST8218 Assignment 1

This assignment is worth 15 marks, for 15% of your final grade.

Due: See the Brightspace for due date

Purpose

Implement a Web Application and RESTful HTTP API to do CRUD operations on sprites in the Sprite Game domain discussed in class. Using the code of the Sprite Case Study demonstrated in class as a starting point, implement a new application with a proper set of JSF pages to do CRUD operations on Sprites, and implement an HTTP RESTful interface as specified below.

Requirements

Architecture: You must maintain the Sprite Case Study tiered architecture, which is similar to the architecture of the Address Book example project studied in class (presentation layer with JSF pages and a named managed bean, a business layer with an Enterprise Session bean façade, and a data layer including an SQL database engine and database schema).

Packaging: The project must be organized in the same way as the Sprite Case Study, which is a Enterprise Application project with a Web (war) sub-project and an EJB sub-project. All EJBs in the project must reside in the EJB sub-project, so the Web sub-project will contain just JSF pages, and their associated Named Bean controller class.

Features:

1. Implement manipulation of Sprites through JSF, including changing color using a converter class that converts Color objects to strings and back. You can use any format for your strings. For example, HEX color codes are appropriate: `#RRGGBB`, as are decimal RGB codes: `[rrr,ggg,bbb]`
2. Implement appropriate data Validation on JSF manipulations (for example, does it makes sense for the x-position of a sprite be negative?)
3. Implement a Restful HTTP Interface capable of CRUD operations on Sprites:
    a. Each response from your API should include an appropriate HTTP response code: (501, 404, 201, 204, …)
    b. Supply an endpoint that can be used to determine how many sprites are in the database
    c. POST on a specific **id** should update the Sprite having that id with the new non-null information given by the Sprite in the body of the request
        i. It's an error if the **id** doesn't exist, or if the Sprite in the body has a non-matching **id**
        ii. Old values should be preserved if they are not overwritten by the new changes. Consider adding a method to the Sprite class so that the method

can be called on the old Sprite with a new Sprite as an argument. Perhaps `newSprite.updates(oldSprite)`, updates the old Sprite with all the non-null new Sprite values

d. PUT on a specific **id** should replace the Sprite having that id with the Sprite in the body of the request.
    i. It is an invalid request if the Sprite with that **id** doesn't exist, or if the Sprite in the body has a non-matching **id**
    ii. The old Sprite is replaced by the new Sprite, so only the new values should remain in the result that is stored in the database.
e. PUT on the root resource (sprite table) is not supported
f. POST on the root resource (sprite table)
    i. Accepts a Sprite in the body of the request
        1. Creates the new sprite if **id** is null
        2. Updates an existing sprite if **id** is not null and exists
            a. Be sure that new non-null attribute values overwrite old values, and old values that are not overwritten are preserved
        3. It is an invalid request if the **id** is not null and a Sprite with that id does not exist

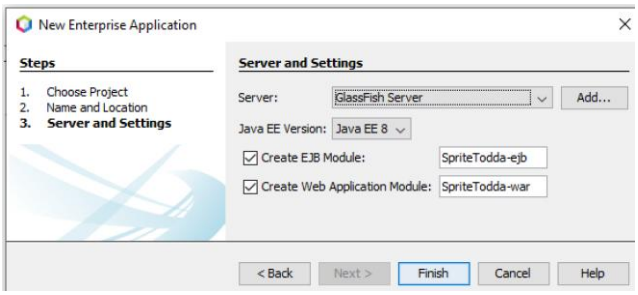
Recommended Steps

- (Preparation) Take inventory (including the annotations) of the Sprite Case Study code (you are not required to deal with the Java Swing Client, because the scope of this course is Web Applications, but you might find the Swing Client useful for testing your application)
    - Understand how the application works, and the role of each java class in the project (**find the annotations** in the six main Java files of the project)
    - Draw the architecture diagram of the version on a sheet of paper (the way we do in class on the whiteboard), with a box for the Glassfish server containing boxes for the EJB components and Web components. Write down the class-names of the application into the appropriate location on the diagram, and draw arrows to show the EJB dependencies. Now is a good time to change the `@EJB` annotation (`javax.ejb.EJB`) to `@Inject` (`javax.inject.Inject`), since it is the preferred annotation (they do the same thing in our case, but Inject is more general than EJB).
- (Tip) I recommend that you make commits to a source code repository often (or otherwise take a snapshot of your project folder), so that you can always revert back to a working version if necessary
- (Step One) Create a new JEE Application project with Netbeans:

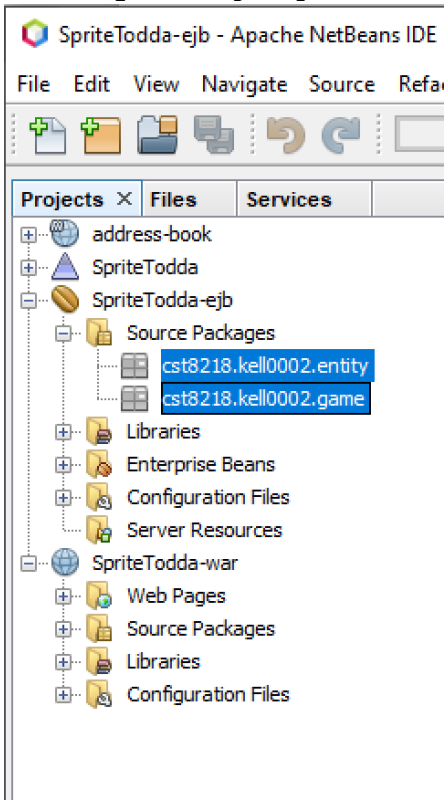
  Name the project `Sprite<Name>` where **`<Name>`** is your first name, for example if your first name is `Todda`, the project name would be `SpriteTodda`

  File->New Project…

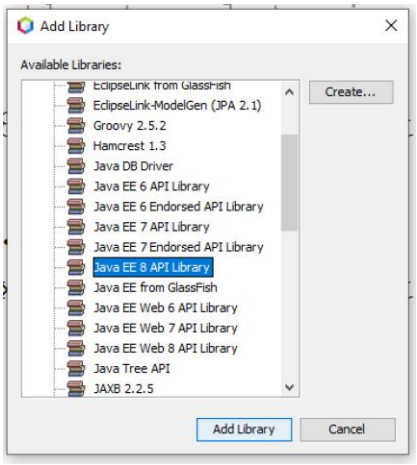
  Java with Ant->Java Enterprise->Enterprise Application



- In the `Sprite<yourname>-ejb` module project (the ejb project, or the ejb module), create the following packages under `Source Packages` (right-click->new->Java Package)
  `cst8218.<yourAlgonquinUserID>.game`
  `cst8218.<yourAlgonquinUserID>.entity`
  where `<yourAlgonquinUserID>` is your Algonquin user ID (example: `kell0002`):



- In the Sprite<yourname>-ejb project, right-click on Libraries, and add the Java 8 EE API library:



- Copy the Java Files from the Brightspace SpriteTodd Project into the appropriate package in your new project, refactoring the package names as necessary. Netbeans will not be able to do ALL of the package name refactoring, so expect to manually edit some package names in the refactoring process.
- Create a persistence.xml file in the `Sprite<yourname>-ejb` project by right-click->New->Other->Persistence->Persistence Unit , accepting the default persistence unit name and select the `DefaultDataSource`. Remember to find and edit the Java class that makes reference to the persistence unit name, and change it from `SpriteToddPU` to your unit name.
- In the `Sprite<yourname>-ejb` project, delete the `Configuration Files->ejb-jar.xml` file, because for our purposes the annotations in our code will be sufficient to configure our ejbs
- Build and run the project (if any of these steps fails, it's worth simply trying a second time, even before investigating what is wrong, but if it fails again, usually investigation will be necessary):
    - Clean and build the ejb
    - Clean and build the war (which depends on the ejb)
    - Clean and build the application (which depends on both)
    - Select the application and click the run (green triangle) button
- Add CRUD JSF pages to the war project, using the right-click "from Entity class" method.
    - If you have an entity java class file in your project, but there are no entity classes to choose from in the wizard, a workaround is to create an entity from database, after which you'll see both entities (pick the one from the ejb project)
    - fix any problems flagged by Netbeans, to get the whole Application running
        - Review Lab 2 as necessary, and ensure there is just one persistence unit, one entity facade, one entity class (if needed, remove the entity file you created while working around the "no entities" problem)
        - If you've double-checked your work on all of the above points, and you still get an error when you try to deploy/run your application, it may be caused by a Glassfish bug uncovered by previously deployed sprite-app versions. View the list of deployed applications and undeploy any previously deployed sprite-related applications. You can launch the

Glassfish Administration console in your browser (it's a webapp) with `Netbeans->Services Tab->Servers->Glassfish->View Domain Console`, and find `Applications` on the left side. Restart Netbeans after undeploying, to be sure the application no longer appears in the deployed list, and if the apps you wish to undeploy are STILL in the list after restarting Netbeans, then try rebooting windows. In the end after all of this if the app(s) won't undeploy, or if this doesn't resolve your issue, then seek help from your lab professor or professor.

    - remove the id field from being included on the JSF pages because the user should not need to worry about ids
- Color objects will not behave well on the JSF pages yet, but you can test creating sprites with the default color if you temporarily remove the color field from being included on the Create page. You will fix this problem and re-include color on the JSF pages later in this Assignment.
- Add a RESTful interface to the war project, and then move the created files to the ejb project (remember we are keeping all EJBs in the ejb project). After you're finished, there should be just one persistence.xml in the ejb project, and there should not be a persistence.xml in the war project.
    - test the interface with Postman
    - note that color objects will be broken in both JSON and XML, for now – we will work on implementing proper conversion through the restful API in a future assignment
    - for now, do not try to create or update the color property through the HTTP API
- Add Data Validation that would prevent the user from making the position of a Sprite negative (for example, prevent positions of x=-4, y=-100)
- Put the color field back in the appropriate JSF page(s), and add a converter to convert a Color object to a string for presentation to the user on the JSF page, and accept user changes to that string (giving the user a way to change color). Adding a color-picker is possible, but not required (the converter class is required).

Submission

Demonstrate your Sprite program to your Lab Instructor, and submit a zipped archive of the Netbeans project folder using the Brightspace link provided. Please use only zip, and do NOT use .7zip, or .rar formats for the submitted archive.

Grading Criteria

1. Implementation comments: (3 marks)
    - Class header comments on every class to explain the purpose of the class: provide comments for all classes, including those classes that were provided to you, and classes that are written by Netbeans. You may ignore `SpriteSession.java` and `SpriteSessionRemote.java`.
    - Method header comments on any methods you write or change (for your converter class, or any other purpose), and the RESTful interface methods. You do not need to provide comments for the methods that you didn't write or change, that do not handle RESTful requests.
2. JSF Pages: (4 marks) = **2 (submission) + 2 (demoing)**
3. RESTful API: (4 marks) = **2 (submission) + 2 (demoing)**
4. Mandatory Demonstration **+ Q&A:** (4 marks)