# CST8218 Assignment 1

This assignment is worth 20 marks, for 20% of your final grade.

Due: See the Brightspace for due date

## Purpose

Implement a Sprite Game Web Application and RESTful HTTP API to do CRUD operations on sprites. Following the process 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, satisfying all the requirements specified below.

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

Packaging: All EJBs in the project must reside in a package (or packages) dedicated to business logic. Classes that are used in the implementation of the JSF presentation layer must reside in a package (or packages) dedicated to the presentation layer. All packages in the project should be appropriately named, as discussed in lecture.

### Features:

1. The Sprite class should include fields sufficient to represent position on an X-Y grid of pixels (such as a canvas or JPanel), as well has how much the position of the sprite changes when its change of position is animated frame by frame. Accordingly, Sprites must have the ability to move.
2. Implement manipulation of Sprites through JavaServer Faces pages.
3. Implement appropriate data Validation on JSF manipulations of position, in that positions cannot be negative. Other limits, such as maximum or minimum speeds, or the maximum position values are not required at this stage.
4. Implement a Restful HTTP Interface capable of CRUD operations on Sprites:
   1. Each response from your API should include an appropriate HTTP response code:

https://docs.oracle.com/javaee/7/api/javax/ws/rs/core/Response.html

return Response.status(Response.Status.CREATED).entity(newEntity).build();

* + - 200
    - 201
    - 204
    - 400
    - 404
    - 405
    - 500
  1. Supply an endpoint that can be used to determine how many sprites are in the database
  2. 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
     + It’s an error if the **id** doesn’t exist, or if the Sprite in the body of the request has a non-matching **id**
     + 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
  3. PUT on a specific **id** should replace the Sprite having that id with the Sprite in the body of the request.
     + 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**
     + 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.
  4. PUT on the root resource (sprite table) is not supported
  5. POST on the root resource (sprite table)
     + Accepts a Sprite in the body of the request
       1. Creates the new sprite if **id** is null, and returns the created sprite
       2. Updates an existing sprite if **id** is not null and exists
          1. 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

* Draw the architecture diagram, like those discussed in class, on a sheet of paper, or using your favorite draw program. As with the diagrams discussed in class, use a box for the Glassfish server containing boxes for the EJB components and Web components. Write down the class-names of the classes needed for the application into the appropriate location on the diagram, and draw arrows to show the interconnections between the components.
* (Tip) I recommend that you make commits to a source code repository, or otherwise take a snapshot of your project folder, so that you can always revert back to a working version if necessary
* Create a new Maven-based Web Application project with Netbeans named Sprite<Name> where **<Name>** is your first name, for example if your first name is Todd, the project name would be SpriteTodd
* For packages, use the following package names:

cst8218.<yourAlgonquinUserID>.game

cst8218.<yourAlgonquinUserID>.entity

where <yourAlgonquinUserID> is your Algonquin user ID (example: kell0002):

* Add CRUD JSF pages to the project, using the right-click “from Entity class” method.
  + remove the id field from being included on the JSF pages because the user should not manage the primary key row ids
* Add a RESTful interface to the project
  + test the interface with Postman.
* 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)
* Go through the Features List above, and do the programming necessary to support each feature (some features may require little or no additional programming on your part)
* You are the programmer responsible for all the code in your project. Write class header comments on every class to explain the purpose of the class: provide comments for all classes, including those classes that you wrote and classes that were created by Netbeans.
* Write method header comments above any methods you write or change, including the RESTful interface methods. You need to provide comments for all methods that handle REST API requests, but other than those you do not need to provide comments for the methods that you didn’t write or change.

## Submission

Demonstrate your Sprite program to your Lab Instructor, and submit a zipped archive of

* your architecture diagram, as a photo of a hand-drawn image, or a pdf image
* 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

* architecture diagram (2 marks)
* Implementation comments: (3 marks)
* Your Implementation comments need to be in your own words, and they will be graded on how well they show your understanding of the purpose the classes and methods in the overall application, and how well they show your understanding of how the overall application works.
* Class header comments on every class to explain the purpose of the class.
* Method header comments on any methods you write or change, and for all of the RESTful interface methods.
* JSF Pages: (2 marks)
* RESTful API: (8 marks)
* Mandatory Demonstration **+ and questions during demonstration:**  (5 marks)
* If you do not demonstrate, there is a penalty (lose 15 marks):
  1. No marks for demonstration (loss of 5 marks), plus
  2. Penalty of -10