The Technology Stack

HTTP

#1

GET "/hello"

POST "/game/"

Browser

Hello testing Method
Hello Controller
Game: Skyrim
Genre: RPG

.jsp / .html **#2**
```
<script src="..."></script>
<body ng-app="GameApp"> ${ name }
  <div ng-controller="GameController as ctrl" >
    <h1>{{ctrl.controllerMessage}}</h1>
    <span>Game: {{ctrl.game.name}}</span><br />
    <span>Genre: {{ctrl.game.genre}}</span>
  </div>
</body>
```

#7

#3 ANGULARJS by Google
*.module.js
```
angular
  .module('app', []);
```

*.controller.js **#4**
```
angular
  .module('GameApp')
  .controller('GameController', GameController);

GameController.$inject = [GameService, '$log'];

function GameController(GameService, $log) {
  var vm = this;
  vm.controllerMessage = "Hello Controller";
  vm.game = { id: ", name: " ", genre: " "};

  vm.getServiceMessage = function (){
    GameController.addGame(game)
    .then( function(data){
      vm.game = data.message;
    });
  }
}}
```

*.service.js
```
angular
  .module('GameApp')
  .service('GameService', GameService);

GameService.$inject = ['$log', '$http'];

function GameService($log, $http) {
  var vm = this;

  vm.addGame = function(game){
    return $http.post('game', game).then(
      function(response){
        return response.data;
      }
    );
  }
}
```

game (JSON) HttpStatus.OK (200)
```
{ response: {
  data : {
    id : "1",
    name : "Skyrim",
    genre : "RPG"
  }
}
}
```
#6

*.Controller.java (MVC Java)
```
@Controller
public class HomeController {

  @RequestMapping(value="/hello")
  public ModelAndView hello(@RequestParam(required=false, defaultValue="World") String name) {
    ModelAndView modelAndView = new ModelAndView("games");
    // Adds Java object to be used in home.jsp
    modelAndView.addObject("name", testingMethod());
    return modelAndView;
  }

  private String testingMethod() {
    return "testing Method"; //bind whatever java objects needed to the view
  }
}
```

*.Controller.java (REST) **#5**
```
@RequestMapping(value ="/game")
@RestController
public class GameController {

  @Autowired
  private GameService gameService;

  @PostMapping(value = "/", consumes = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<> createGame(@RequestBody GameImpl game) {
    Game savedGame = gameService.saveGame(game);
    return new ResponseEntity<>(savedGame, HttpStatus.OK);
  }
// ...
}
```

*Manager.java / *Service.java
```
@Service
public class GameServiceImpl implements GameService {

  @Autowired
  private GameDao gameDAO;

  public Game saveGame(Game game) {
    return gameDAO.save(game);
  }
//...
}
```

*DAO.java / *Repo.java
```
@Repository
public class GameDAO{

  public Game save(Game game) {
    //Hibernate, SQL, or other implemtation omitted
    return game
  }
}
```

Data

Server

Game.java (I)
```
public interface Game {
  Long getId();
  void setId(Long id);
  String getName();
//...
}
```

GameImpl.java (C)
```
@Component
public class GameImpl implements Game {

  private Long id;
  private String name;
  private String genre;

  @Override
  public Long getId() {
    return id;
  }
  @Override
  public void setId(Long id) {
    this.id = id;
  }
  // getters setters omitted
}
```
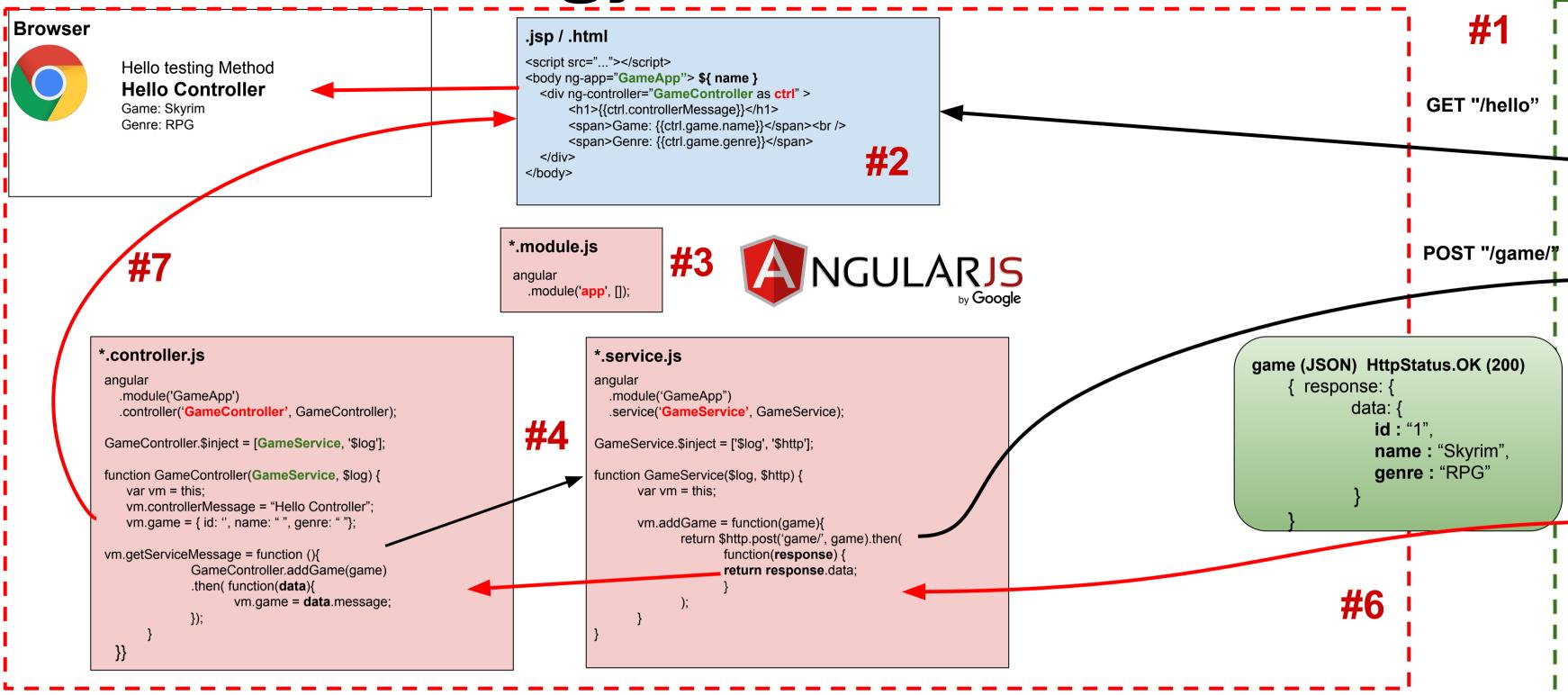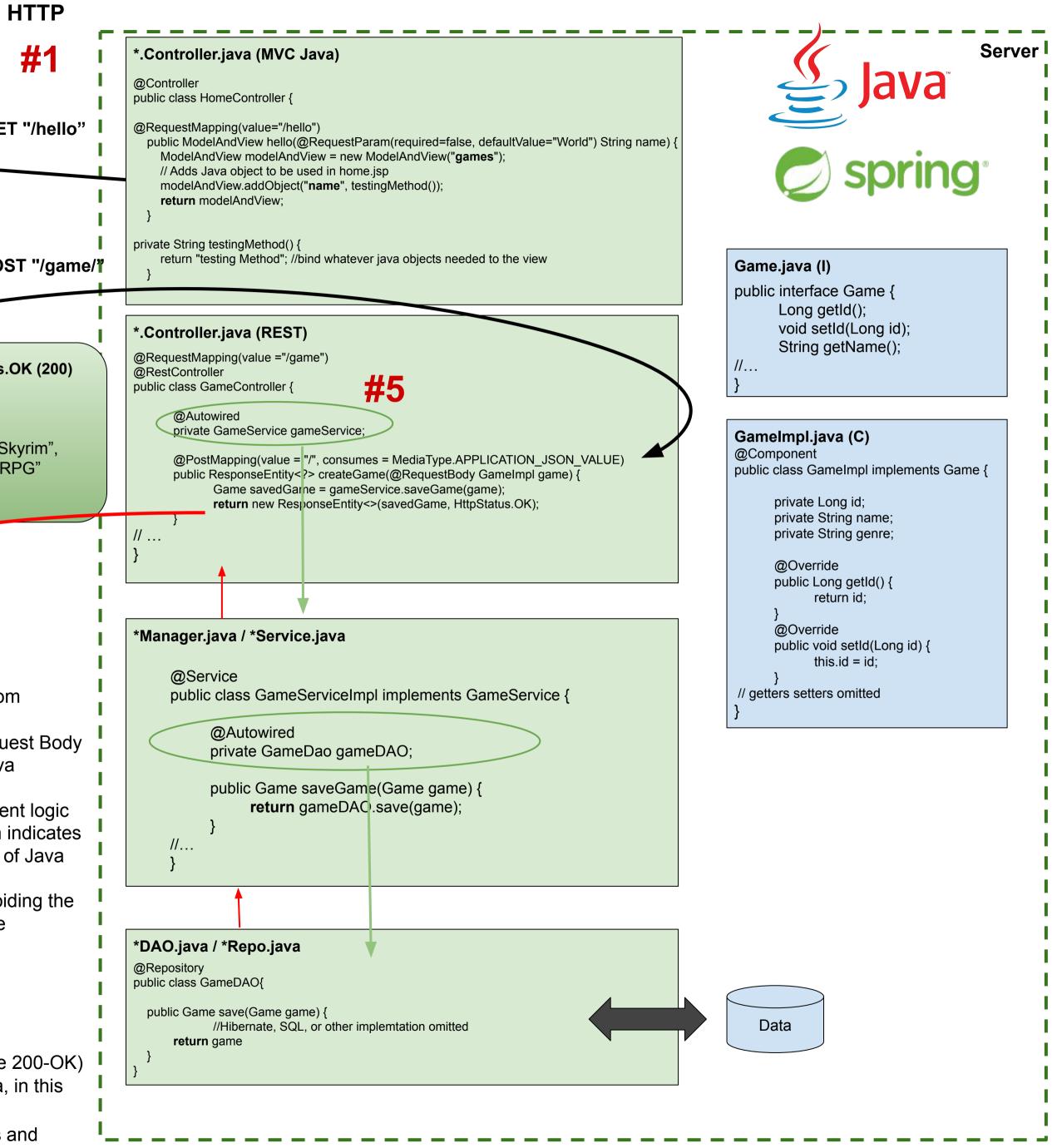
1. Browser initiates a "GET" URL request to a Java Controller returning a "ModelAndView" (Which are Java objects added to view, in this case "home" aka home.jsp Think of JSP as HTML /w Java, which serves as the foundation/starting point for our UI. Keep in mind, entire synchronous applications can be built using nothing but Java and JSP.

2. However we *like* Asynchronous UI, that doesn't reload the entire page every time we make a change. To do this, our JSP, contains a simple AngularJS app, that we use to demonstrate simplified RESTful asynchronous interaction with Java data via a very basic REST API..

3. Using <script> tags in the JSP, we load (in-order) the:
   a. AngularJS library,
   b. our module.js (declares the app),
   c. service.js (a dependency to the controller, which RESTfully talks to our Java), (See #4)
   d. and finally controller.js (which initializes our view of the page)

4. When the controller.js initializes, it uses service.js, to call the Java "REST" Controller to initialize and interact with our application data.

5. A URL "POST" request "/game/" with a Game JSON object is sent from service.js.
   a. Java library called Jackson, attempts to translate our JSON Request Body to a Game object defined in Java. The object then enters our Java Controller.
   b. We then "go down the stack" in Java, stepping through our different logic layers and back again (in order). Note the @Autowire annotation indicates that we are using Spring Framework to manage the instantiation of Java layers (@Service, @Repository, examples of specified "Beans") in-memory, so we offload manual memory management with avoiding the "new" keyword in Java. Each represents a layer of concern in the application.
      i.   Controller
      ii. Manager/Service
      iii. DAO -> Database

6. Once data has returned to the Controller.java A Server Response (like 200-OK) with a body containing our JSON representation of a Java object data, in this case an object called "Game

7. The JSON data is then bound to Javascript models in the controller.js and displayed by angularjs {{ }} reference to those in the HTML tags.