PROJECT Design Documentation

Team Information

- Team name: Lightning
- Team members
    - Ben Sippel
    - Andrew Apollo
    - Jessica Ahls
    - Aidan Ruiz

Executive Summary

An E-Store for the NHL Hockey Team Tampa Bay Lightning.

Purpose

> Customers can browse jerseys, add jerseys to cart, and make purchases. Admin are allowed to edit the inventory, adding or removing jerseys as they see fit.

Glossary and Acronyms

> *A table of terms and acronyms.*

Term	Definition
SPA	Single Page

Requirements

This section describes the features of the application.

Definition of MVP

> *The Minimum Viable Product involves an E-Store with basic front and back-end capabilities. The admin of the site should be able to create new jerseys to be shown on the site, being able to determine the name, price, size, and color for the jersey, and determine the quantity of the jersey in the inventory. The admin can also edit the jersey's information to change one or more attributes. The admin can also remove jerseys so that they are no longer displayed on the site. As a customer, you are able to log into your personal account and have a cart that persists through logging in/out. You can browse the in-stock jerseys and add to your cart. In your cart, you can increase the quantities of the jerseys in your cart and checkout, which clears your cart.*

MVP Features

> *API Functionality*

```
Create new product

-   Get a single product

-   Get entire inventory

-   Delete a single product

-   Update product

-   Search Product

Admin Functionality

-   Edit Jersey Information

-   Add to Inventory

-   Remove from Inventory

-   Admin Login

Customer Functionality

-   Browse Products

-   View a Product

-   Add to Cart

-   Remove from Cart

-   Change Quantity

-   Cart Checkout

-   Customer Login
```

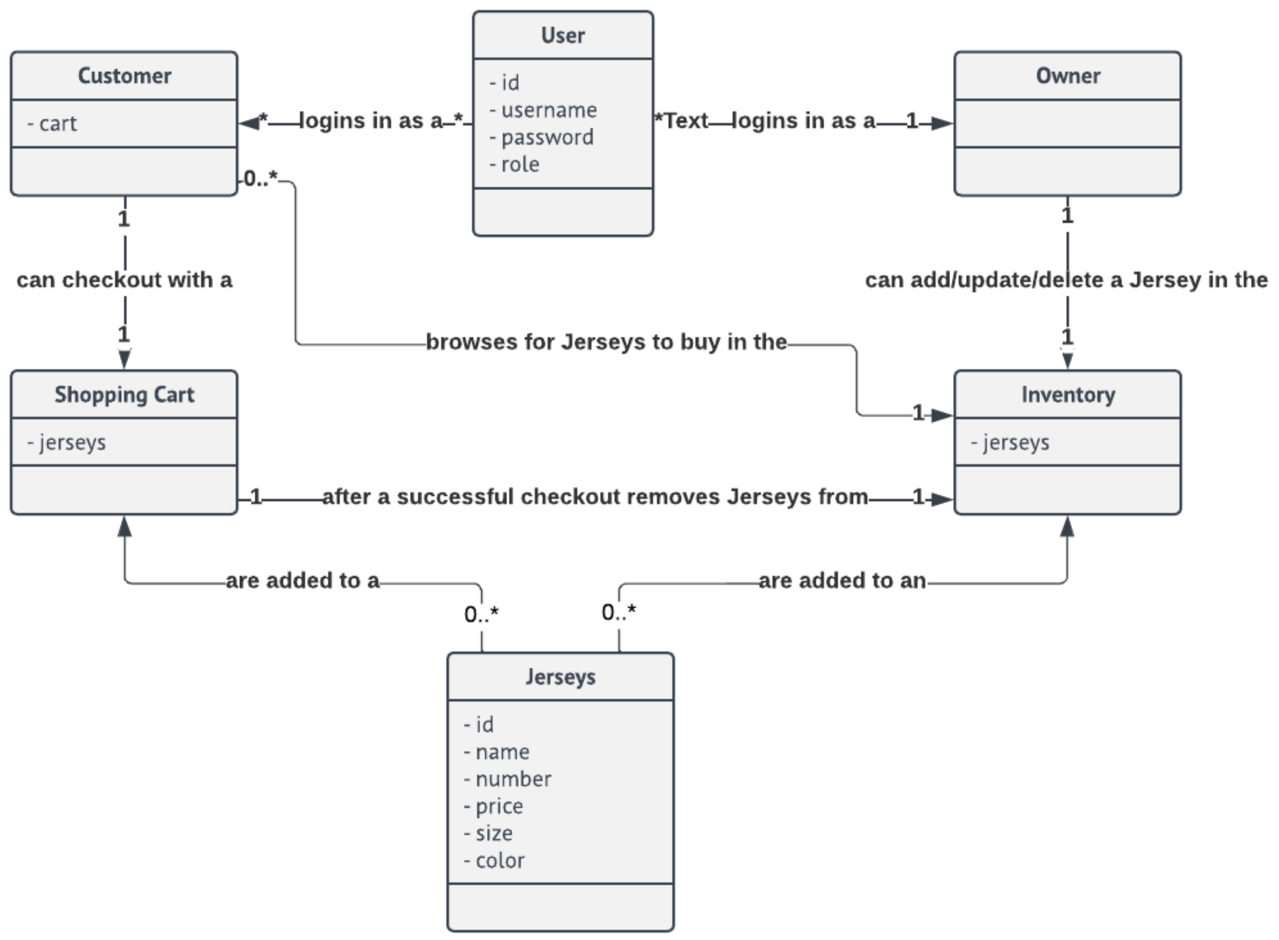
Roadmap of Enhancements

> *Future enhancements:*

When the admin removes a jersey that a customer has in their cart, this change should be reflected in the customer's cart, furthermore they should not be able to check out a jersey that has been removed. During the checkout process, the customer should be required to enter their shipping and billing information and this information should be validated, whether it be for formatting or if the address and payment information themselves are valid. Our 10% feature involves a finite stock that is updated when a customer checks out jerseys from the site. A customer should not be able to add more jerseys than what is in the inventory to their cart, therefore when the customer checks out, the in-stock quantity will decrease successfully. Customers will be able to view what jerseys are low in stock and see what jerseys are unavailable, or having a quantity of 0 available.

Application Domain

> *This section describes the application domain.*



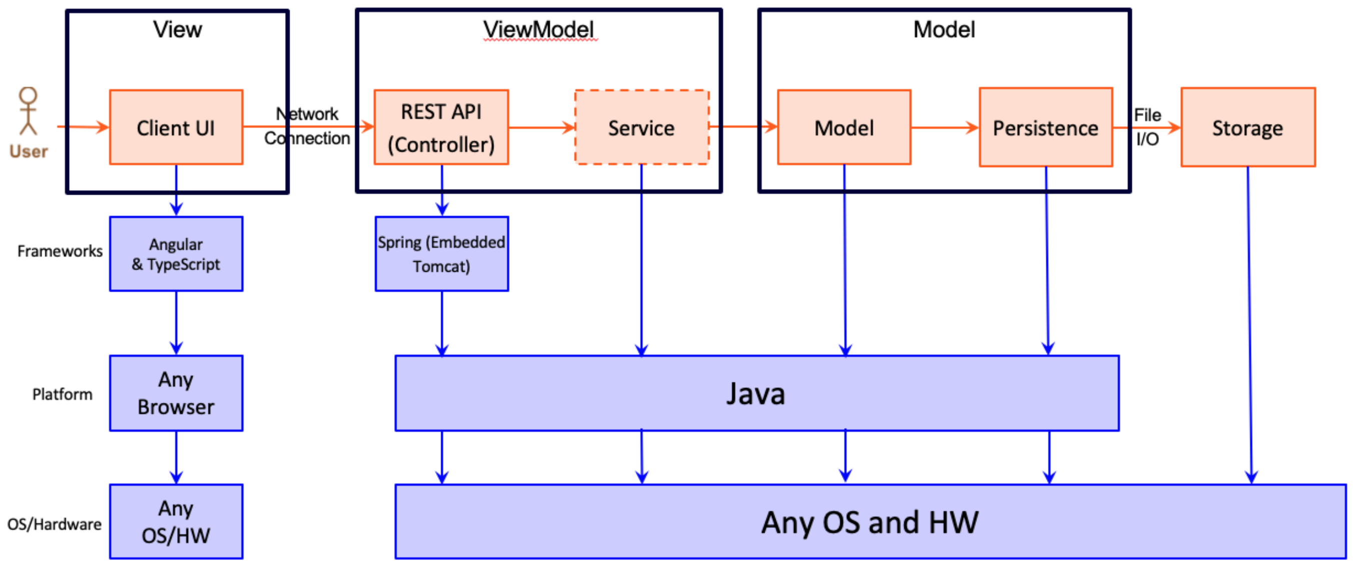
> This Estore will begin with a User entering the site and being able to either explore the site or enter their credentials. If the user is a customer, they will have access to their personal cart and be able to browse and select jerseys they wish to add to cart. If the user is an admin, they will be able to add, remove, or edit jersey data.

Architecture and Design

This section describes the application architecture.

Summary

The following Tiers/Layers model shows a high-level view of the webapp's architecture.



The e-store web application, is built using the Model–View–ViewModel (MVVM) architecture pattern.

The Model stores the application data objects including any functionality to provide persistance.

The View is the client-side SPA built with Angular utilizing HTML, CSS and TypeScript. The ViewModel provides RESTful APIs to the client (View) as well as any logic required to manipulate the data objects from the Model.

Both the ViewModel and Model are built using Java and Spring Framework. Details of the components within these tiers are supplied below.

Overview of User Interface

This section describes the web interface flow; this is how the user views and interacts with the e-store application.

Customers

> A user visits the e-store and is greeted with a view of all the jerseys available. They can search for a Jersey by name and see the Browse page update simultaneously to show only matching Jerseys. Clicking on a Jersey allows them to view its details. In order to purchase Jerseys they visit the login page and enter valid login credentials. After logging in they now see a cart they can navigate to at the top of the page. When viewing a Jersey's details there is now an option to add to cart. Adding a Jersey to cart will immediately bring them to their newly updated cart. Users can change the quantity of a cart item by 1 or by entering in a custom value. If they decide they no longer want to purchase a Jersey they can remove it from cart. Users can view their cart's total on the right of the cart page and it will be updated whenever they make changes. When they want to checkout users click the checkout button and their cart will be cleared.

Owner

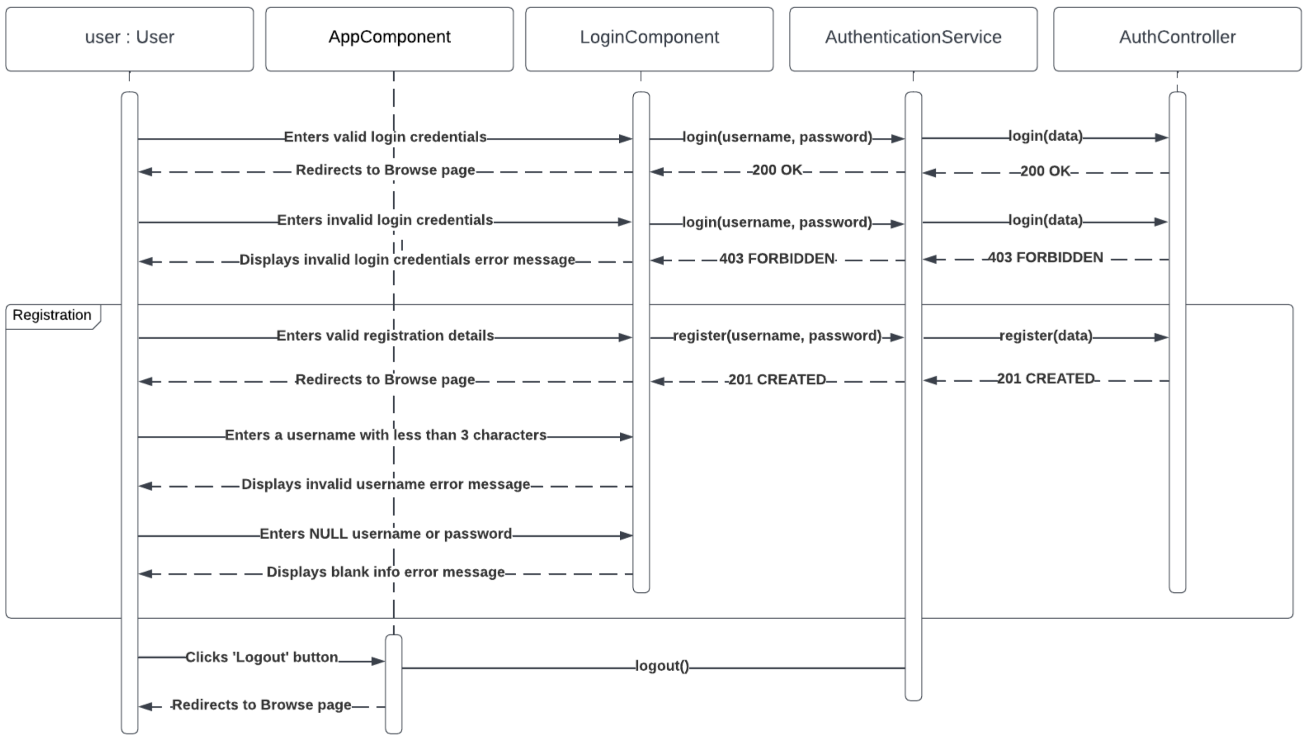
> Once logging in with valid credentials owners have all the functionality of a user that is logged in, except they do not have a cart and cannot purchase Jerseys, instead there is an inventory page where they can manage their stock. Once on the inventory page, admins see a list of Jerseys and their names sorted by ID. They can remove a Jersey from inventory by clicking the 'X' next to its name, and if they

> give confirmation it is deleted. If they press the 'Add new Jersey' button they are taken to a new page that allows them to create a new Jersey in inventory. There is a 'Go Back' button that takes them back to inventory. Admins can modify a Jersey's any of a Jersey's attributes by clicking on its name on the list. Admins cannot add or update a Jersey so that it has completely identical attributes as another, and if they attempt to do so, they are notified of the failure to create/update.

View Tier

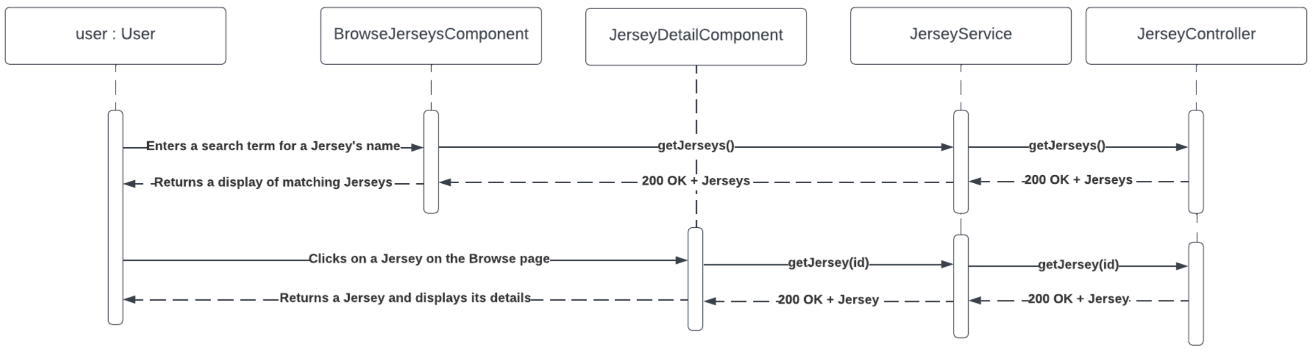
> *The following sequence diagrams show how the View and ViewModel interact, and include relevant HTTP requests from the client-side to the server-side to help illustrate the end-to-end flow.*

LoginComponent


user : User
AppComponent
LoginComponent
AuthenticationService
AuthController
Enters valid login credentials
login(username, password)
login(data)
Redirects to Browse page
200 OK
200 OK
Enters invalid login credentials
login(username, password)
login(data)
Displays invalid login credentials error message
403 FORBIDDEN
403 FORBIDDEN
Registration
Enters valid registration details
register(username, password)
register(data)
Redirects to Browse page
201 CREATED
201 CREATED
Enters a username with less than 3 characters
Displays invalid username error message
Enters NULL username or password
Displays blank info error message
Clicks 'Logout' button
logout()
Redirects to Browse page

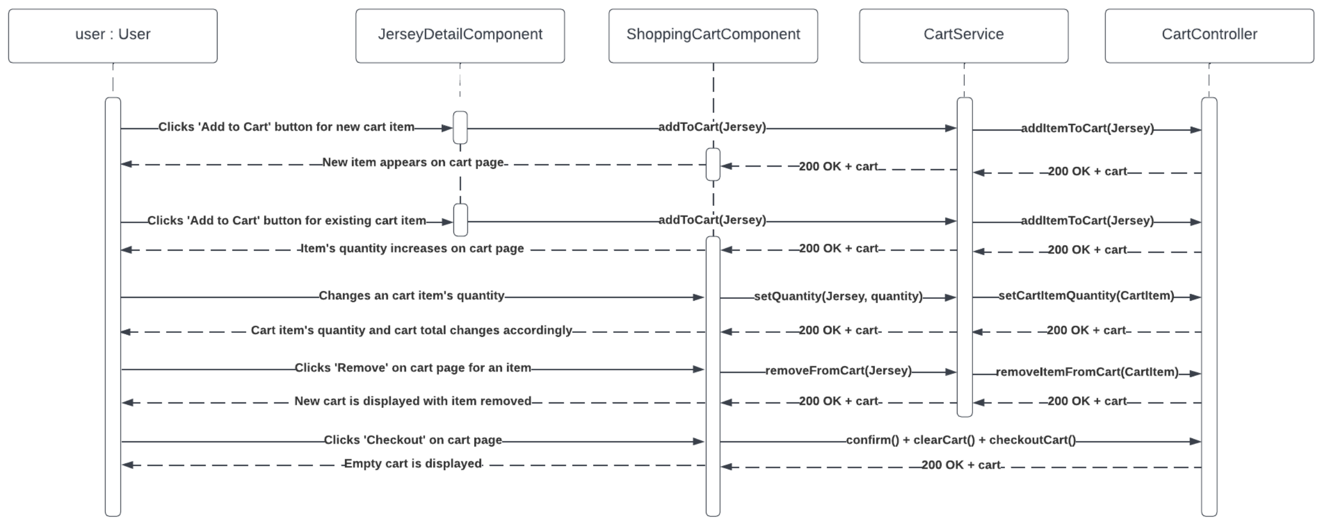
> *A user logging on and off the E-store.*

BrowseJerseysComponent and JerseyDetailComponent



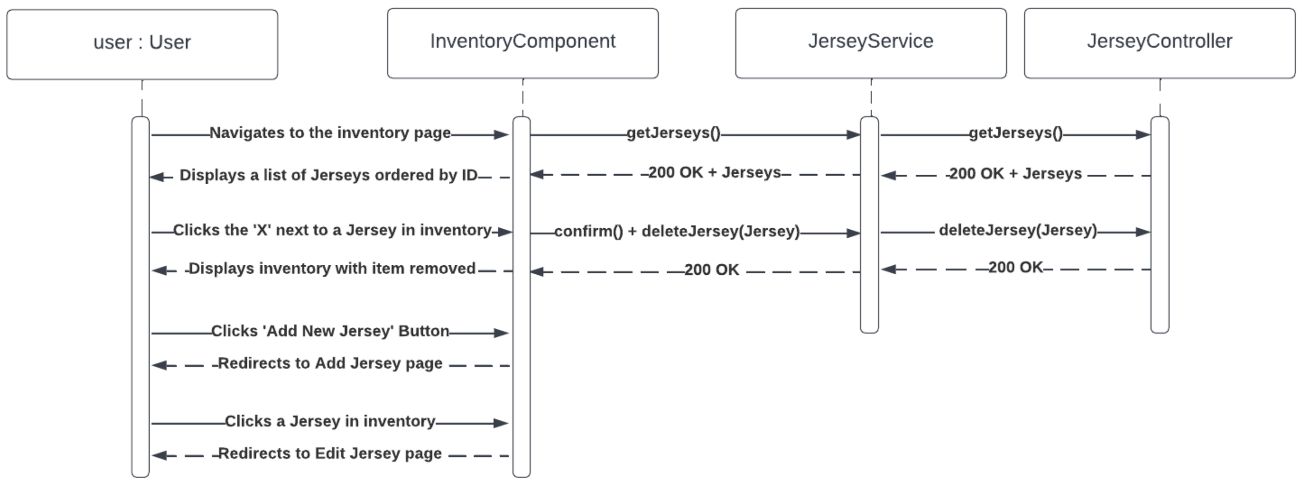
> *A customer browsing, searching, and reading the details of Jerseys in the E-store.*

ShoppingCartComponent


user : User
JerseyDetailComponent
ShoppingCartComponent
CartService
CartController
Clicks 'Add to Cart' button for new cart item
addToCart(Jersey)
addItemToCart(Jersey)
New item appears on cart page
200 OK + cart
200 OK + cart
Clicks 'Add to Cart' button for existing cart item
addToCart(Jersey)
addItemToCart(Jersey)
Item's quantity increases on cart page
200 OK + cart
200 OK + cart
Changes an cart item's quantity
setQuantity(Jersey, quantity)
setCartItemQuantity(CartItem)
Cart item's quantity and cart total changes accordingly
200 OK + cart
200 OK + cart
Clicks 'Remove' on cart page for an item
removeFromCart(Jersey)
removeItemFromCart(CartItem)
New cart is displayed with item removed
200 OK + cart
200 OK + cart
Clicks 'Checkout' on cart page
confirm() + clearCart() + checkoutCart()
Empty cart is displayed
200 OK + cart

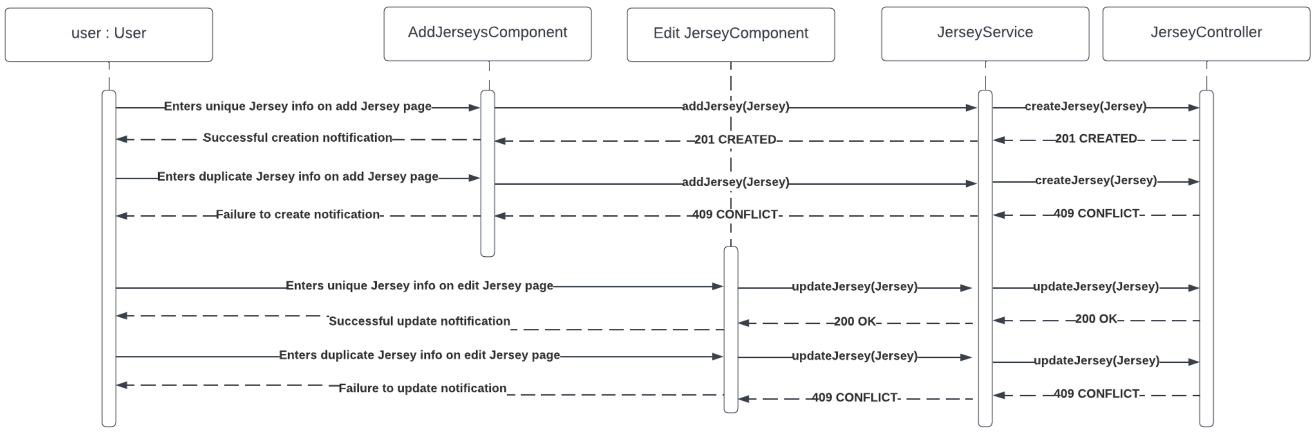
> *A customer managing their cart by adding, removing, and updating the quantity of cart items.*

InventoryComponent


user : User
InventoryComponent
JerseyService
JerseyController
Navigates to the inventory page
getJerseys()
getJerseys()
Displays a list of Jerseys ordered by ID
200 OK + Jerseys
200 OK + Jerseys
Clicks the 'X' next to a Jersey in inventory
confirm() + deleteJersey(Jersey)
deleteJersey(Jersey)
Displays inventory with item removed
200 OK
200 OK
Clicks 'Add New Jersey' Button
Redirects to Add Jersey page
Clicks a Jersey in inventory
Redirects to Edit Jersey page

> *An admin viewing and removing items from inventory.*

AddJerseyComponent, and EditJerseyComponent

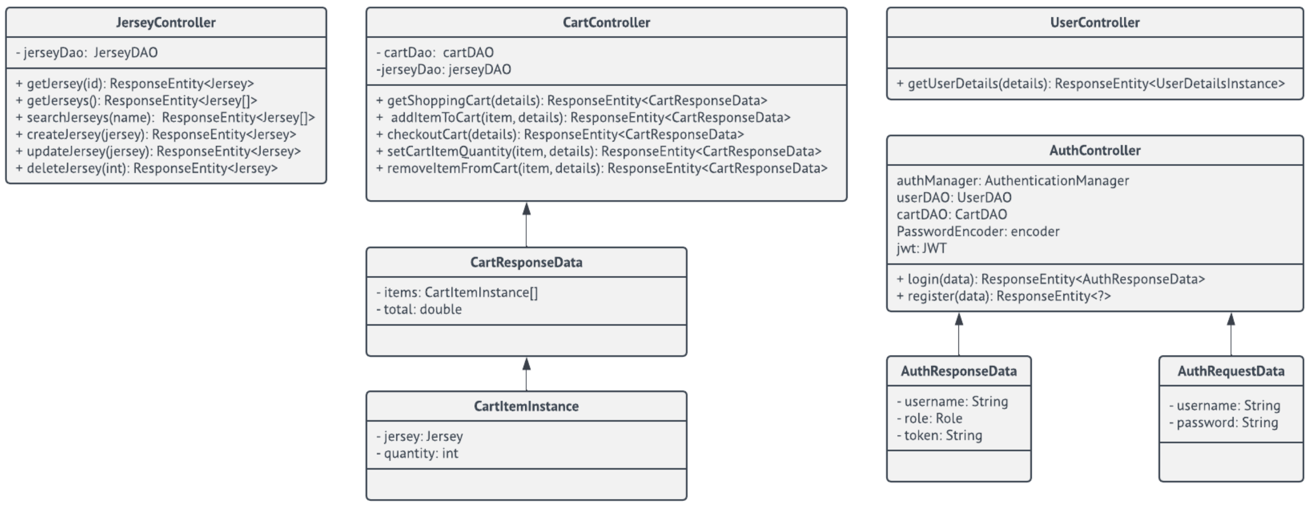


> *An admin managing inventory by adding and updating items in the inventory. Includes the cases when an admin attempts to create a duplicate Jersey.*

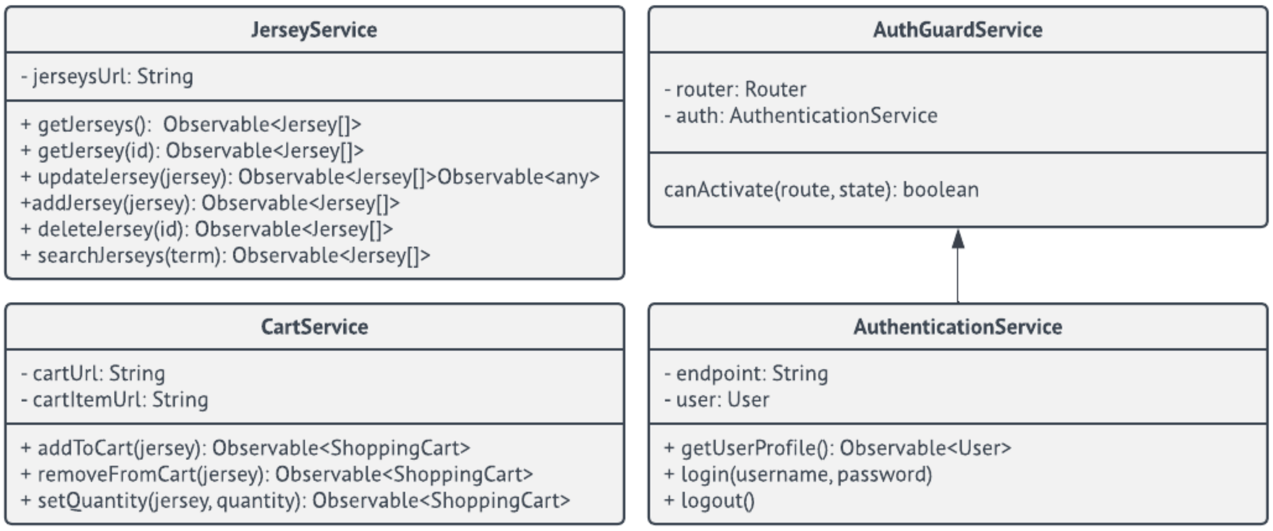
ViewModel Tier

> *An overview of the ViewModel tier of the application's architecture. Includes UML diagrams of all controllers and services.*

Controllers


JerseyController
- jerseyDao: JerseyDAO
+ getJersey(id): ResponseEntity<Jersey>
+ getJerseys(): ResponseEntity<Jersey[]>
+ searchJerseys(name): ResponseEntity<Jersey[]>
+ createJersey(jersey): ResponseEntity<Jersey>
+ updateJersey(jersey): ResponseEntity<Jersey>
+ deleteJersey(int): ResponseEntity<Jersey>
CartController
- cartDao: cartDAO
-jerseyDao: jerseyDAO
+ getShoppingCart(details): ResponseEntity<CartResponseData>
+ addItemToCart(item, details): ResponseEntity<CartResponseData>
+ checkoutCart(details): ResponseEntity<CartResponseData>
+ setCartItemQuantity(item, details): ResponseEntity<CartResponseData>
+ removeItemFromCart(item, details): ResponseEntity<CartResponseData>
CartResponseData
- items: CartItemInstance[]
- total: double
CartItemInstance
- jersey: Jersey
- quantity: int
UserController
+ getUserDetails(details): ResponseEntity<UserDetailsInstance>
AuthController
authManager: AuthenticationManager
userDAO: UserDAO
cartDAO: CartDAO
PasswordEncoder: encoder
jwt: JWT
+ login(data): ResponseEntity<AuthResponseData>
+ register(data): ResponseEntity<?>
AuthResponseData
- username: String
- role: Role
- token: String
AuthRequestData
- username: String
- password: String

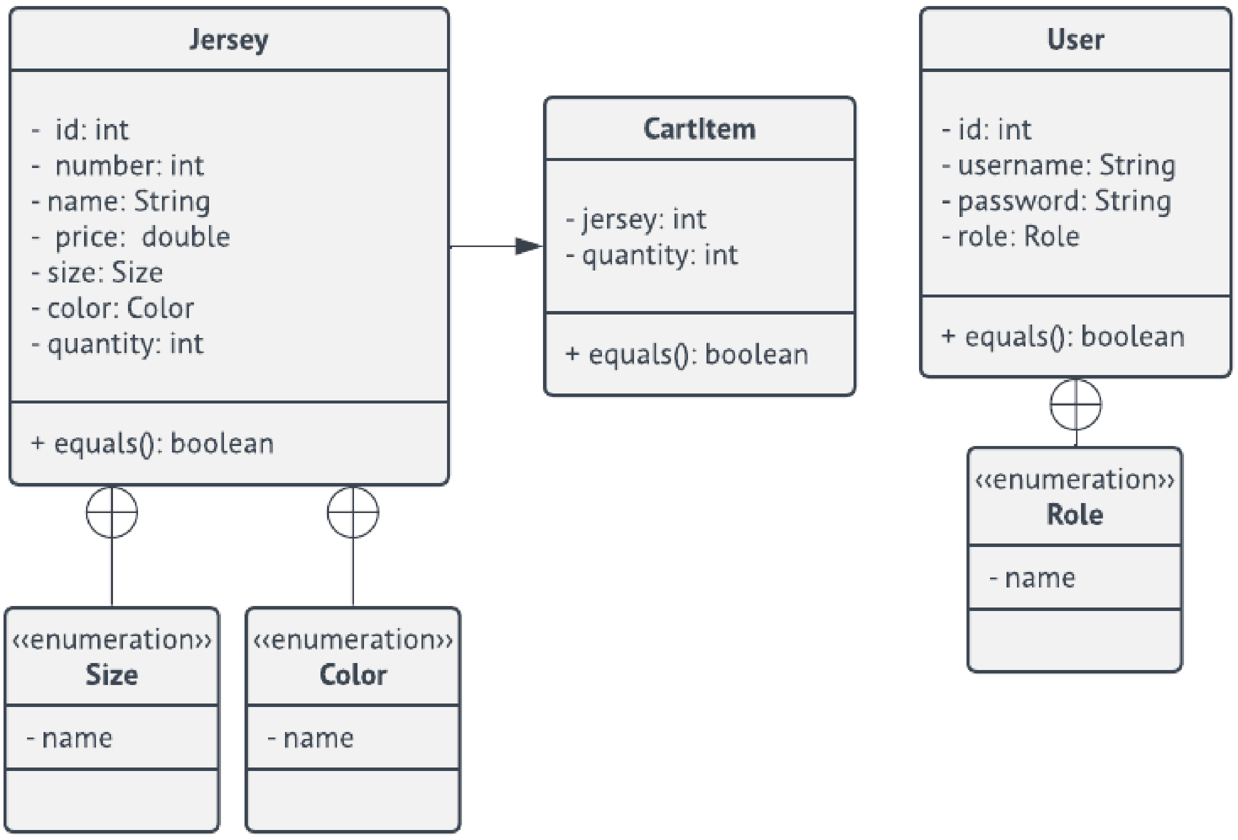
Services


JerseyService
- jerseysUrl: String
+ getJerseys(): Observable<Jersey[]>
+ getJersey(id): Observable<Jersey[]>
+ updateJersey(jersey): Observable<Jersey[]>Observable<any>
+addJersey(jersey): Observable<Jersey[]>
+ deleteJersey(id): Observable<Jersey[]>
+ searchJerseys(term): Observable<Jersey[]>
AuthGuardService
- router: Router
- auth: AuthenticationService
canActivate(route, state): boolean
CartService
- cartUrl: String
- cartItemUrl: String
+ addToCart(jersey): Observable<ShoppingCart>
+ removeFromCart(jersey): Observable<ShoppingCart>
+ setQuantity(jersey, quantity): Observable<ShoppingCart>
AuthenticationService
- endpoint: String
- user: User
+ getUserProfile(): Observable<User>
+ login(username, password)
+ logout()

Model Tier

> *An overview of the Model tier of the application's architecture. Includes UML diagrams of Model and Persistence sections.*

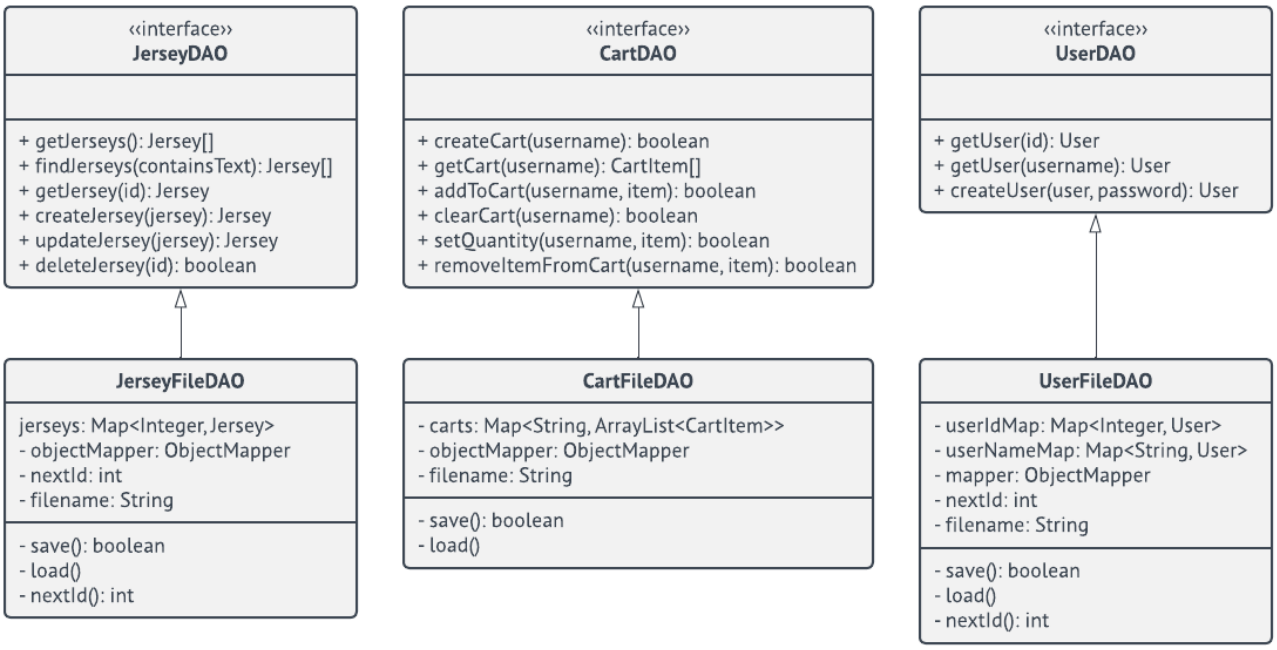
Model



> - *Jerseys's equals() returns true if all fields are equivalent excluding id.*
> - *CartItems's equals() returns true if the jerseys are equivalent.*

> - *User's equals() returns true if the usernames are equivalent.*

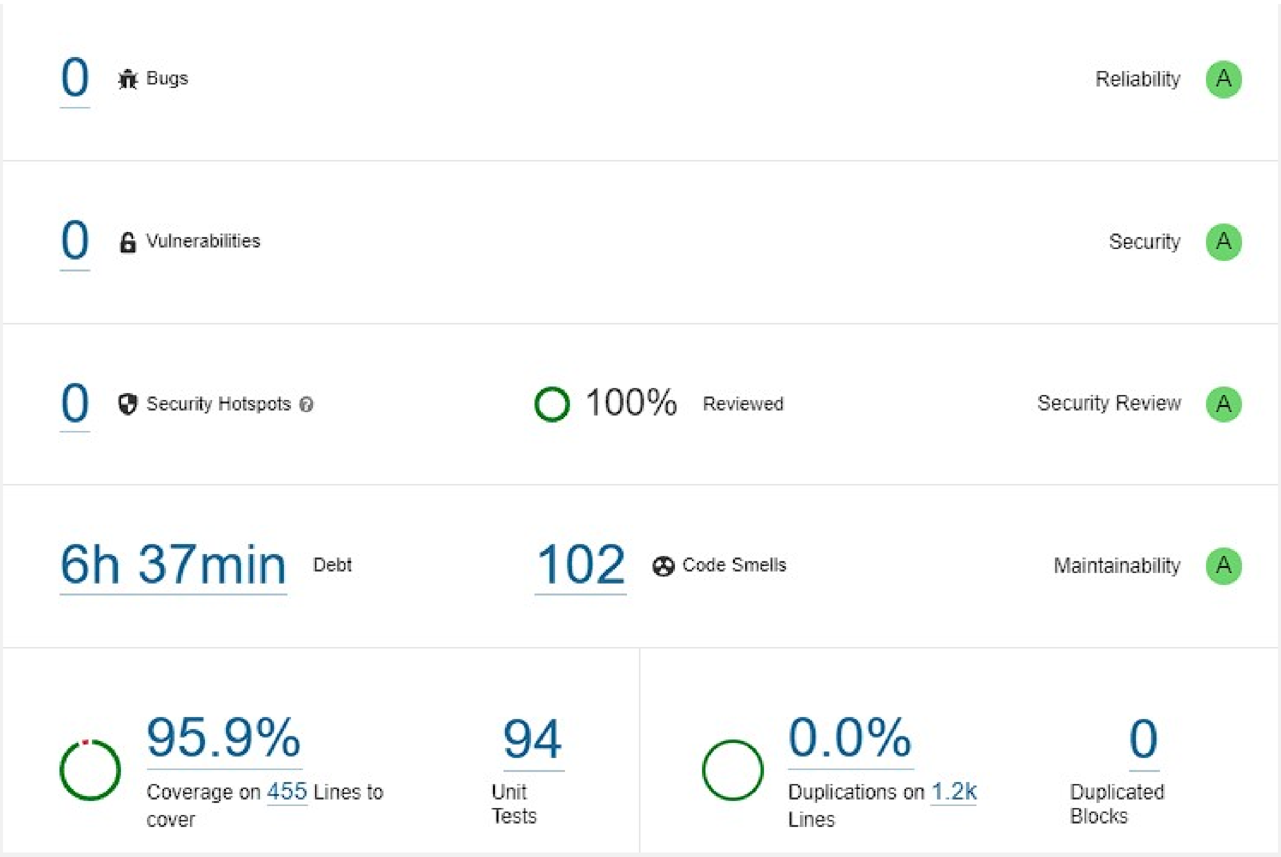
Persistence


«interface»
JerseyDAO
+ getJerseys(): Jersey[]
+ findJerseys(containsText): Jersey[]
+ getJersey(id): Jersey
+ createJersey(jersey): Jersey
+ updateJersey(jersey): Jersey
+ deleteJersey(id): boolean
«interface»
CartDAO
+ createCart(username): boolean
+ getCart(username): CartItem[]
+ addToCart(username, item): boolean
+ clearCart(username): boolean
+ setQuantity(username, item): boolean
+ removeItemFromCart(username, item): boolean
«interface»
UserDAO
+ getUser(id): User
+ getUser(username): User
+ createUser(user, password): User
JerseyFileDAO
jerseys: Map<Integer, Jersey>
- objectMapper: ObjectMapper
- nextId: int
- filename: String
- save(): boolean
- load()
- nextId(): int
CartFileDAO
- carts: Map<String, ArrayList<CartItem>>
- objectMapper: ObjectMapper
- filename: String
- save(): boolean
- load()
UserFileDAO
- userIdMap: Map<Integer, User>
- userNameMap: Map<String, User>
- mapper: ObjectMapper
- nextId: int
- filename: String
- save(): boolean
- load()
- nextId(): int

> *Each FileDAO keeps a local cache of jerseys, shopping carts, and users respectively, initialized from three json files which are updated whenever a change is made.*

Static Code Analysis

> SonarQube static code analysis reports for our backend show that we've attained high grades for our code coverage and quality. There is however some code smell and tech debt, but it seems that these mostly involve the accessibility of certain methods rather than any critical flaws, for example, a lot of accumulated debt comes from unit tests having public modifiers where they aren't needed.



0 Bugs — Reliability A

0 Vulnerabilities — Security A

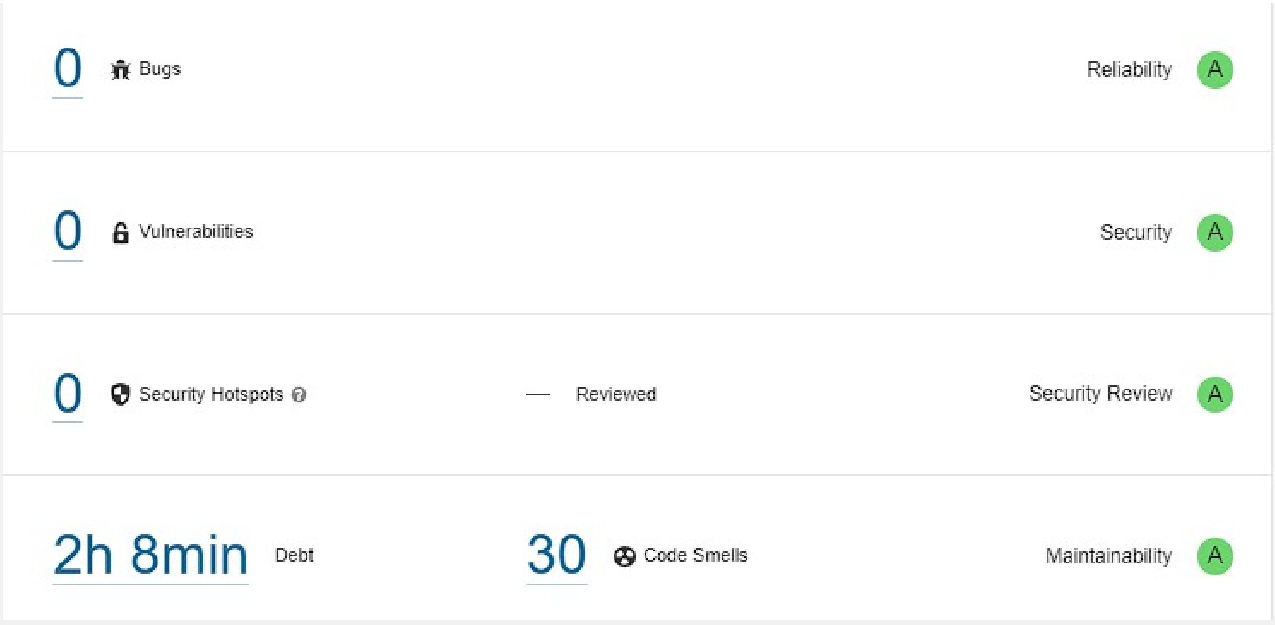
0 Security Hotspots — 100% Reviewed — Security Review A

6h 37min Debt — 102 Code Smells — Maintainability A

95.9% Coverage on 455 Lines to cover — 94 Unit Tests

0.0% Duplications on 1.2k Lines — 0 Duplicated Blocks

> We have also achieved high grades for code quality on our frontend. There is some code smell, however, this appears to be accumulated from some instances of commented out code and uses of `var` instead of `let` rather than anything particularly critical that could cause a problem for either security or maintainability.



0 Bugs — Reliability A

0 Vulnerabilities — Security A

0 Security Hotspots — — Reviewed — Security Review A

2h 8min Debt — 30 Code Smells — Maintainability A

Recommendations for Improvement

> Shopping Cart implementation may not have been optimal, as Shopping carts were stored based on username rather than ID, which may cause issues if for whatever reason there is a need for changing usernames in the future. There were also some odd request methods for different cart functionalities, e.g. PUT request clearing cart. The rest of our project turned out to work pretty smoothly and we did not wish to change much else.

Design Principles Analysis

> **Single Responsibility Principle** - The JerseyController, CartController, and UserController are examples of where this principle has been implemented. Each controller's purpose is to handle their respective subject's API requests and provide responses using HTTP protocols. The management of the subject's data is taken care of in the JerseyDAO, CartDAO, and UserDAO.

> **Dependency Inversion Principle** - We have also used multiple services in the UI JerseyService, CartService, AuthGuardService, AuthenticationService which are injected into various components which allows the logic of the UI to be independent from where the https requests are sent.

> **Open/Closed Principle** - This is demonstrated in the JerseyController, CartController, and AuthController because they each take in higher level abstractions of JerseyDAO, CartDAO, and UserDao. JerseyFileDAO, CartFileDAO, and UserFileDAO are injected into their respective controllers and ones that need access to their subject's data. This allows us to test the Controllers and FileDAO independently. It also provides a mechanism for extending functionality without modification because the data is managed separately from the controllers.

Testing

> *This section will provide information about the testing performed and the results of the testing.*

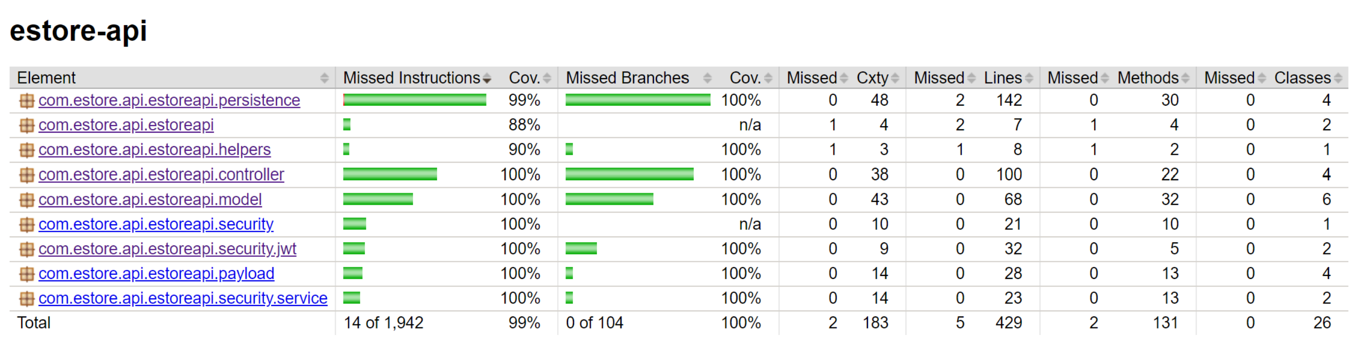
Acceptance Testing

> We have 20 user stories and 29 acceptance criteria tests. All user stories have at least one test. As of Sprint 4 all acceptance criteria tests are passing. Some issues that we found during testing was that testing criteria sometimes overlaped between user stories and this could be improved on by being more concise.

Unit Testing and Code Coverage

> Our team's unit testing strategy was to have as much code coverage as possible. We met our goal and got an overall code coverage of 99%.

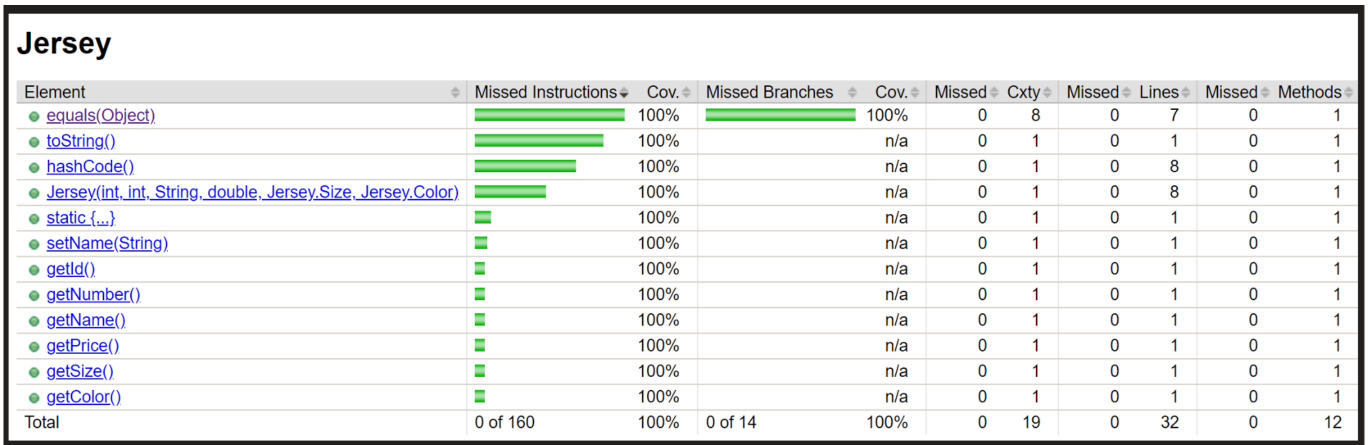
Overview



estore-api

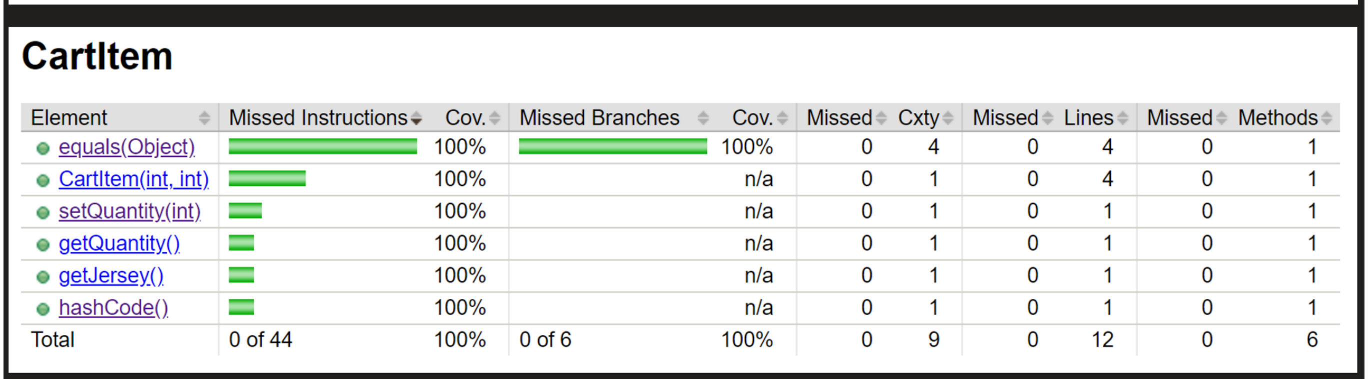
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.estore.api.estoreapi.persistence	🟩	99%	🟩	100%	0	48	2	142	0	30	0	4
com.estore.api.estoreapi	🟩	88%		n/a	1	4	2	7	1	4	0	2
com.estore.api.estoreapi.helpers	🟩	90%	🟩	100%	1	3	1	8	1	2	0	1
com.estore.api.estoreapi.controller	🟩	100%	🟩	100%	0	38	0	100	0	22	0	4
com.estore.api.estoreapi.model	🟩	100%	🟩	100%	0	43	0	68	0	32	0	6
com.estore.api.estoreapi.security	🟩	100%		n/a	0	10	0	21	0	10	0	1
com.estore.api.estoreapi.security.jwt	🟩	100%	🟩	100%	0	9	0	32	0	5	0	2
com.estore.api.estoreapi.payload	🟩	100%	🟩	100%	0	14	0	28	0	13	0	4
com.estore.api.estoreapi.security.service	🟩	100%	🟩	100%	0	14	0	23	0	13	0	2
Total	14 of 1,942	99%	0 of 104	100%	2	183	5	429	2	131	0	26

Model



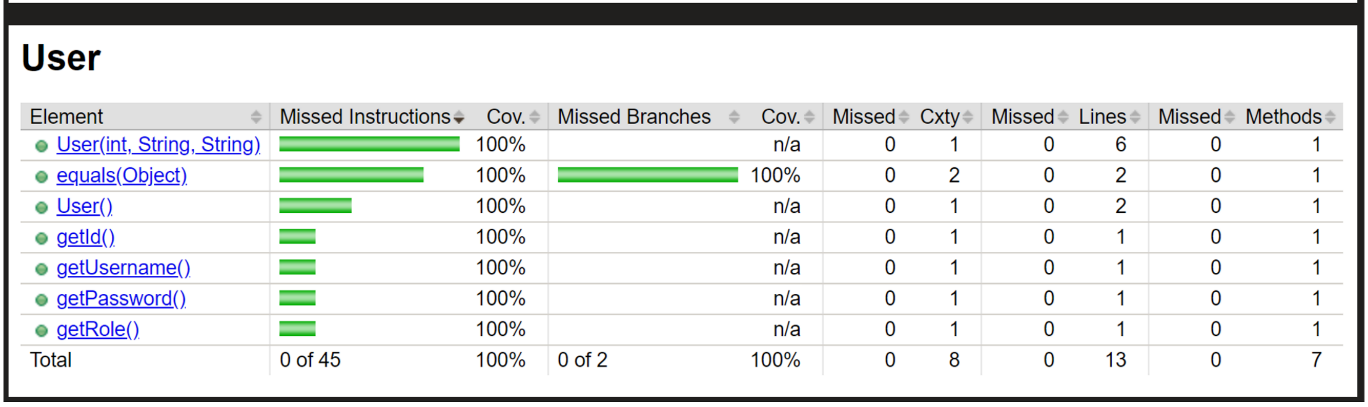
Jersey

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
equals(Object)	🟩	100%	🟩	100%	0	8	0	7	0	1
toString()	🟩	100%		n/a	0	1	0	1	0	1
hashCode()	🟩	100%		n/a	0	1	0	8	0	1
Jersey(int, int, String, double, Jersey.Size, Jersey.Color)	🟩	100%		n/a	0	1	0	8	0	1
static {...}	🟩	100%		n/a	0	1	0	1	0	1
setName(String)	🟩	100%		n/a	0	1	0	1	0	1
getId()	🟩	100%		n/a	0	1	0	1	0	1
getNumber()	🟩	100%		n/a	0	1	0	1	0	1
getName()	🟩	100%		n/a	0	1	0	1	0	1
getPrice()	🟩	100%		n/a	0	1	0	1	0	1
getSize()	🟩	100%		n/a	0	1	0	1	0	1
getColor()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 160	100%	0 of 14	100%	0	19	0	32	0	12



CartItem

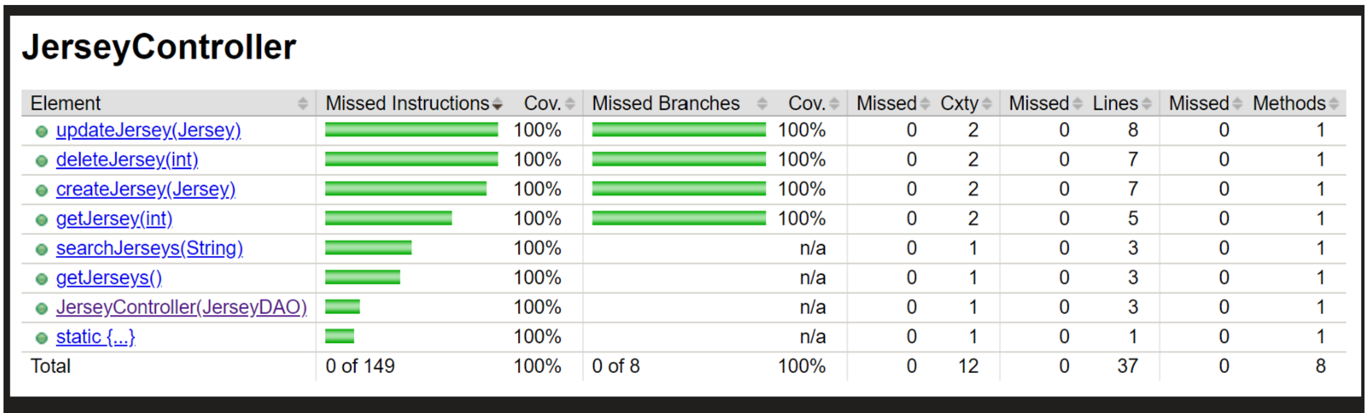
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
equals(Object)	🟩	100%	🟩	100%	0	4	0	4	0	1
CartItem(int, int)	🟩	100%		n/a	0	1	0	4	0	1
setQuantity(int)	🟩	100%		n/a	0	1	0	1	0	1
getQuantity()	🟩	100%		n/a	0	1	0	1	0	1
getJersey()	🟩	100%		n/a	0	1	0	1	0	1
hashCode()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 44	100%	0 of 6	100%	0	9	0	12	0	6



User

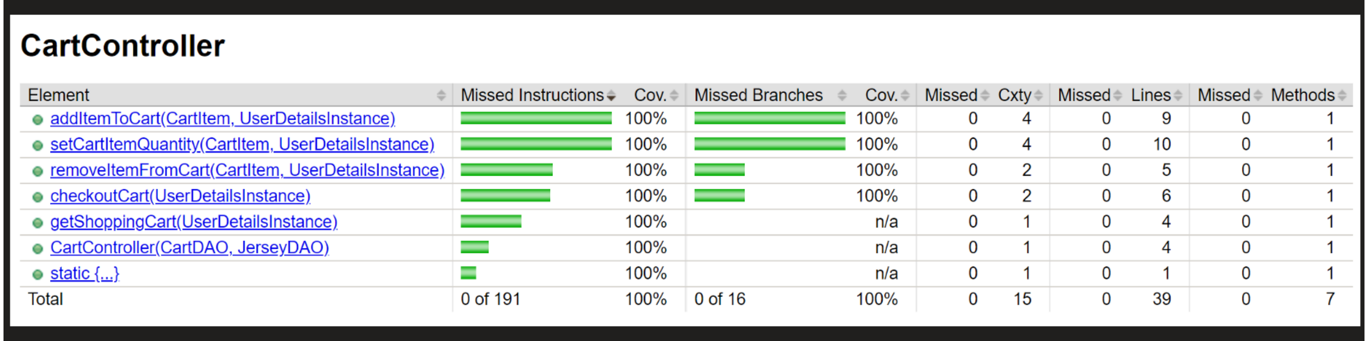
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
User(int, String, String)	🟩	100%		n/a	0	1	0	6	0	1
equals(Object)	🟩	100%	🟩	100%	0	2	0	2	0	1
User()	🟩	100%		n/a	0	1	0	2	0	1
getId()	🟩	100%		n/a	0	1	0	1	0	1
getUsername()	🟩	100%		n/a	0	1	0	1	0	1
getPassword()	🟩	100%		n/a	0	1	0	1	0	1
getRole()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 45	100%	0 of 2	100%	0	8	0	13	0	7

Controller



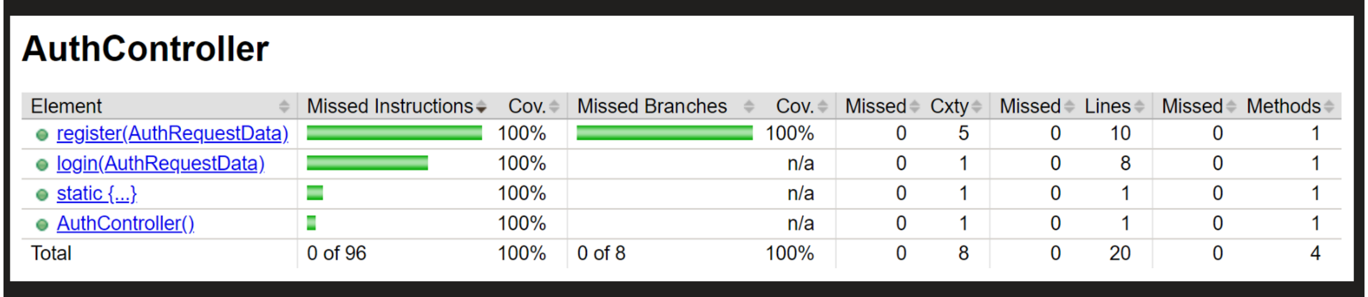
JerseyController

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
updateJersey(Jersey)	🟩	100%	🟩	100%	0	2	0	8	0	1
deleteJersey(int)	🟩	100%	🟩	100%	0	2	0	7	0	1
createJersey(Jersey)	🟩	100%	🟩	100%	0	2	0	7	0	1
getJersey(int)	🟩	100%	🟩	100%	0	2	0	5	0	1
searchJerseys(String)	🟩	100%		n/a	0	1	0	3	0	1
getJerseys()	🟩	100%		n/a	0	1	0	3	0	1
JerseyController(JerseyDAO)	🟩	100%		n/a	0	1	0	3	0	1
static {...}	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 149	100%	0 of 8	100%	0	12	0	37	0	8



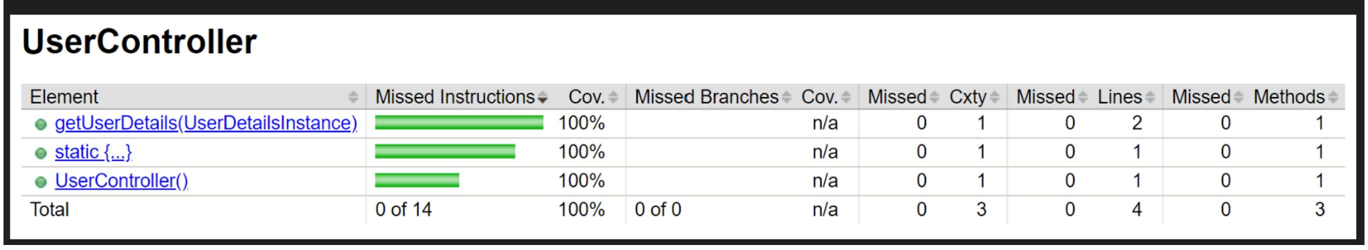
CartController

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
addItemToCart(CartItem, UserDetailsInstance)	🟩	100%	🟩	100%	0	4	0	9	0	1
setCartItemQuantity(CartItem, UserDetailsInstance)	🟩	100%	🟩	100%	0	4	0	10	0	1
removeItemFromCart(CartItem, UserDetailsInstance)	🟩	100%	🟩	100%	0	2	0	5	0	1
checkoutCart(UserDetailsInstance)	🟩	100%	🟩	100%	0	2	0	6	0	1
getShoppingCart(UserDetailsInstance)	🟩	100%		n/a	0	1	0	4	0	1
CartController(CartDAO, JerseyDAO)	🟩	100%		n/a	0	1	0	4	0	1
static {...}	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 191	100%	0 of 16	100%	0	15	0	39	0	7



AuthController

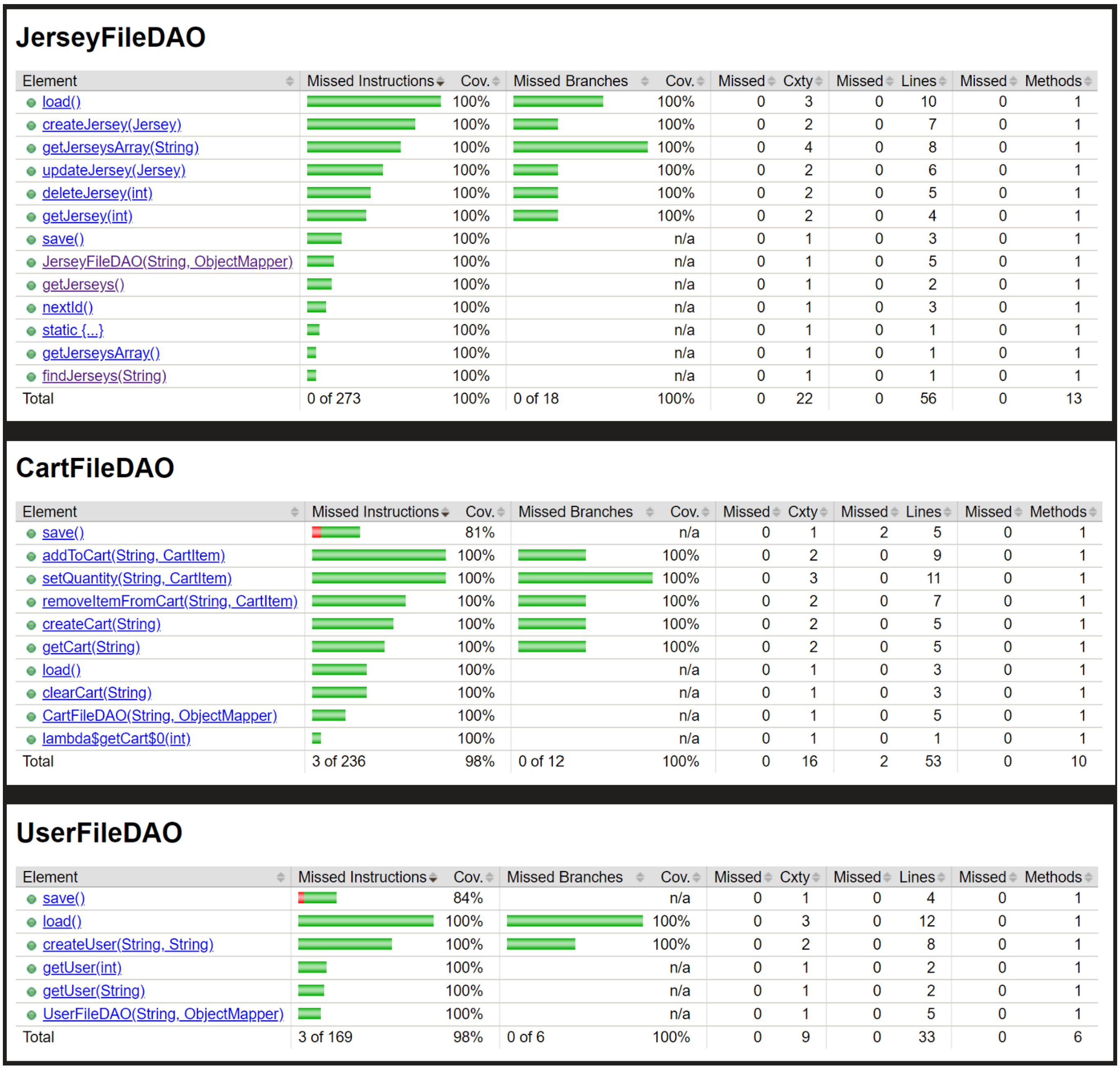
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
register(AuthRequestData)	🟩	100%	🟩	100%	0	5	0	10	0	1
login(AuthRequestData)	🟩	100%		n/a	0	1	0	8	0	1
static {...}	🟩	100%		n/a	0	1	0	1	0	1
AuthController()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 96	100%	0 of 8	100%	0	8	0	20	0	4



UserController

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getUserDetails(UserDetailsInstance)	🟩	100%		n/a	0	1	0	2	0	1
static {...}	🟩	100%		n/a	0	1	0	1	0	1
UserController()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 14	100%	0 of 0	n/a	0	3	0	4	0	3

Persistence



JerseyFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
load()		100%		100%	0	3	0	10	0	1
createJersey(Jersey)		100%		100%	0	2	0	7	0	1
getJerseysArray(String)		100%		100%	0	4	0	8	0	1
updateJersey(Jersey)		100%		100%	0	2	0	6	0	1
deleteJersey(int)		100%		100%	0	2	0	5	0	1
getJersey(int)		100%		100%	0	2	0	4	0	1
save()		100%		n/a	0	1	0	3	0	1
JerseyFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
getJerseys()		100%		n/a	0	1	0	2	0	1
nextId()		100%		n/a	0	1	0	3	0	1
static {...}		100%		n/a	0	1	0	1	0	1
getJerseysArray()		100%		n/a	0	1	0	1	0	1
findJerseys(String)		100%		n/a	0	1	0	1	0	1
Total	0 of 273	100%	0 of 18	100%	0	22	0	56	0	13

CartFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
save()		81%		n/a	0	1	2	5	0	1
addToCart(String, CartItem)		100%		100%	0	2	0	9	0	1
setQuantity(String, CartItem)		100%		100%	0	3	0	11	0	1
removeItemFromCart(String, CartItem)		100%		100%	0	2	0	7	0	1
createCart(String)		100%		100%	0	2	0	5	0	1
getCart(String)		100%		100%	0	2	0	5	0	1
load()		100%		n/a	0	1	0	3	0	1
clearCart(String)		100%		n/a	0	1	0	3	0	1
CartFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
lambda$getCart$0(int)		100%		n/a	0	1	0	1	0	1
Total	3 of 236	98%	0 of 12	100%	0	16	2	53	0	10

UserFileDAO

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
save()		84%		n/a	0	1	0	4	0	1
load()		100%		100%	0	3	0	12	0	1
createUser(String, String)		100%		100%	0	2	0	8	0	1
getUser(int)		100%		n/a	0	1	0	2	0	1
getUser(String)		100%		n/a	0	1	0	2	0	1
UserFileDAO(String, ObjectMapper)		100%		n/a	0	1	0	5	0	1
Total	3 of 169	98%	0 of 6	100%	0	9	0	33	0	6