Logan Mann
Naif Alassaf
CSCI 4448
4/26/22

Final Project Report

Project Name: CloudCanvas

Team Members: Logan Mann, Naif Alassaf

State of System:

Currently, our system is fully implemented with all the functional requirements, constraints, and use cases we set out to include in Project 5. We have a fully functioning authentication service which allows users to create accounts and login, and issues JSON Web Tokens to verify subsequent API calls via the desktop client. Users are able to create canvases via a canvas creation page, where they can specify a unique canvas name, and designate the canvas as either public or private. The users can select a canvas to draw on out of the list of all the canvases they have available on the service, change their pen color and the shape type, and can add a variety of shapes to their canvases, which update in real time on each client as other users add shapes to them. We were also able to implement a stretch goal of implementing API verification via JWTs. The one change between our proposal in Project 5 and the final deliverable in Project 7, is that we opted to simply have a map of canvases stored in memory on the backend, and write the data to a JSON file on disk when the backend service was shut down, rather than persisting the data to S3.

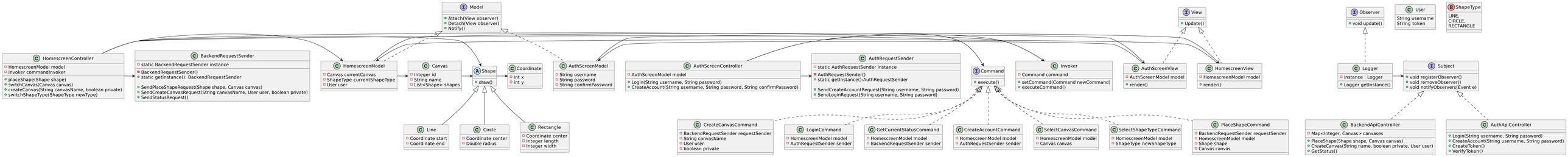
Final Class Diagram and Comparison Statement

Since our design work was submitted in Project 5 and 6, we ended up using more additional patterns than we originally included in our Project 5 class diagram. In addition to the Command, MVC, Observer, and Singleton patterns which made it into our final system, we also found that the Strategy pattern fit very well into our system, as we required different strategies for mouse listeners and for building menu UI components depending on what shape type the user had selected, so we were able to incorporate the Strategy pattern very seamlessly to handle the problem of keeping these different function

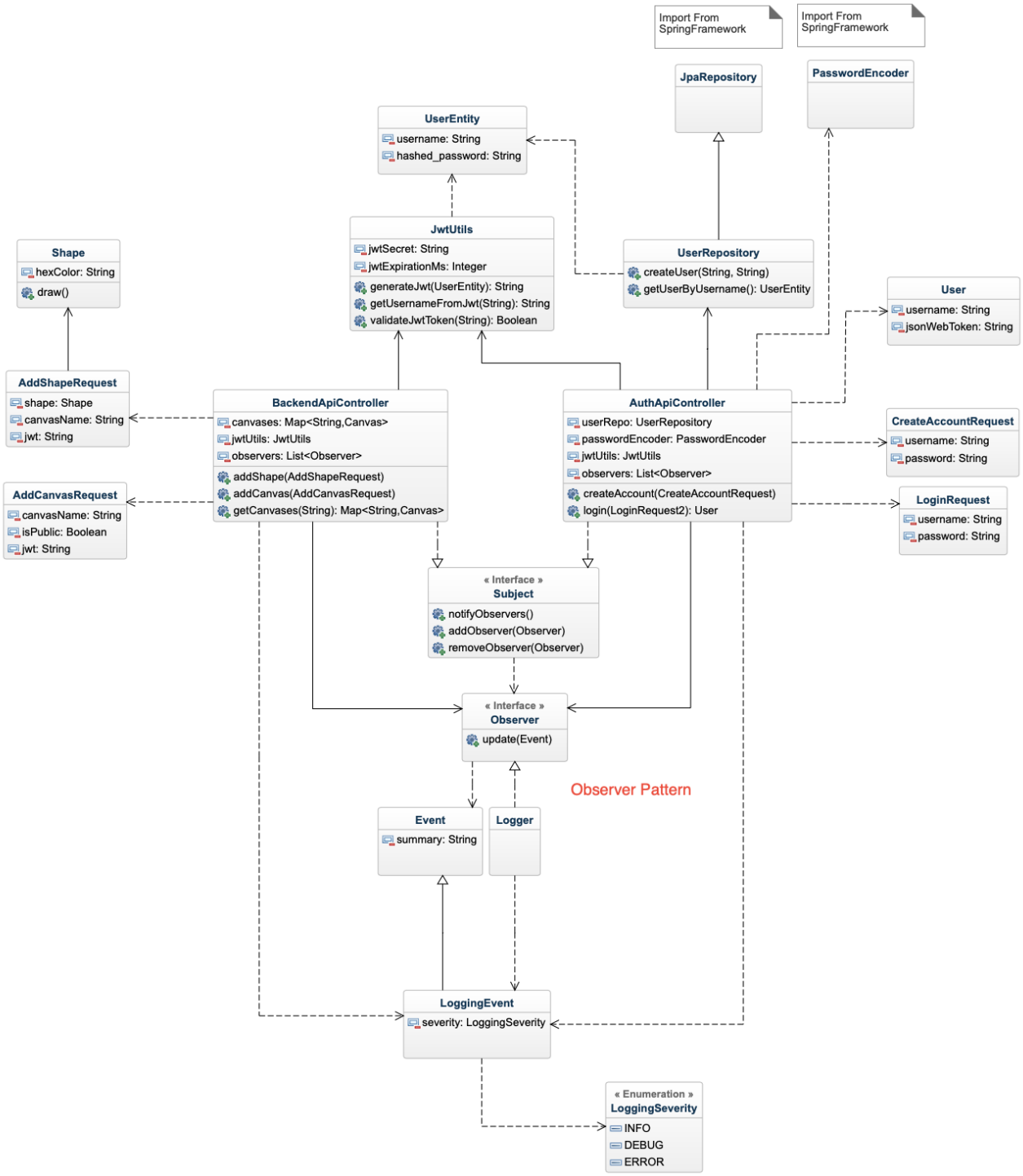
implementations decoupled from other components, easily switched out when the user changed shape types, and in modules with tight cohesion specifically geared towards different shape creation strategies. Additionally, we were able to use the Factory pattern to keep the instantiation of these ShapeCreationStrategy objects decoupled from our MVC classes which made up the bulk of our client application, reducing these classes' dependency on any given ShapeCreationStrategy implementation, and hiding the implementation details of each strategy from the classes that used them. We also ended up adding additional "subview" classes which represented smaller UI components that made up the larger View objects that we initially included in our Project 5 design. Doing that kept the cohesion relatively tight within our various View classes, ensuring each one had the responsibility of rendering a fairly self-contained UI component. If we hadn't done this, our main HomescreenView class would have quickly gotten far far too massive in size. Finally, we also added a JWTUtils class to our backend services, which allowed us to increase our application's security, and a StateSaver class which is responsible for loading and saving canvas state to and from disk.

Note: We separated our final client and backend services into two separate UML diagrams for the sake of readability.

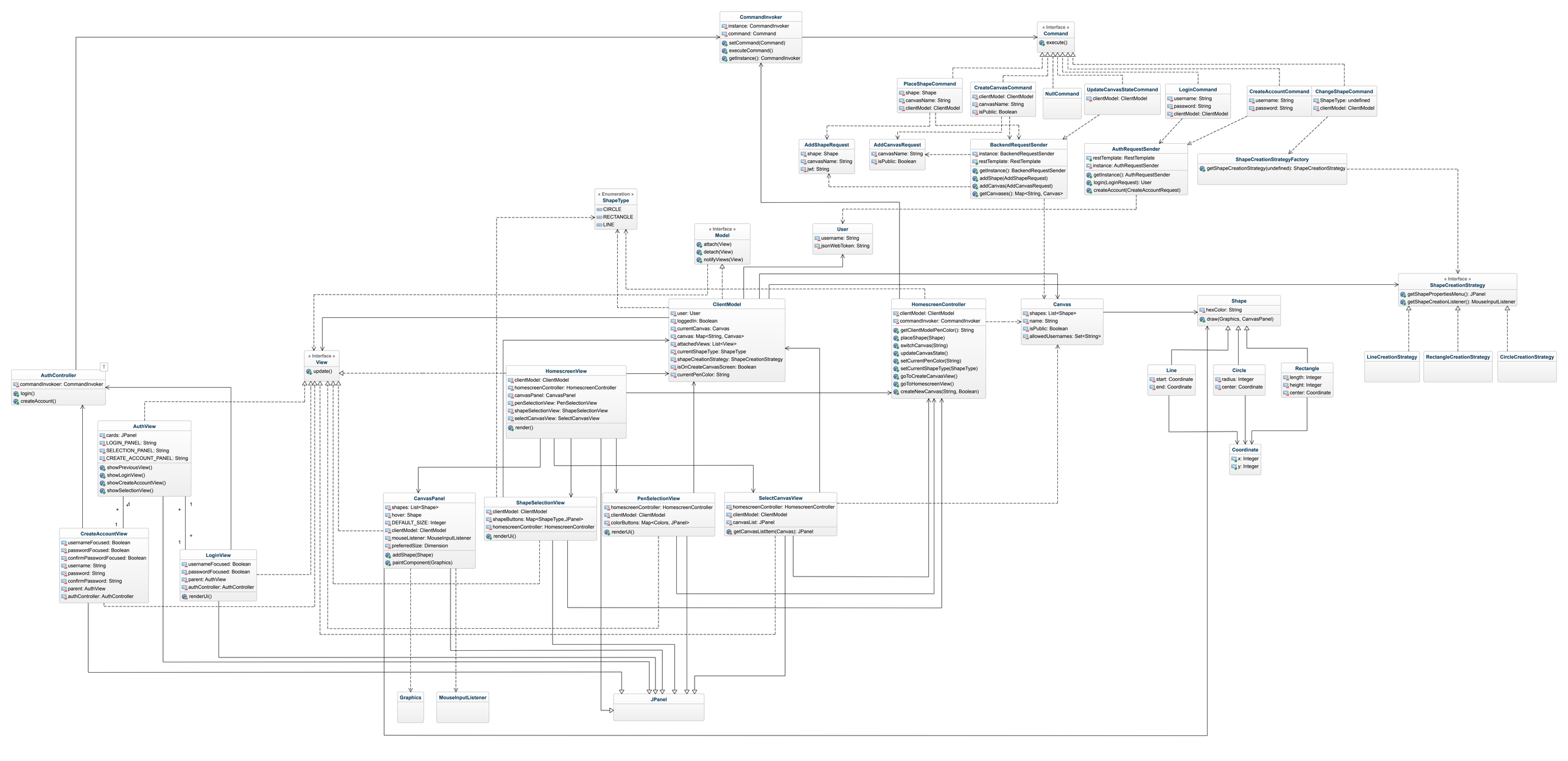
Initial Project 5 class diagram is added as a separate page below for readability:


Model
+ Attach(View observer)
+ Detach(View observer)
+ Notify()
View
+ Update()
Observer
+ void update()
User
String username
String token
ShapeType
LINE,
CIRCLE,
RECTANGLE
HomescreenController
- HomescreenModel model
- Invoker commandInvoker
+ placeShape(Shape shape)
+ switchCanvas(Canvas canvas)
+ createCanvas(String canvasName, boolean private)
+ switchShapeType(ShapeType newType)
BackendRequestSender
- static BackendRequestSender instance
- BackendRequestSender()
+ static getInstance(): BackendRequestSender
+ SendPlaceShapeRequest(Shape shape, Canvas canvas)
+ SendCreateCanvasRequest(String canvasName, User user, boolean private)
+ SendStatusRequest()
HomescreenModel
- Canvas currentCanvas
- ShapeType currentShapeType
- User user
Canvas
- Integer id
- String name
- List<Shape> shapes
Shape
+ draw()
Coordinate
- int x
- int y
Line
- Coordinate start
- Coordinate end
Circle
- Coordinate center
- Double radius
Rectangle
- Coordinate center
- Integer length
- Integer width
AuthScreenModel
- String username
- String password
- String confirmPassword
AuthScreenController
- AuthScreenModel model
+ Login(String username, String password)
+ CreateAccount(String username, String password, String confirmPassword)
AuthRequestSender
- static AuthRequestSender instance
- AuthRequestSender()
+ static getInstance():AuthRequestSender
+ SendCreateAccountRequest(String username, String password)
+ SendLoginRequest(String username, String password)
Command
+ execute()
Invoker
- Command command
+ setCommand(Command newCommand)
+ executeCommand()
AuthScreenView
- AuthScreenModel model
+ render()
HomescreenView
- HomescreenModel model
+ render()
Logger
- instance : Logger
+ Logger getInstance()
Subject
+ void registerObserver()
+ void removeObserver()
+ void notifyObservers(Event e)
CreateCanvasCommand
- BackendRequestSender requestSender
- String canvasName
- User user
- boolean private
LoginCommand
- HomescreenModel model
- AuthRequestSender sender
GetCurrentStatusCommand
- HomescreenModel model
- BackendRequestSender sender
CreateAccountCommand
- HomescreenModel model
- AuthRequestSender sender
SelectCanvasCommand
- HomescreenModel model
- Canvas canvas
SelectShapeTypeCommand
- HomescreenModel model
- ShapeType newShapeType
PlaceShapeCommand
- BackendRequestSender requestSender
- HomescreenModel model
- Shape shape
- Canvas canvas
BackendApiController
- Map<Integer, Canvas> canvases
+ PlaceShape(Shape shape, Canvas canvas)
+ CreateCanvas(String name, boolean private, User user)
+ GetStatus()
AuthApiController
+ Login(String username, String password)
+ CreateAccount(String username, String password)
+ CreateToken()
+ VerifyToken()

Final Backend UML Class Diagram:



Final Client UML Class Diagram Is Added as a Custom Sized Page For Readability:


CommandInvoker
instance: CommandInvoker
command: Command
setCommand(Command)
executeCommand()
getInstance(): CommandInvoker
« Interface »
Command
execute()
PlaceShapeCommand
shape: Shape
canvasName: String
clientModel: ClientModel
CreateCanvasCommand
clientModel: ClientModel
canvasName: String
isPublic: Boolean
NullCommand
UpdateCanvasStateCommand
clientModel: ClientModel
LoginCommand
username: String
password: String
clientModel: ClientModel
CreateAccountCommand
username: String
password: String
ChangeShapeCommand
ShapeType: undefined
clientModel: ClientModel
AddShapeRequest
shape: Shape
canvasName: String
jwt: String
AddCanvasRequest
canvasName: String
isPublic: Boolean
BackendRequestSender
instance: BackendRequestSender
restTemplate: RestTemplate
getInstance(): BackendRequestSender
addShape(AddShapeRequest)
addCanvas(AddCanvasRequest)
getCanvases(): Map<String, Canvas>
AuthRequestSender
restTemplate: RestTemplate
instance: AuthRequestSender
getInstance(): AuthRequestSender
login(LoginRequest): User
createAccount(CreateAccountRequest)
ShapeCreationStrategyFactory
getShapeCreationStrategy(undefined): ShapeCreationStrategy
« Enumeration »
ShapeType
CIRCLE
RECTANGLE
LINE
« Interface »
Model
attach(View)
detach(View)
notifyViews(View)
User
username: String
jsonWebToken: String
« Interface »
ShapeCreationStrategy
getShapePropertiesMenu(): JPanel
getShapeCreationListener(): MouseInputListener
LineCreationStrategy
RectangleCreationStrategy
CircleCreationStrategy
ClientModel
user: User
loggedIn: Boolean
currentCanvas: Canvas
canvas: Map<String, Canvas>
attachedViews: List<View>
currentShapeType: ShapeType
shapeCreationStrategy: ShapeCreationStrategy
isOnCreateCanvasScreen: Boolean
currentPenColor: String
HomescreenController
clientModel: ClientModel
commandInvoker: CommandInvoker
getClientModelPenColor(): String
placeShape(Shape)
switchCanvas(String)
updateCanvasState()
setCurrentPenColor(String)
setCurrentShapeType(ShapeType)
goToCreateCanvasView()
goToHomescreenView()
createNewCanvas(String, Boolean)
Canvas
shapes: List<Shape>
name: String
isPublic: Boolean
allowedUsernames: Set<String>
Shape
hexColor: String
draw(Graphics, CanvasPanel)
Line
start: Coordinate
end: Coordinate
Circle
radius: Integer
center: Coordinate
Rectangle
length: Integer
height: Integer
center: Coordinate
Coordinate
x: Integer
y: Integer
« Interface »
View
update()
AuthController
commandInvoker: CommandInvoker
login()
createAccount()
HomescreenView
clientModel: ClientModel
homescreenController: HomescreenController
canvasPanel: CanvasPanel
penSelectionView: PenSelectionView
shapeSelectionView: ShapeSelectionView
selectCanvasView: SelectCanvasView
render()
AuthView
cards: JPanel
LOGIN_PANEL: String
SELECTION_PANEL: String
CREATE_ACCOUNT_PANEL: String
showPreviousView()
showLoginView()
showCreateAccountView()
showSelectionView()
CanvasPanel
shapes: List<Shape>
hover: Shape
DEFAULT_SIZE: Integer
clientModel: ClientModel
mouseListener: MouseInputListener
preferredSize: Dimension
addShape(Shape)
paintComponent(Graphics)
ShapeSelectionView
clientModel: ClientModel
shapeButtons: Map<ShapeType,JPanel>
homescreenController: HomescreenController
renderUi()
PenSelectionView
homescreenController: HomescreenController
clientModel: ClientModel
colorButtons: Map<Colors, JPanel>
renderUi()
SelectCanvasView
homescreenController: HomescreenController
clientModel: ClientModel
canvasList: JPanel
getCanvasListItem(Canvas): JPanel
CreateAccountView
usernameFocused: Boolean
passwordFocused: Boolean
confirmPasswordFocused: Boolean
username: String
password: String
confirmPassword: String
parent: AuthView
authController: AuthController
LoginView
usernameFocused: Boolean
passwordFocused: Boolean
parent: AuthView
authController: AuthController
renderUi()
Graphics
MouseInputListener
JPanel
T
1
*

Third Party Code vs. Original Code Statement

The vast majority of the code in our final submission is original. All of our MVC classes, Canvas, User, Shape, and other data model classes, our RequestSenders, RestControllers, Logger/Observer classes, Strategy, Command, and Factory pattern classes, etc. etc. are all original. We did reference external documentation and tutorials for using frameworks like Spring and Swing however, and have detailed these references below:

Swing Documentation for UI code:

https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

Baeldung.com and Spring/SpringBoot Documentation for Spring Framework:

https://www.baeldung.com/

https://docs.spring.io/spring-framework/docs/current/reference/html/

Our JSON Web Token Utility class referenced this example for signing/verifying JWTs for user auth (also annotated in our code comments):

https://www.bezkoder.com/spring-boot-jwt-authentication/

Statement on OOAD Process

Throughout the development process of our final project, a design process technique that we really felt was effective in helping us build our final system, was to break our system down into its core elements. Once we had an idea of the core pieces of functionality that we wanted our service to allow, working to break these use cases up into their fundamental modules was very effective in helping us break an intimidatingly large number of pieces of functionality into easily digestible chunks, which could then be implemented as classes with high cohesion. Additionally, during development, we encountered unforeseen technical problems (most notably finding a clean way to deal with different methods/UI components depending on the shape type the user was using), and trying to find a OO design pattern that solved this problem (eventually we used the strategy pattern) in an elegant way was very interesting, as

we found ourselves using the concepts and patterns learned in class to solve real-world technical problems. Finally, in applying our patterns and integrating the various components of our system in general, a design process element that we found particularly helpful was to look for intuitive relationships between the various entities in our system (i.e. RestControllers notify loggers of key actions, our RequestSenders act as an intermediary between our backend service and the rest of our client application, etc.). This allowed us to connect all of our classes in ways that were easy to follow, and in ways that allowed for highly decoupled modules throughout the system.