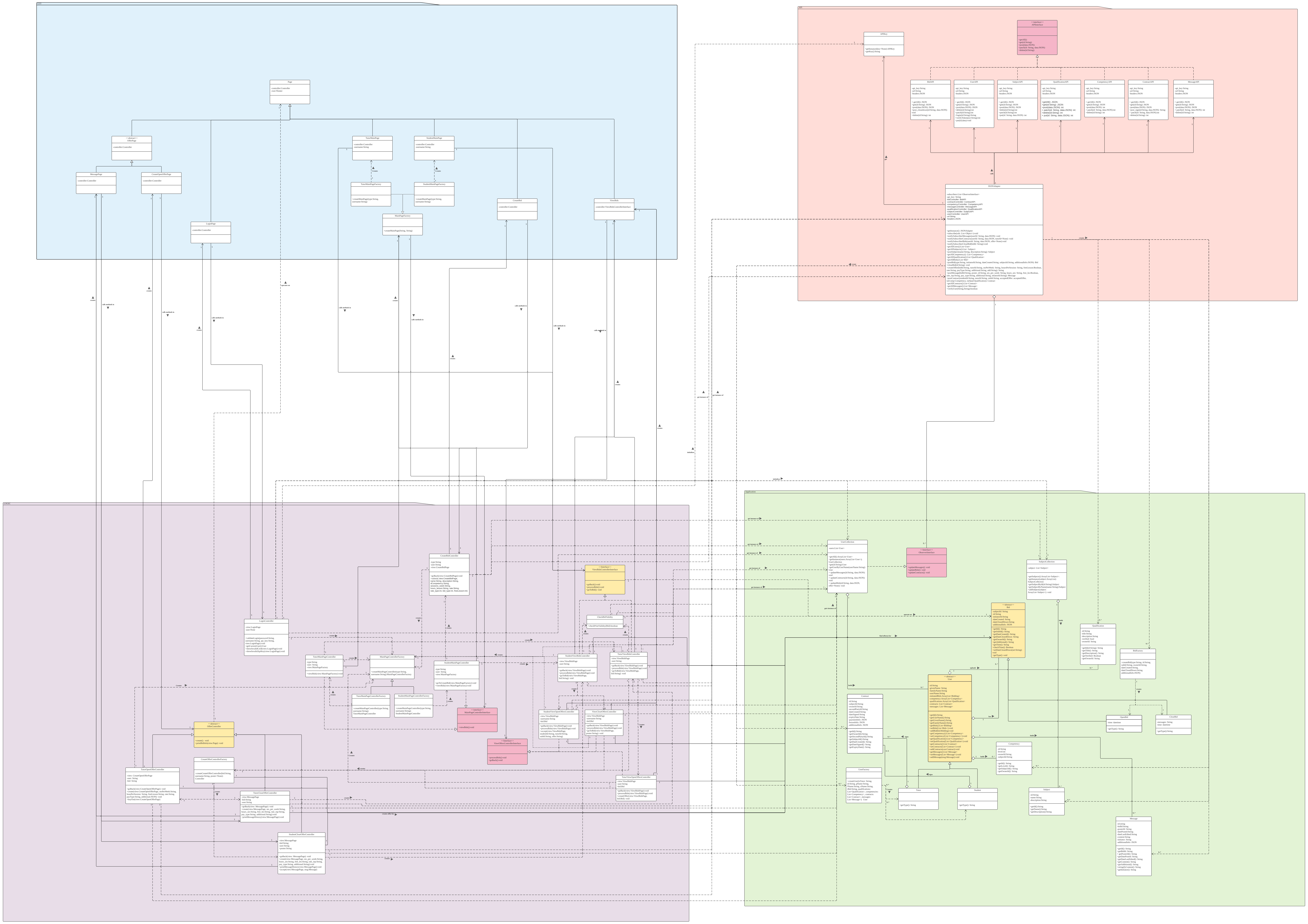
GUI
Page
-controller:Controller
-root:Tkinter
<<abstract>> OfferPage
-controller:Controller
MessagePage
-controller:Controller
CreateOpenOfferPage
-controller:Controller
LoginPage
-controller:Controller
TutorMainPage
-controller:Controller
-username:String
StudentMainPage
-controller:Controller
-username:String
Creates
TutorMainPageFactory
+createMainPage(type:String, username:String)
StudentMainPageFactory
+createMainPage(type:String, username:String)
MainPageFactory
+createMainPage(String, String)
CreateBid
-controller:Controller
ViewBids
-controller:ViewBidsControllerInterface
creates
calls methods in
operates on
API
APIKey
+getInstance(key=None):APIKey
+getKey():String
<<interface>> APIInterface
+getAll():
+get(id:String):
+post(data:JSON):
+patch(id: String, data:JSON):
+delete(id:String):
BidAPI
-api_key:String
-url:String
-headers:JSON
+ getAll(): JSON
+get(id:String): JSON
+post(data:JSON): JSON
+post_closedown(id:String, data:JSON): void
+delete(id:String): int
UserAPI
-api_key:String
-url:String
-headers:JSON
+ getAll(): JSON
+get(id:String): JSON
+post(data:JSON): JSON
+delete(id:String):int
+patch(id:String):int
+login(id:String):String
+verifyToken(jwt:String):int
+put(id,data):void
SubjectAPI
-api_key:String
-url:String
-headers:JSON
+ getAll(): JSON
+get(id:String): JSON
+post(data:JSON): JSON
+delete(id:String):int
+patch(id:String):int
+put(id: String, data:JSON): int
QualificationAPI
-api_key:String
-url:String
-headers:JSON
+getAll(): JSON
+get(id:String): JSON
+post(data:JSON): int
+ patch(id: String, data:JSON): int
+delete(id:String): int
+ put(id: String, data:JSON): int
CompetencyAPI
-api_key:String
-url:String
-headers:JSON
+ getAll(): JSON
+get(id:String): JSON
+post(data:JSON): int
+ patch(id: String, data:JSON):int
+delete(id:String): int
ContractAPI
-api_key:String
-url:String
-headers:JSON
+ getAll(): JSON
+get(id:String): JSON
+post(data:JSON): JSON
+post_sign(id:String, data:JSON): String
+ patch(id: String, data:JSON):int
+delete(id:String): int
MessageAPI
-api_key:String
-url:String
-headers:JSON
+ getAll(): JSON
+get(id:String): JSON
+post(data:JSON): int
+ patch(id: String, data:JSON): int
+delete(id:String): int
get
calls
JSONAdapter
-subscribers:List<ObserverInterface>
-api_key: String
-bidController: BidAPI
-contractController: ContractAPI
-competencyController: CompetencyAPI
-messageController: MessageAPI
-qualificationController: QualificationAPI
-subjectController: SubjectAPI
-userController: UserAPI
-url:String
-headers:JSON
+getInstance(): JSONAdapter
+subscribe(sub: List<Object>):void
+notifySubscriberMessages(userId: String, data:JSON): void
+notifySubscriberContracts(userId: String, data:JSON, tutorId=None): void
+notifySubscriberBids(userId: String, data:JSON, offer=None):void
+notifySubscriberCloseBid(bidId: String):void
+getAllUsers():List<User>
+getAllSubjects():List< Subject>
+postSubject(name:String, description:String): Subject
+getAllCompetency(): List<Competency>
+getAllQualification():List<Qualification>
+getAllBids():List<Bid>
+postBid(type:String, initiatorId:String, dateCreated:String, subjectId:String, additionalInfo:JSON): Bid
+closeBid(id:String): void
+createOffer(bidId:String, tutorId:String, sesPerWeek: String, hoursPerSession: String, freeLesson:Boolean, rate:String, payType:String, additional:String, add:String): String
+postMessage(bidId:String, poster_id:String, ses_per_week: String, hours_ses: String, free_les:Boolean, rate_inp:String, pay_type:String, additional:String, intiatorId:String): Message
+postContract(studentId:String, tutorId:String, subId:String, acceptedOffer: acceptedOffer, tutComp:Competency, tutQual:Qualification): Contract
+getAllContracts():List<Contract>
+getAllMessages():List<Message>
+verifyUser(String,String):boolean
create
creates
get instance of
initializes
Controller
LoginController
-view:LoginPage
-user:None
+validateLogin(password:String, username:String, api_key:String, view:LoginPage):void
+getCurrentUser():User
+showInvalidCred(view:LoginPage):void
+showInvalidApiKey(view:LoginPage):void
CreateBidController
-type:String
-user:String
-view:CreateBidPage
+goBack(view:CreateBidPage):void
+submit(view:CreateBidPage, name:String, description:String, dCompetency:String, sessions_week:String, hours_lesson:String, rate:String, rate_type:int, bid_type:int, freeLesson:int)
TutorMainPageController
-type:String
-user: String
-view:MainPageFactory
+viewBids(view:MainPageFactory):void
MainPageControllerFactory
+createMainPageController(type:String, username:String):MainPageControllerFactory
StudentMainPageController
-type:String
-user: String
-view:MainPageFactory
+goToCreateBid(view:MainPageFactory):void
+viewBids(view:MainPageFactory):void
TutorMainPageControllerFactory
+createMainPageController(type:String, username:String): TutorMainPageController
StudentMainPageControllerFactory
+createMainPageController(type:String, username:String): StudentMainPageController
<<interface>> MainPageControllerInterface
+viewBids():void
<<Interface>> ViewBidsControllerInterface
+goBack():void
+processBids():void
+goToBid(): void
CheckBidValidity
+checkPrintValidity(Bid):boolean
StudentViewBidsController
-view:ViewBidsPage
-user:String
+goBack(view:ViewBidsPage):void
+processBids(view:ViewBidsPage):void
+goToBid(view:ViewBidsPage, bid:String): void
TutorViewBidsController
-view:ViewBidsPage
-user:String
+goBack(view:ViewBidsPage):void
+processBids(view:ViewBidsPage):void
+goToBid(view:ViewBidsPage, bid:String): void
StudentViewOpenOffersController
-view:ViewBidsPage
-username:String
-bid:Bid
+goBack(view:ViewBidsPage):void
+processBids(view:ViewBidsPage):void
+accept(view:ViewBidsPage, studentId:String, tutorId:String, subId:String, offer:String)
ViewCloseOffersController
-view:ViewBidsPage
-username:String
-bid:Bid
+goBack(view:ViewBidsPage):void
+processBids(view:ViewBidsPage):void
+goToBid(view:ViewBidsPage, poster:String): void
<<Interface>> ViewOffersControllerInterface
+processBids():void
+goBack():void
TutorViewOpenOffersController
-view:ViewBidsPage
-user:String
-bid:Bid
+goBack(view:ViewBidsPage):void
+processBids(view:ViewBidsPage):void
+createOffer(view:ViewBidsPage, bid:Bid): void
<<abstract>> OfferController
+create(): void
+printBidInfo(view:Page): void
CreateOfferControllerFactory
+createCreateOfferController(bid:String, username:String, poster=None): Controller
TutorOpenOfferController
-view: CreateOpenOfferPage
-user: String
-bid: String
+goBack(view:CreateOpenOfferPage): void
+create(view:CreateOpenOfferPage, sesPerWeek:String, housPerSession: String, freeLesson:String, rate:String, payType:String, additional:JSON): void
+buyOut(view:CreateOpenOfferPage)
TutorCloseOfferController
-view:MessagePage
-bid:String
-user:String
+goBack(view: MessagePage): void
+create(view:MessagePage, ses_per_week:String, hours_ses:String, free_les:String, rate_inp:String, pay_type:String, additional:String):void
+printMessageHistory(view:MessagePage):void
StudentCloseOfferController
-view:MessagePage
-bid:String
-user:String
-poster:String
+goBack(view: MessagePage): void
+create(view:MessagePage, ses_per_week:String, hours_ses:String, free_les:String, rate_inp:String, pay_type:String, additional:String):void
+printMessageHistory(view:MessagePage):void
+accept(view:MessagePage, msg:Message)
creates
Creates
creates offer for
Application
UserCollection
-users:List<User>
+getAll():ArrayList<User>
+getInstance(users:ArrayList<User>); UserCollection
+get(id:String):User
+getUserByUserName(userName:String): User
+ updateMessages(id:String, data:JSON): void
+ updateContracts(id:String, data:JSON): void
+ updateBids(id:String, data:JSON, offer=None): void
<<Interface>> ObserverInterface
+updateMessages(): void
+updateBids(): void
+updateContracts(): void
SubjectCollection
-subject: List<Subject>
+getSubjects():ArrayList<Subject>:
+getInstance(subject:ArrayList): SubjectCollection
+getSubjectById(id:String):Subject
+getSubjectByName(name:String):Subject
+addSubject(subject: ArrayList<Subject>): void
<<abstract>> Bid
-subjectId: String
-id:String
-initiatorId:String
-dateCreated: String
-dateClosedDown:String
-additionalInfo: JSON
+getId(): Stirng
+getSubId(): String
+getDateCreated(): String
+getDateClosedDown: String
+getOwnerId(): String
+getAdditional(): String
+getTime(): String
+checkTime(): Boolean
+setDateClosedDown(set:String): void
+getType(): void
Qualification
-id:String
-title:String
-description:String
-verified: bool
-ownerId: String
+getId(id:String): String
+getTitle(): String
+getDescription(): String
+getVerified: Boolean
+getOwnerId: String
BidFactory
+createBid(type:String, id:String, subId:String, ownerId:String, dateCreated:String, dateClosedDown:String, additionalInfo:JSON)
<<abstract>> User
-id:String
-givenName: String
-familyName:String
-userName:String
-initiatedBids:ArrayList<Bidding>
-competency:ArrayList<Comptency>
-qualifications:ArrayList<Qualification>
-contracts: List<Contract>
-messages: List<Message>
+getId():String
+getUserName():String
+getGivenName():String
+getFamilyName():String
+getBids():List<Bidding>
+setBids(List<Bids>):void
+addBid(bid:Bidding):void
+getCompetency():List<Competency>
+setCompetency(List<Competency>):void
+getQualification():List<Competency>
+setQualification(List<Qualification>):void
+getContracts():List<Contract>
+setContracts(List<Contract>):void
+addContracts(con:Contract):void
+getMessages():List<Message>
+setMessages(List<Message>):void
+addMessage(msg:Message):void
Contract
-id:String
-subjectId:String
-ownerId:String
-secondPartyId:String
-dateCreated:String
-dateSigned:String
-expiryDate:String
-paymentInfo: JSON
-lessonInfo: JSON
-additionalInfo: JSON
+getId():String
+getOwnerId():String
+getSecondPartyId():String
+getSubjectId():String
+getDateCreated(): String
+getDateSigned(): String
+getExpiryDate(): String
OpenBid
-time: datetime
+getType(): String
CloseBid
-messages: String
-time: datetime
+getType():String
Competency
-id:String
-level:int
-ownerId:String
-subjectId:String
+getId(): String
+getLevel(): String
+getSubjectId(): String
+getOwnerId(): String
UserFactory
+createUser(isTutor: String, id:String, gName:String, fName:String, uName:String, iBid:String, qualifications: List<Qualification>, competencies: List<Competency>, contracts: List<Contract>, messages: List<Message>): User
Tutor
+getType(): String
Student
+getType(): String
Subject
-id:String
-name:String
-description:String
+getId():String
+getName():String
+getDescription():String
Message
-id:string
-bidId:String
-posterId: String
-datePosted:String
-dateLastEdited:String
-content:String
-initiator: String
-additionalInfo: JSON
+getId(): String
+getBidId: String
+ getPosterId(): String
+getDatePosted: String
+getDateLastEdited(): String
+getContent(): String
+getAdditional(): String
+stringifyContent(): String
+getInitiator(): String
gets instance of
operate on
find offer(s) for
uploads
holds
has
signs
Creates
0..*
0..5
1

RATIONALE

The Common Reuse Principle (CRP): We have attempted to follow this principle to the greatest extent. For instance, in the LOGIC package. All the classes are in one way or another related to each other, mostly through redirects, therefore, if one controller class is being used, all the controller classes have to be used.

SOLID principles

Note: In this assignment, we have attempted to follow all the solid principles. However, we have put a heavier emphasis on the single responsibility and open closed principles for the sake of maintainability of the system.

Single Responsibility Principle (SRP): To abide by the SRP, we have attempted to divide each class into reasonably smaller classes to balance out the complexity of the system and the responsibilities of each class. For instance, in the LOGIC package, we have a class "CheckBidValidity" that is responsible for checking the validity of the bid that is being printed. Although this could have been implemented in the classes that have dependencies with this class, the choice was to make it a separate class as it is a large responsibility that could potentially require to be reused. We have also used abstract classes for users and bids in order to separate the potential difference in functionalities for the different bids. This way, different bid subclasses may have slightly different responsibilities, while the abstract class may be able to handle the common behaviours.

***Open Closed Principle (OCP)*:** The main way we have tried to abide by the open/closed principle is by introducing abstraction into our design. We have several interfaces, such as ViewOffersControllerInterface and/or Observer interface, and through the use of abstract classes. These may hold abstract methods, which in turn, allow us to extend the system with more classes that may have similar functionalities by implementing these interfaces or extending the classes.

Patterns used:

Adapter: The adapter pattern is used to convert JSON data into classes and class data into JSON. Specifically, for this, we have created a JSONAdapter utility class in the API package. This choice was made to ensure that each class follows the single responsibility principle. For instance, without the adapter class, to convert data about the user from REST api into the user class instance, we would need the method to do so in the user class. This breaks the single responsibility principle since the User does not necessarily need to be responsible for converting data.

Factory pattern: In our design we have used 5 factory patterns. For construction of User object, bids, main page of the application, the controllers of the mainpage, and lastly, the ViewBidsControllers. We found the use of factory methods here very helpful in order to support the open/closed principle and make the application more extensible by introducing factory classes where new functionality can be added by allowing extending it to be able to create more concrete objects. Not to mention that it also helps us to support the single responsibility principle (SRP), by allowing us by separating the responsibility of the product creation from the client code into a separate class. Having said that, we at times, have modified the factory methods by removing the concrete factory methods. This is mostly due to the fact that, upon our analysis some of the functionality was unlikely to extend to the point where the single factory class would have too many dependencies to the classes it is responsible for creating. Therefore, it was in our interest to reduce the complexity of the system by removing the concrete factory classes and have one factory class responsible for creating the concrete objects.

Observer: In our design, we have used observer pattern to establish a subscription mechanism between the JSONAdapter that acts as a publisher and Closed bid, bids, messages and contracts that act as a subscriber.[1] This behavioural pattern keeps the subscribers notified at runtime and also makes sure that the JSONAdapter can subscribe subscribers at any time and implement additional update functions without breaking client code with the help of Observer Interface that is implemented by subscribers. For instance, we use this functionality to update the model package class's data during runtime, but listening to method calls that override data in the REST api. This way we know when and what data we need to change in the model. We have found this design pattern very useful to support Open/Closed Principle as you can introduce new subscriber classes without having to change the publisher's code (and vice versa if there's a publisher interface). It also helps us follow a single responsibility principle by having a separate responsibility; Publisher to notify subscribers and invoke update actions when some change is observed and subscriber to call update function.

Singleton: In our design we have used a singleton pattern to make sure that the class with singleton has only one instance and this instance holds the access point globally and keeps the content cached.[2] We have used this singleton pattern for JSONAdapter, SubjectCollection and UserCollection to make sure that it is globally accessed and keep the cache content. This breaks the single responsibility principle since it controls their own creation and life cycle of the instance. However, this is useful to implement observer pattern and some caching as it will remove the possibility of creating more than one instance of these classes and introducing bugs. Hence we had to compensate for this violation as it would in turn make our system more extensible.

MVC: We have also attempted to use the Model View Controller architecture by separating the responsibilities of having a package of classes responsible for holding data (Model) which we have named “APP”, a package responsible for the GUI aspects of the system (View), named “GUI” and a separate package responsible for the interaction between the view and the model packages (Controller), which we have called “LOGIC”. By using this architecture, we have hoped to increase the extensibility of our application by separating the responsibilities of holding data, rendering the UI on multiple page views and performing the logic on UI using the data. This allows us to follow the single responsibility principle and hence would help us extend the system in the future. Furthermore, we began to use the MVC architecture in hopes or removing the cyclic dependency between the View package and the Model package by introducing a Controller package in between that would be responsible for updating the view with the data from the model. However, using python's tkinter we were not able to succeed in that goal, as the GUI page needs to operate on an instance of a controller, which causes cyclic dependency between Controller package and View package.

Reference:

[1]: Unknown Author, Observer Pattern, Retrieved May 02, 2021, from https://refactoring.guru/design-patterns/observer
[2]: Unknown Author, Singleton Pattern, Retrieved May 02, 2021, from https://refactoring.guru/design-patterns/singleton
[3]: “What is MVC? Advantages and Disadvantages of MVC - Interserver Tips,” *Interserver Tips*, 2016. https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/.