Engine

Driver
- UserAPI userAPI
+ main()

View

LoginView
+ LoginView()

UserTypeView
- studentBtn: JButn
- tutorBtn: JButn
+ UserTypeView()

DashboardView
+ DashboardView()
+ setContractsPanel(ArrayList<Contract> contracts)
- getTable(): JTable

BiddingCreationView
+ BiddingCreationView(BiddingCreationModel m)
+ getFields(): JSONObject
- extractSubjects(Names: ArrayList<String>

<<abstract>>
BiddingsView
+ BiddingsView()

OpenCloseBidView
- bidViewType: int
+ OpenCloseBidView(BiddingsModel m, int bidViewType)
+ setPanel(ArrayList<BidOfferModel> bidOffers
- getTable(BidOfferModel b): Jtable

TutorBidOffersView
+ TutorBidOffersView(BiddingModel m)
+ setPanel(ArrayList<Bid> bids)
- getTable(Bid b): JTable

BidOfferView
- type: String
+ BidOfferView(BidOfferModel m, String type)
+ getFields(): JSONObject
+update(Observable o, Object arg):void

CloseBidMsgView
+ CloseBidMsgView()

SignContractView
+ SignContractView()
+ setContent(Contract contract, String type): void

Controller

LoginController
- loginView: LoginView
- loginModel: LoginModel
+ LoginController(LoginView v)
+ getUserAction()
- loginSuccess(String jwtCode)

UserTypeController
- view: UserTypeView
+ UserTypeController(UserTypeView view)
+ userTypeSelected()

DashboardController
- view: DashboardView
- model: DashboardModel
+ DashboardController(DashboardView v, DashboardModel m)
+ updateProfile(): void
+ update(Observable o, Object arg)
+ actionPerformed(ActionEvent e)

BiddingCreationController
- view: BiddingCreationView
- model: BiddingCreationModel
- parentModel: DashboardModel
+ BiddingCreationController(BiddingCreationView v, BiddingCreationModel m, DashboardModel pm)
- getLatestYearFromDOL()
+ postBid()

OpenCloseBidController
- model: BiddingsModel
- view: OpenCloseBidView
+ OpenCloseBidController(OpenCloseBidView v, BiddingsModel m)
- updateBidOffers(int bidViewType)
+ actionPerformed(ActionEvent e)
+ update(Observable o, Object arg)

TutorBidOffersController
- model: BiddingsModel
- view: TutorBidOffersView
+ TutorBidOffersController(TutorBidOffersView v, BiddingsModel m)
- updateBids()
+ actionPerformed(ActionEvent e)
+ update(Observable o, Object arg)

BidOfferController
- model: BidOfferModel
- view: CloseBidMsgView
+ BidOfferController(BidOfferView v, BidOfferModel m)
+ sendOffer():void
+ update(Observable o, Object arg)

CloseBidMsgController
- model: BidOfferModel
- view: CloseBidMsgView
+ CloseBidMsgController(CloseBidMsgView v, BidOfferModel m)
+ actionPerformed(ActionEvent e)

SignContractController
- view: CloseBidMsgView
- contractId : String
+ SignContractController(SignContractView v, String contractId)
+ actionPerformed(ActionEvent e)

java.util

<<interface>>
Observer
+ update(Observable o, Object arg): void

Observable

java.awt.event

java.awt.library is used for action listeners.

javax.swing

javax.swing is used for all the GUI components of the program, all the classes in package View.

The observer pattern for the active mvc architecture used throughout the program is depending on the classes provided in java.util

The observer pattern for the active mvc architecture used throughout the program is depending on the classes provided in java.util

LogicSystem

UserCookie (Singleton)
- userFactory: UserFactory
- ourInstance: UserCookie
- jwtToken: String
- user: User
- userType: int
+ getInstance(): ourInstance
+ init(int userType)

UserFactory
+ IS_STUDENT: int
+ IS_TUTOR: int
+ createUser(String userInfo, int userType): User

Tutor
+ Tutor()
+ getCompetency(List<Subject>)

<<abstract>>
User
- id: String
- givenName: String
- familyName: String
- userName: String
- competencies: ArrayList<Competency>
- qualifications: ArrayList<Qualification>
- contracts: ArrayList<Contract>

Student
- initiatedBid: Bid
+ Student()
+ postBid(String bidType, String subjectId, JSONObject addInfo)

Model

DashboardModel
- contractList: ArrayList<Contract>
- username: String
- userType: String
- fullName: String
+ setProfile(): void
+ checkPromoBid()

BiddingsModel
- bidOffersList: ArrayList<BidOfferModel>
- bids: ArrayList<Bid>
- bid: Bid
+ BiddingsModel()
+ BiddingsModel(String bidId)
+ setBiddings(int biddingsViewType)
- setBids()
- setBidOffersList()
+ selectOffer(BidOfferModel b)

BidOfferModel
- bid: Bid
- bidOfferObj: JSONObject
+ BidOfferModel(Bid b, JSONObject offerObj)
+ BidOfferModel(Bid b)
+ sendOffer(JSONObject jsonObj)

BiddingCreationModel
- bidAPI: BidAPIInterface
+ postBid()

java.awt.event

java.awt.library is used for action listeners.

Contract development

Contract
- id: String
- firstParty: Poster
- secondParty: Poster
- subject: Subject
- dateCreated: Timestamp
- expiryDate: Timestamp
- paymentInfo: JSONObject
- lessonInfo: JSONObject
- additionalInfo: JSONObject
- dateSigned: Timestamp
+ setDateSigned(Timestamp t):void
+ getDayTime(): String

Bidding System

Competency
- id: String
- owner: User
- subject: Subject
- level: int

Message
- id: String
- poster: Poster
- content: String
- JSONObject: additionalInfo
+ tutorUpdateMessageContent(String content)
+ studentReplyMsg(String content)

Subject
- id: String
- name: String
- description: String

<<abstract>>
Bid
- id: String
- type: String
- poster: Poster
- dateCreated: Timestamp
- dateClosedDown: Timestamp
- subject: Subject
- additionalInfo: JSONObject
+ selectBidder(BidOffer offer)
+ close()

<<interface>>
BidInterface
+ selectBidder(BidOfferModel offer)
+ isExpired(): boolean
+ getExpiredDuration(): String
+ close()

BidFactory
+ createBid(JSONObject params): Bid

OpenBid
- bidders: ArrayList<BidOfferModel>
+ OpenBid()
+ isExpired(): boolean
+ getExpiredDuration():String
+ buyout(String tutorId)
+ tutorOfferBid(JSONObject bidOffer)

CloseBid
+ isExpired(): boolean
+ getExpiredDuration():String
+ buyout(String tutorId)
+ tutorOfferBid(JSONObject bidOffer)

API

APIFacade
- ourInstance: APIFacade
- contractAPI: ContractAPIInterface
- bidAPI: BidAPIInterface
- messageAPI: MessageAPIInterface
- userAPI: UserAPIInterface
- subjectAPI: SubjectAPIInterface
- competencyAPI: CompetencyAPIInterface
- qualificationAPI: QualificationAPIInterface
+ getInstance(): ourInstance
+ getUserCompetenciesByBid: ArrayList<Competency>
+ getUserQualificationsByBid: ArrayList<Qualification>
+ getUserInitiatedBidsByBid(String id): Bid
+ getAllQualifications(): ArrayList<Qualification>
+ getAllBids(): ArrayList<Bid>
+ getBidByBid(String id): Bid
+ createBid(String type, String id, String subjectId, JSONObject additionalInfo): Bid
+ updateBidBid(String id, JSONObject requestBody)
+ closeDownBidBid(String id)
+ getMessageById(String id): Message
+ createMessage(String bidId, String posterId, JSONObject requestBody)
+ signContractById(String id)
+ creatContract(String studentId, String tutorId, String subjectId, Timestamp expiry, JSONObject paymentInfo, JSONObject lessonInfo, JSONObject additionalInfo): Contract
+ getContractsByUserId(String userId): ArrayList<Contract>

APIService
+GETRequest(String route)
+ UpdateRequest(String url, StringBuilder params, int requestType)

APIAdapters

<<abstract>>
APIRouter
- route: String
- c: Class
- objMapper: ObjectMapper
+ getAll(): Object
+ create(StringBuilder params): Bid
+ getById(String id, String queryParam): Bid
+ deleteById(String id)
+ updatePartialById(String id, StringBuilder params): Bid

UserAPI
+ UserAPI()
+ getAll(): JSONArray
+ getById(): JSONObject
+ userLogin(String username, String password): JSONObject
+ userVerifyLogin(String jwt): JSONObject

BidAPI
+ BidAPI()
+ getAll(): ArrayList<Bid>
+ getById(String id, String queryParam): Bid
+ create(StringBuilder params): Bid
+ updatePartialById(String id, StringBuilder params): Bid
+ parseToJsonForCreate(String type, String initiatorId, String subjectId, JSONObject additionalInfo): StringBuilder
+ parseToJsonForPartialUpdate(JSONObject additionalInfo): StringBuilder
+ closeDownBidById(String id): boolean

MessageAPI
+ MessageAPI()
+ getAll(): ArrayList<Message>
+ parseToJsonForCreate(String bidId, String posterId, String content): StringBuilder
+ parseToJsonForPartialUpdate(String content, JSONObject additionalInfo): StringBuilder

ContractAPI
+ ContractAPI()
+ getAll(): ArrayList<Contract>
+ parseToJson(String firstPartyId, String secondPartyId, String subjectId, Timestamp expiryDate, JSONObject paymentInfo, JSONObject lessonInfo, JSONObject additionalInfo)
+ signContract(String id): boolean

ClassInterfaces

<<interface>>
BaseAPIInterface
+ getAll()
+ create(StringBuilder params)
+ getById(String id)
+ deleteById(String id)
+ updatePartialById(String id, StringBuilder params)

<<interface>>
UserAPIInterface
+ userLogin(String username, String password): JSONObject
+ userVerifyLogin(String jwt): JSONObject

<<interface>>
BidAPIInterface
+ parseToJsonForCreate(String type, String initiatorId, String subjectId, JSONObject additionalInfo): StringBuilder
+ parseToJsonForPartialUpdate(JSONObject additionalInfo): StringBuilder
+ closeDownBidById(String id): boolean

<<interface>>
MessageAPIInterface
+ parseToJsonForCreate(String bidId, String posterId, String content): StringBuilder
+ parseToJsonForPartialUpdate(String content, JSONObject additionalInfo): StringBuilder

<<interface>>
ContractAPIInterface
+ parseToJson(String firstPartyId, String secondPartyId, String subjectId, Timestamp expiryDate, JSONObject paymentInfo, JSONObject lessonInfo, JSONObject additionalInfo)
+ signContract(String id): boolean

A library that maps JSONObject to java class objects.

com.fasterxml.jackson

org.apache.http is a library in maven that handles the HTTP requests of the program

org.apache.http

Adapter Pattern
APIRouter is the adapter and APIService is the adapter. The concrete classes inherting the APIRouter are the variant concrete class which implements it's own Client Interface contains some methods that only belongs to it's respective concrete class which also extends from the BaseAPIInterface that contains all the common methods needed.

The **Adapter pattern** is used in handling the API requests throughout the program implemented in the APIAdapters and ClientInterfaces package. Here the concrete classes inherited the APIRouter abstract class are the adapters and the APIService class is the adaptee. The interfaces in ClientInterface were introduced adhering to each concrete adapter class to provide a clear protocol for the client to use the adapter. This pattern used is to ensure that the adapters are getting raw data from the client and passing the correct data structure to the adapter. Furthermore, this pattern allows new APIs requests to be added easily by just introducing a new adapter class extending from the APIRouter without changing anything from the APIService which upholds the **Open/Close Principle**. The downside of this pattern is that every new adapter introduced needs to introduce a new interface in order to adhere to the adapter pattern rule.

The **Facade pattern** is used in the API package. This is to provide convenient access for the client to make API calls and also isolates the systems from the complexity of the API package (subsystem). Client classes only have to call the appropriate function in the APIFacade to make an API request. With this pattern, clients do not need to initialize all the APIadapters, keep track of dependencies and execute methods in the correct order, as all these will be provided in the functions of the Facade class. Therefore, it also greatly increased the readability of the code outside of the API subsystem. [1] Moreover, the APIFacade class is also transformed into a **Singleton** since a single APIFacade object is sufficient across the system. However, this violates the Single Responsibility Principle as it contains a large number of function calls which interact with all concrete classes of the APIAdapter package.

The **Factory pattern** is implemented in the UserFactory class and the BidFactory class. These factory classes are used to create variants of User and Bid objects where the client only has to pass in a param and the factory class will return the client the appropriate subclass. This is to ensure that we avoid tight coupling between the UserFactory/BidFactory and the Student/Tutor or OpenBid/CloseBid classes. Hence, these classes also uphold the Single Responsibility Principle as product creation code is extracted into one place in the program which also increases the readability of our code.
Moreover, it also provides the convenience of **Open/Close Principle**, where we can easily introduce new variants of the product into the program without breaking the existing code.

Finally, we implemented the **Observer pattern** for the **active MVC architecture** used throughout the program which depends on the classes provided in java.util. We have decided to implement this as the View classes need to handle data changes in the Model class dynamically. Therefore, this pattern is enforced so that whenever a user presses the refresh button, the Controller will listen to it and inform the Model (observable) a change has happened and handle the data update, the Controller observing it will trigger its update() functions and updates the View's content. This pattern solves the problem of hectic and abundance of function calls from Model to Controller and finally to View.

The **Common Closure Principle** is used in the BiddingSystem and API packages, where all the classes related to the bidding functionality and all the classes related to API requests are grouped together accordingly. For instance, when a Bid object is changed, all the affected classes have to be changed and only these classes have to be changed, same applies to the API's classes. This way it helps in increasing maintainability where developers change something in the package, they know which classes have to be changed accordingly that are within the package.

Resources:
[1] Relations with Other Patterns https://refactoring.guru/design-patterns/facade