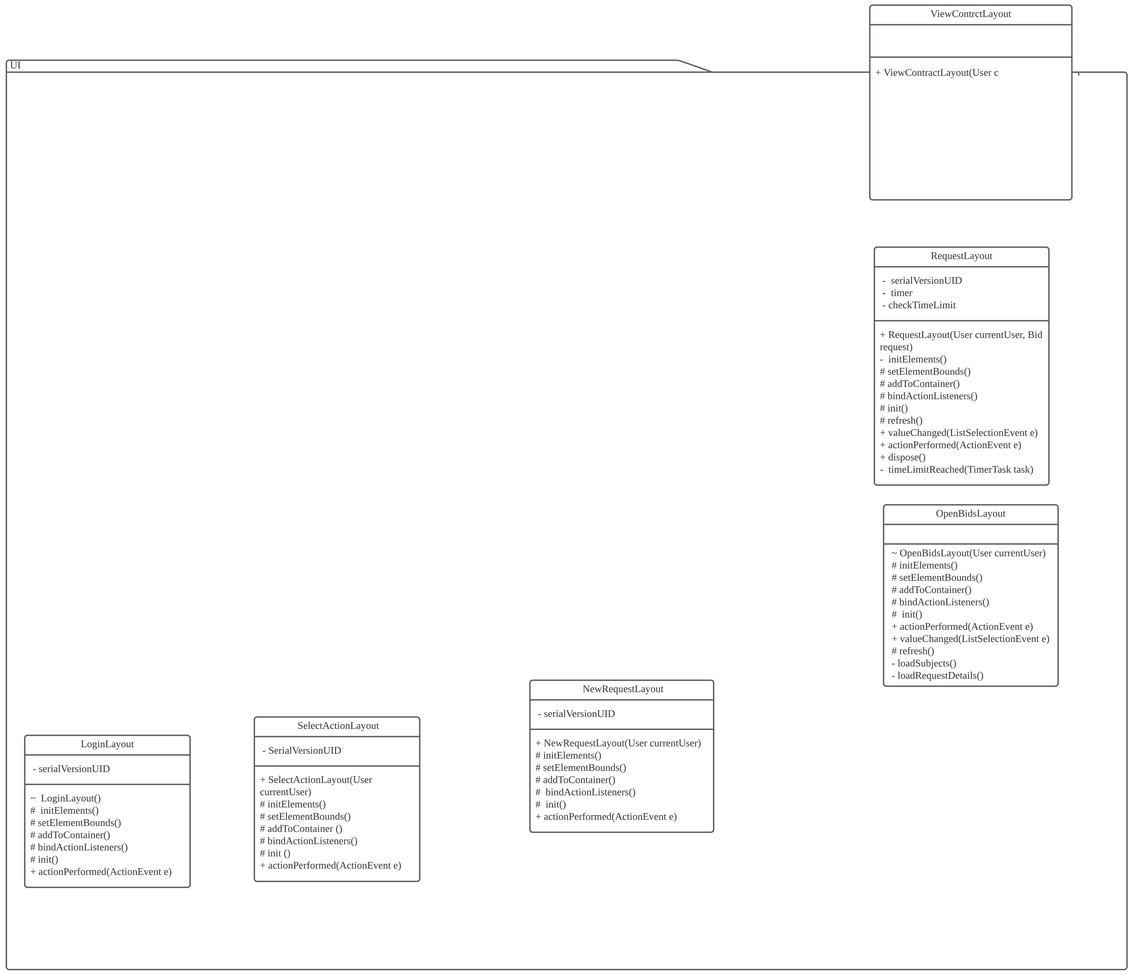
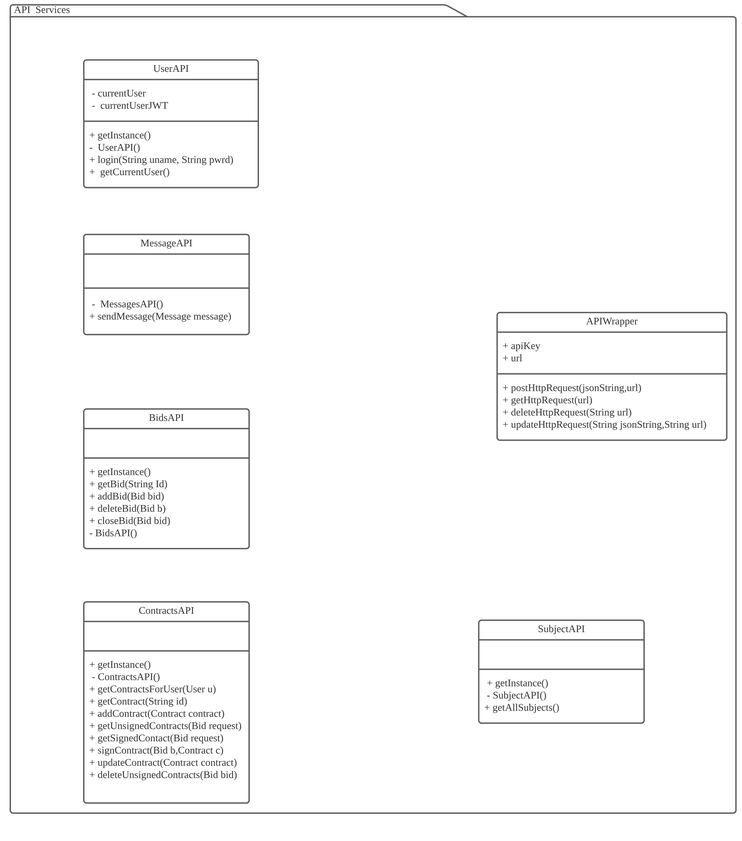
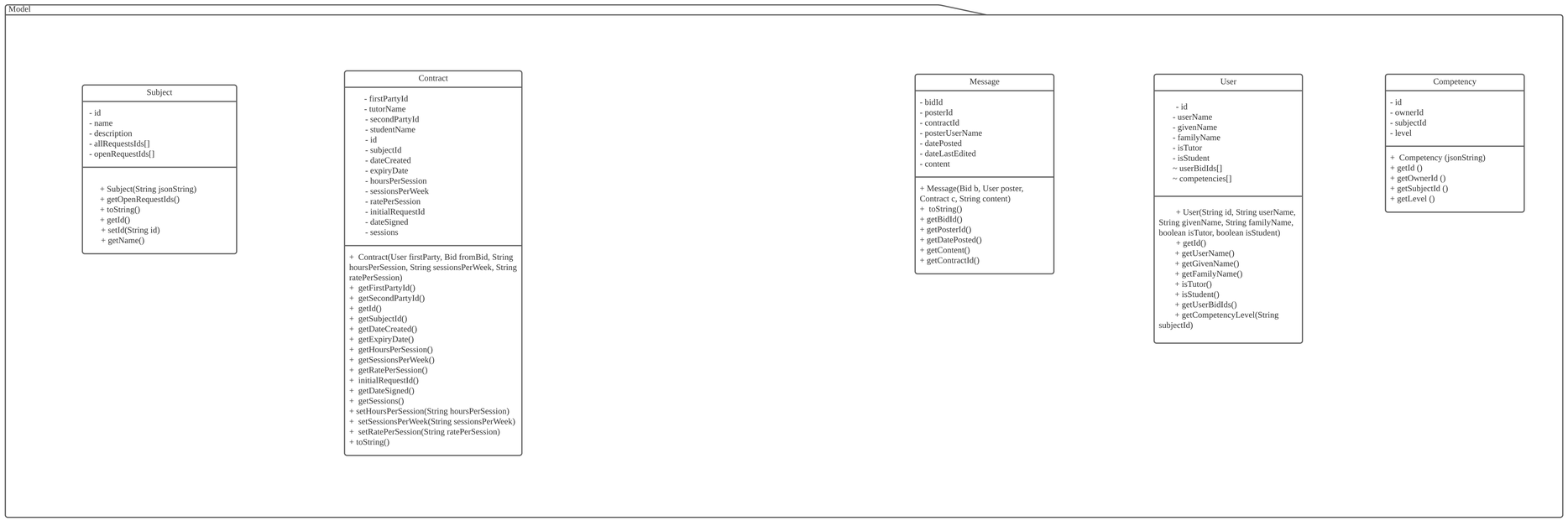

UI
ViewContrctLayout
+ ViewContractLayout(User c
RequestLayout
- serialVersionUID
- timer
- checkTimeLimit
+ RequestLayout(User currentUser, Bid request)
- initElements()
setElementBounds()
addToContainer()
bindActionListeners()
init()
refresh()
+ valueChanged(ListSelectionEvent e)
+ actionPerformed(ActionEvent e)
+ dispose()
- timeLimitReached(TimerTask task)
OpenBidsLayout
~ OpenBidsLayout(User currentUser)
# initElements()
# setElementBounds()
# addToContainer()
# bindActionListeners()
# init()
+ actionPerformed(ActionEvent e)
# refresh()
- loadSubjects()
- loadRequestDetails()
NewRequestLayout
- serialVersionUID
+ NewRequestLayout(User currentUser)
# initElements()
# setElementBounds()
# addToContainer()
# bindActionListeners()
# init()
+ actionPerformed(ActionEvent e)
SelectActionLayout
- SerialVersionUID
+ SelectActionLayout(User currentUser)
# initElements()
# setElementBounds()
# addToContainer ()
# bindActionListeners()
# init ()
+ actionPerformed(ActionEvent e)
LoginLayout
- serialVersionUID
~ LoginLayout()
# initElements()
# setElementBounds()
# addToContainer()
# bindActionListeners()
# init()
+ actionPerformed(ActionEvent e)


API Services
UserAPI
- currentUser
- currentUserJWT
+ getInstance()
- UserAPI()
+ login(String uname, String pwrd)
+ getCurrentUser()
MessageAPI
- MessagesAPI()
+ sendMessage(Message message)
APIWrapper
+ apiKey
+ url
+ postHttpRequest(jsonString,url)
+ getHttpRequest(url)
+ deleteHttpRequest(String url)
+ updateHttpRequest(String jsonString,String url)
BidsAPI
+ getInstance()
+ getBid(String Id)
+ addBid(Bid bid)
+ deleteBid(Bid b)
+ closeBid(Bid bid)
- BidsAPI()
ContractsAPI
+ getInstance()
- ContractsAPI()
+ getContractsForUser(User u)
+ getContract(String id)
+ addContract(Contract contract)
+ getUnsignedContracts(Bid request)
+ getSignedContact(Bid request)
+ signContract(Bid b,Contract c)
+ updateContract(Contract contract)
+ deleteUnsignedContracts(Bid bid)
SubjectAPI
+ getInstance()
- SubjectAPI()
+ getAllSubjects()


Model
Subject
- id
- name
- description
- allRequestsIds[]
- openRequestIds[]
+ Subject(String jsonString)
+ getOpenRequestIds()
+ toString()
+ getId()
+ setId(String id)
+ getName()
Contract
- firstPartyId
- tutorName
- secondPartyId
- studentName
- id
- subjectId
- dateCreated
- expiryDate
- hoursPerSession
- sessionsPerWeek
- ratePerSession
- initialRequestId
- dateSigned
- sessions
+ Contract(User firstParty, Bid fromBid, String hoursPerSession, String sessionsPerWeek, String ratePerSession)
+ getFirstPartyId()
+ getSecondPartyId()
+ getId()
+ getSubjectId()
+ getDateCreated()
+ getExpiryDate()
+ getHoursPerSession()
+ getSessionsPerWeek()
+ getRatePerSession()
+ initialRequestId()
+ getDateSigned()
+ getSessions()
+ setHoursPerSession(String hoursPerSession)
+ setSessionsPerWeek(String sessionsPerWeek)
+ setRatePerSession(String ratePerSession)
+ toString()
Message
- bidId
- posterId
- contractId
- posterUserName
- datePosted
- dateLastEdited
- content
+ Message(Bid b, User poster, Contract c, String content)
+ toString()
+ getBidId()
+ getPosterId()
+ getDatePosted()
+ getContent()
+ getContractId()
User
- id
- userName
- givenName
- familyName
- isTutor
- isStudent
~ userBidIds[]
~ competencies[]
+ User(String id, String userName, String givenName, String familyName, boolean isTutor, boolean isStudent)
+ getId()
+ getUserName()
+ getGivenName()
+ getFamilyName()
+ isTutor()
+ isStudent()
+ getUserBidIds()
+ getCompetencyLevel(String subjectId)
Competency
- id
- ownerId
- subjectId
- level
+ Competency (jsonString)
+ getId ()
+ getOwnerId ()
+ getSubjectId ()
+ getLevel ()

Design Rationale

Group Name: AndrewAndBassel

There are several class design principles that are adhered to in this project. The most apparent of which is the Liskov Substitution Principle (LSP). This is especially apparent in the classes of the UI & Services Package. All windows are inherited from WindowLayout and are forced to perform the 4 mandatory methods upon initialization, streamlining the window creation process therefore ensuring that its subclasses can be interchangeably in place with other subclasses whenever we need to switch between windows in the application path controller (in the future).

In the Model package, the classes are designed to strictly enforce the Single Responsibility Principle (SRP), as each class represents a single entity and therefore only changes whenever the entity's attribute needs to change. In the UI and API Services packages however, the SRP principle is violated, UI classes have several responsibilities to deal with given by the number of different user inputs.

The classes in the API Services package however, follow the SIngleton Design Pattern. This decision was made because one and only one instance of the API Class is needed for each entity (UserAPI, SubjectAPI, etc.) Singleton design enforces the fact that the API can only be accessed by the main program in a controlled manner as dictated by the methods in the API services class. This is done to ensure the server's data integrity. It makes the overall code neater and maintainable as it only provides one global interface that acts as an access point to the API. Any changes to the API architecture will only require changes to the classes in the API services package, abstracting the API connections code from the main program. The disadvantage is that SRP will be violated as the API Service classes have more than one responsibility (eg. connect to API, parse json to java object, enforcing server rules, etc.).

On a package-wide scope, packages in the code design adhere to the Common Closure Principle (CCP). They are highly cohesive as classes are grouped together based on their major purposes. They are sensitive to the same type of changes in the external environment. For example, the API Service package is sensitive to the changes in the API standards, the UI package is sensitive to changes in the Java Swing Library and the Model package is sensitive to real world logic. The downside of having strict CCP adherence is that it makes each package very large, demonstrated by the UI class. However, this is still manageable for this project.

Enforcing CCP can reduce some, but not all coupling and dependencies between packages. In the class diagram, there are a lot of dependencies between classes of different packages. However, the bigger picture shows that package design strictly follows the Acyclic Dependency Principle (ADP) where there are no cyclic dependencies between classes of different packages. This is done to increase maintainability as we avoid making changes in a negative feedback loop.

Another package coupling pattern that is adhered to is the Stable Dependency Principle (SDP). Observing from tracing the dependency arrow flow, it is apparent that the dependencies typically end at the Model package. This is intentionally done because classes in the model package mirror the Real World Logic which is typically very stable. Therefore it is assumed that the model package is the most stable package of all. While the UI package is typically very unstable as it is dependent on the client's requirements (which usually changes very frequently). therefore it is deemed as the most unstable package in the system.

Although vague, it can be observed that the overall architecture of the system loosely adheres to the Model-View-Controller (MVC) architecture. There is a stable model package holding the classes that mirror real world objects, an unstable view package (UI package) that holds the UI related classes, and a controller package (API services) which dictates how the views interact with the model.