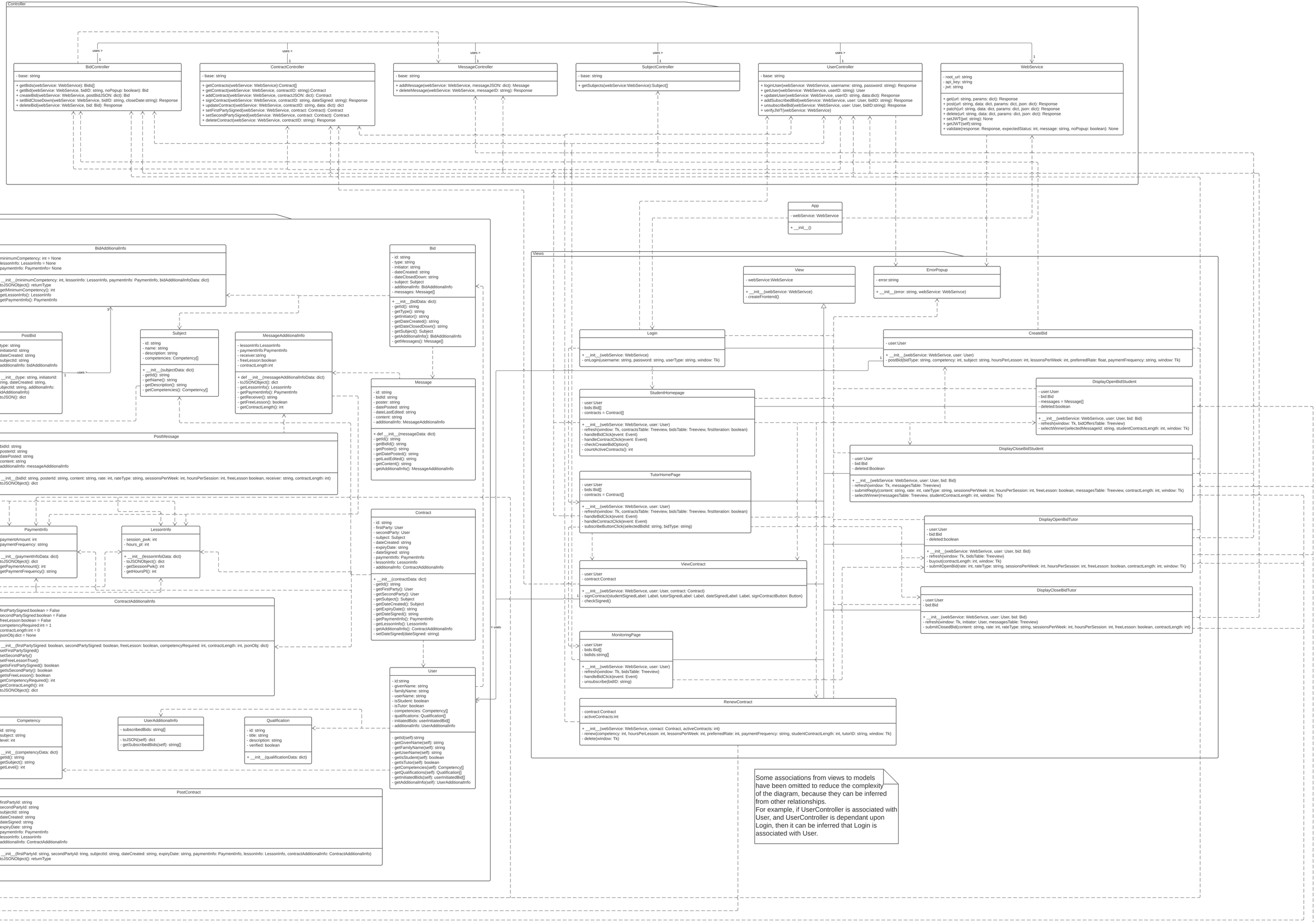
Controller
uses >
1
BidController
- base: string
+ getBids(webService: WebService): Bids[]
+ getBid(webService: WebService, bidID: string, noPopup: boolean): Bid
+ createBid(webService: WebService, postBidJSON: dict): Bid
+ setBidCloseDown(webService: WebService, bidID: string, closeDate:string): Response
+ deleteBid(webService: WebService, bid: Bid): Response
ContractController
- base: string
+ getContracts(webService: WebService):Contract[]
+ getContract(webService: WebService, contractID: string):Contract
+ addContract(webService: WebService, contractJSON: dict): Contract
+ signContract(webService: WebService, contractID: string, dateSigned: string): Response
+ updateContract(webService: WebService, contractID: string, data: dict): dict
+ setFirstPartySigned(webService: WebService, contract: Contract): Contract
+ setSecondPartySigned(webService: WebService, contract: Contract): Contract
+ deleteContract(webService: WebService, contractID: string): Response
MessageController
- base: string
+ addMessage(webService: WebService, messageJSON: dict): Message
+ deleteMessage(webService: WebService, messageID: string): Response
SubjectController
- base: string
+ getSubjects(webService:WebService):Subject[]
UserController
- base: string
+ loginUser(webService: WebService, username: string, password: string): Response
+ getUser(webService: WebService, userID: string): User
+ updateUser(webService: WebService, userID: string, data:dict): Response
+ addSubscribeeBid(webService: WebService, user: User, bidID: string): Response
+ unsubscribeBid(webService: WebService, user: User, bidID:string): Response
+ verifyJWT(webService: WebService)
WebService
- root_url: string
- api_key: string
- jwt: string
+ get(url: string, params: dict): Response
+ post(url: string, data: dict, params: dict, json: dict): Response
+ patch(url: string, data: dict, params: dict, json: dict): Response
+ delete(url: string, data: dict, params: dict, json: dict): Response
+ setJWT(jwt: string): None
+ getJWT(self):string
+ validate(response: Response, expectedStatus: int, message: string, noPopup: boolean): None
App
- webService: WebService
+ __init__()
models
BidAdditionalInfo
- minimumCompetency: int = None
- lessonInfo: LessonInfo = None
- paymentInfo: PaymentInfo= None
+ __init__(minimumCompetency: int, lessonInfo: LessonInfo, paymentInfo: PaymentInfo, bidAdditionalInfoData: dict)
- toJSONObject(): returnType
- getMinimumCompetency(): int
- getLessonInfo(): LessonInfo
- getPaymentInfo(): PaymentInfo
Bid
- id: string
- type: string
- initiator: string
- dateCreated: string
- dateClosedDown: string
- subject: Subject
- additionalInfo: BidAdditionalInfo
- messages: Message[]
+ __init__(bidData: dict):
- getId(): string
- getType(): string
- getInitiator(): string
- getDateCreated(): string
- getDateClosedDown(): string
- getSubject(): Subject
- getAdditionalInfo(): BidAdditionalInfo
- getMessages(): Message[]
PostBid
- type: string
- initiatorId: string
- dateCreated: string
- subjectId: string
- additionalInfo: bidAdditionalInfo
+ __init__(type: string, initiatorId: string, dateCreated: string, subjectId: string, additionalInfo: bidAdditionalInfo)
- toJSON(): dict
uses >
1
Subject
- id: string
- name: string
- description: string
- competencies: Competency[]
+ __init__(subjectData: dict)
- getId(): string
- getName(): string
- getDescription(): string
- getCompetencies(): Competency[]
MessageAdditionalInfo
- lessonInfo:LessonInfo
- paymentInfo:PaymentInfo
- receiver:string
- freeLesson:boolean
- contractLength:int
+ def __init__(messageAdditionalInfoData: dict)
- toJSONObject(): dict
- getLessonInfo(): LessonInfo
- getPaymentInfo(): PaymentInfo
- getReceiver(): string
- getFreeLesson(): boolean
- getContractLength(): int
Message
- id: string
- bidId: string
- poster: string
- datePosted: string
- dateLastEdited: string
- content: string
- additionalInfo: MessageAdditionalInfo
+ def __init__(messageData: dict)
- getId(): string
- getBidId(): string
- getPoster(): string
- getDatePosted(): string
- getLastEdited(): string
- getContent(): string
- getAdditionalInfo(): MessageAdditionalInfo
PostMessage
- bidId: string
- posterId: string
- datePosted: string
- content: string
- additionalInfo: messageAdditionalInfo
+ __init__(bidId: string, posterId: string, content: string, rate: int, rateType: string, sessionsPerWeek: int, hoursPerSession: int, freeLesson boolean, receiver: string, contractLength: int)
- toJSONObject(): dict
PaymentInfo
- paymentAmount: int
- paymentFrequency: string
+ __init__(paymentInfoData: dict)
- toJSONObject(): dict
- getPaymentAmount(): int
- getPaymentFrequency(): string
LessonInfo
- session_pwk: int
- hours_pl: int
+ __init__(lessonInfoData: dict)
- toJSONObject(): dict
- getSessionPwk(): int
- getHoursPl(): int
Contract
- id: string
- firstParty: User
- secondParty: User
- subject: Subject
- dateCreated: string
- expiryDate: string
- dateSigned: string
- paymentInfo: PaymentInfo
- lessonInfo: LessonInfo
- additionalInfo: ContractAdditionalInfo
+ __init__(contractData: dict)
- getId(): string
- getFirstParty(): User
- getSecondParty(): User
- getSubject(): Subject
- getDateCreated(): Subject
- getExpiryDate(): string
- getDateSigned(): string
- getPaymentInfo(): PaymentInfo
- getLessonInfo(): LessonInfo
- getAdditionalInfo(): ContractAdditionalInfo
- setDateSigned(dateSigned: string)
ContractAdditionalInfo
- firstPartySigned:boolean = False
- secondPartySigned:boolean = False
- freeLesson:boolean = False
- competencyRequired:int = 1
- contractLength:int = 0
- jsonObj:dict = None
+ __init__(firstPartySigned: boolean, secondPartySigned: boolean, freeLesson: boolean, competencyRequired: int, contractLength: int, jsonObj: dict)
- setFirstPartySigned()
- setSecondParty()
- setFreeLessonTrue()
- getIsFirstPartySigned(): boolean
- getIsSecondParty(): boolean
- getIsFreeLesson(): boolean
- getCompetencyRequired(): int
- getContractLength(): int
- toJSONObject(): dict
User
- id:string
- givenName: string
- familyName: string
- userName: string
- isStudent: boolean
- isTutor: boolean
- competencies: Competency[]
- qualifications: Qualification[]
- initiatedBids: userInitiatedBid[]
- additionalInfo: UserAdditionalInfo
- getId(self):string
- getGivenName(self): string
- getFamilyName(self): string
- getUserName(self): string
- getIsStudent(self): boolean
- getIsTutor(self): boolean
- getCompetencies(self): Competency[]
- getQualifications(self): Qualification[]
- getInitiatedBids(self): userInitiatedBid[]
- getAdditionalInfo(self): UserAdditionalInfo
Competency
- id: string
- subject: string
- level: int
+ __init__(competencyData: dict)
- getId(): string
- getSubject(): string
- getLevel(): int
UserAdditionalInfo
- subscribedBids: string[]
- toJSON(self): dict
- getSubscribedBids(self): string[]
Qualification
- id: string
- title: string
- description: string
- verified: boolean
+ __init__(qualificationData: dict)
PostContract
- firstPartyId: string
- secondPartyId: string
- subjectId: string
- dateCreated: string
- dateSigned: string
- expiryDate: string
- paymentInfo: PaymentInfo
- lessonInfo: LessonInfo
- additionalInfo: ContractAdditionalInfo
+ __init__(firstPartyId: string, secondPartyId: tring, subjectId: string, dateCreated: string, expiryDate: string, paymentInfo: PaymentInfo, lessonInfo: LessonInfo, contractAdditionalInfo: ContractAdditionalInfo)
- toJSONObject(): returnType
< uses
Views
View
- webService:WebService
+ __init__(webService: WebService)
- createFrontend()
ErrorPopup
- error:string
+ __init__(error: string, webService: WebService)
Login
+ __init__(webService: WebService)
- onLogin(username: string, password: string, userType: string, window: Tk)
CreateBid
- user:User
+ __init__(webService: WebSerivce, user: User)
- postBid(bidType: string, competency: int, subject: string, hoursPerLesson: int, lessonsPerWeek: int, preferredRate: float, paymentFrequency: string, window: Tk)
1
StudentHomepage
- user:User
- bids:Bid[]
- contracts = Contract[]
+ __init__(webService: WebSerivce, user: User)
- refresh(window: Tk, contractsTable: Treeview, bidsTable: Treeview, firstIteration: boolean)
- handleBidClick(event: Event)
- handleContractClick(event: Event)
- checkCreateBidOption()
- countActiveContracts(): int
DisplayOpenBidStudent
- user:User
- bid:Bid
- messages = Message[]
- deleted:boolean
+ __init__(webService: WebSerivce, user: User, bid: Bid)
- refresh(window: Tk, bidOffersTable: Treeview)
- selectWinner(selectedMessageId: string, studentContractLength: int, window: Tk)
DisplayCloseBidStudent
- user:User
- bid:Bid
- deleted:Boolean
+ __init__(webService: WebSerivce, user: User, bid: Bid)
- refresh(window: Tk, messagesTable: Treeview)
- submitReply(content: string, rate: int, rateType: string, sessionsPerWeek: int, hoursPerSession: int, freeLesson: boolean, messagesTable: Treeview, contractLength: int, window: Tk)
- selectWinner(messagesTable: Treeview, studentContractLength: int, window: Tk)
TutorHomePage
- user:User
- bids:Bid[]
- contracts = Contract[]
+ __init__(webService: WebSerivce, user: User)
- refresh(window: Tk, contractsTable: Treeview, bidsTable: Treeview, firstIteration: boolean)
- handleBidClick(event: Event)
- handleContractClick(event: Event)
- subscribeButtonClick(selectedBidId: string, bidType: string)
DisplayOpenBidTutor
- user:User
- bid:Bid
- deleted:boolean
+ __init__(webService: WebSerivce, user: User, bid: Bid)
- refresh(window: Tk, bidsTable: Treeview)
- buyout(contractLength: int, window: Tk)
- submitOpenBid(rate: int, rateType: string, sessionsPerWeek: int, hoursPerSession: int, freeLesson: boolean, contractLength: int, window: Tk)
DisplayCloseBidTutor
- user:User
- bid:Bid
+ __init__(webService: WebSerivce, user: User, bid: Bid)
- refresh(window: Tk, initator: User, messagesTable: Treeview)
- submitClosedBid(content: string, rate: int, rateType: string, sessionsPerWeek: int, hoursPerSession: int, freeLesson: boolean, contractLength: int)
ViewContract
- user:User
- contract:Contract
+ __init__(webService: WebSerivce, user: User, contract: Contract)
- signContract(studentSignedLabel: Label, tutorSignedLabel: Label, dateSignedLabel: Label, signContractButton: Button)
- checkSigned()
1
MonitoringPage
- user:User
- bids:Bid[]
- bidIds:string[]
+ __init__(webService: WebSerivce, user: User)
- refresh(window: Tk, bidsTable: Treeview)
- handleBidClick(event: Event)
- unsubscribe(bidID: string)
RenewContract
- contract:Contract
- activeContracts:int
+ __init__(webService: WebSerivce, conract: Contract, activeContracts: int)
- renew(competency: int, hoursPerLesson: int, lessonsPerWeek: int, preferredRate: int, paymentFrequency: string, studentContractLength: int, tutorID: string, window: Tk)
- delete(window: Tk)
Some associations from views to models have been omitted to reduce the complexity of the diagram, because they can be inferred from other relationships.
For example, if UserController is associated with User, and UserController is dependant upon Login, then it can be inferred that Login is associated with User.

Design Rationale

Refactoring

Many codesmells were identified at the beginning of assignment 3 that were required to be removed through the implementation of refactoring techniques. The first codesmell that was identified was bloaters. All classes within the views package contained a createFrontend method which was used to create the GUI that users would be able to see and interact with. This method was very extensive, difficult to work with and contained too many lines of code making it difficult to understand and extend if necessary.

As this was defined to be a problem of a long method, it was decided to use the extract method refactoring technique. A portion of the createFrontend method was extracted and placed into a new method called refresh, which would be used to periodically refresh and populate the information that would be displayed on the frontend. The benefits of this refactoring were that the code in createFrontend became easier to read and it would result in less duplication of code as populating the tables would happen in only one method.

Another bloater codesmell identified was large classes. Many classes within the views package were very long and contained many lines of code. It was further discovered that these classes also contained change preventers codesmells in the form of divergent change and shotgun surgery as they contained a lot of copy and pasted code. It was identified that most of the duplicated code present was relating to API calls.

In order to remove these two codesmells, the extract class refactoring technique was adopted whilst creating the controllers package. This allowed us to refactor the API calls into their respective controller classes which contained static methods. This allowed us to replace many lines of code to reduce our large classes and eliminate the change preventers codesmells as there was no longer any need to change code in multiple locations if the API calls needed to be modified. Another benefit to this refactoring is the single responsibility principle has been reinforced as all functionality relating to API calls was placed in one central class that could be accessed wherever required.

Design Extensions

In order to complete the new requirements as part of assignment 3, many modifications and extensions were made to our codebase.

As part of requirement 4, we overhauled our codebase to use the MVC (Model View Controller) architecture. This greatly improved the quality and design of our code, and made implementing the other new requirements much easier.

To implement requirement one, a new UserAdditionalInfo class was created, which would store an array of subscribed bid IDs. The User class had to be modified to account for this,

and a new MonitoringPage class was created to display the required output. Further to this, a new automatic refresh method was implemented for the MonitoringPage, so the output could be updated every N seconds. This new automatic refresh system was implemented in many places throughout the code to improve on the manual method previously used in places like StudentHomePage.

Requirement 2 was implemented by adding a contractLength field into MessageAdditionalInfo. This allowed both students and tutors to communicate their preferred contract length with each other. As part of this, DisplayOpenBidTutor, DisplayOpenBidStudent, and DisplayClosedBidStudent were updated to allow users to provide this contractLength input. Furthermore, an automatic check was added to the system that will display a popup if a contract is expiring within one month, upon login.

Requirement 3 was implemented by adding extra fields into ContractAdditionalInfo (competencyRequired and contractLength). By storing these fields in the appropriate model, we were able to easily display and use them throughout the application, such as in the ViewContract class. Furthermore, a RenewContract class was created to facilitate the bulk of the functionality for this requirement. Our code previously automatically signed the contract when the Student would select a winner, or the tutor would buyout a bid. However this was changed, and now both parties must explicitly sign the contract after it is created. The homepages now also check if a contract is expired or not, and provide different functionality in that case.

Design & Package Principles

Many object-oriented principles and ideologies were followed when creating the Tutoring System to ensure maintainability and extensibility of the project.

The Don’t Repeat Yourself principle was also considered throughout the project and applied by creating common classes and functions to be reused throughout the application. Controller classes were created to store functions that are needed in many places throughout the application. For example, UserController has a getUser method. Getting a specific user is an important piece of functionality that was previously copy/pasted in many places throughout the codebase. Having a controller method for this keeps the functionality in one place, and allows us to easily extend/maintain it in the future, to stay in line with API changes. Previously, a change in the API endpoints would require changes through many places in the code. However, now only the controller classes would need to be updated. Previously, we thought the WebService class itself was enough for the DRY principle. However, by taking this one step further, we have greatly improved maintainability and extensibility, as well as reduced repeated code.

During our project, we considered the Single Responsibility Principle (SRP) to ensure our classes have only one purpose. A good example of this is again our controller classes. Previously the classes within our views package were doing far too much, which resulted in large and confusing classes. The views classes were both rendering the frontend, as well as performing business logic. The controllers classes now extract this responsibility, by performing the business logic instead. This allows our views to have a single responsibility,

which is to display the frontend UI, while the controllers can manage the business logic. As another example, a new RenewContract class was created, rather than adding to the ViewContract class. This is because both classes can have a clearly distinct purpose (renewing vs viewing contracts), and separating the two helps prevent God-classes within our system and ensures our classes are cohesive. This ultimately makes our code more manageable and extensible.

The Open/Closed Principle (OCP) was an important principle considered to ensure that classes are easy to extend and add new functionality whilst ensuring they are still able to be used in the same manner. This means that when our classes are changed to include new parameters or methods, there is no need for changes in code where these classes are currently being used. Examples of this principle being applied within our project is the ContractAdditionalInfo class which was required to be extended to include contractLength and competencyRequired as new attributes. However, the Contract class, which depends on ContractAdditionalInfo was not required to be modified to store contract information or whilst returning a contract instance. Another example of this principle being applied includes the User class which was required to be extended to include UserAdditionalInfo as a new attribute. However, any of the other classes that were associated or depended on the User class were not required to be modified as a result of the changes in the User class.

We also implemented the Liskov Substitution Principle in our code. Previously, all of our frontend views had much repeated code and attributes. They all took in WebService as parameter, and had a createFrontend function. By creating a generic View class, and using that as a superclass for the other views, we simplified our codebase. Any view can now be substituted in a place where the code is attempting to create a frontend view, and it will work without breaking the application. Further to reducing repeated code, implementing LSP makes our code more robust, as views can easily be substituted and interchanged anywhere a parent View object is expected. This improves the extensibility of our code as well, as new views can easily be added. The subclasses override the createFrontend method, however they take in the same input parameters, ensuring they follow LSP.

At a package level, one of the design principles implemented is the Acyclic Dependency Principle. This principle aims to ensure there are no cyclic dependencies within packages. Refactoring the project to incorporate MVC architectural design ensures there are no dependencies between the models, views and controllers packages.

Furthermore, another package level design principle incorporated is Release Reuse Equivalence Principle. The controllers package primarily helps to achieve this design principle as all controller classes have been packaged together. The controller classes are solely responsible for all API calls and relevant methods are able to be reused wherever they are required. The key advantage of this approach is that if there are any changes to the API or if new versions of the API are released, then these changes are required to be implemented only within the controllers package.

Design Patterns

The Singleton design pattern was implemented in our codebase in the WebService class. Only one instance of WebService had to be used throughout the application, as it stores the

JSON Web Token for authentication. If only one instance was not used, the user would have to login multiple times so that each instance could store a JWT. Using the singleton pattern made our application more user friendly and secure, as the users login details would not be transmitted multiple times. Only one instance of WebService is created, and that same instance is used throughout the code by each Controller.

Architecture Pattern

Originally our application did not follow any specific architecture pattern. Our frontend views also contained business logic, which interacted with models. However, implementing the Model View Controller (MVC) design pattern greatly improved the quality and understandability of our code. MVC allows our application to have the business logic separated from frontend views, and to have our data models separate as well. These clear distinctions make the code much easier to extend upon, which made implementing the new requirements much easier.

For example, assignment 3 required the ability to send PATCH requests to the server. By following the MVC pattern, there was a clear place for this functionality to go (inside the controllers). The controllers would then return a model to the view, making all the pieces fit together easily.

However, implementing the MVC pattern also increased the complexity of our code, as many more classes were added, such as all of the controllers. Previously, if the frontend needed to make an API call, the request code would be directly there. However it is now in a different file, and the model file may need to be checked to see how the request works with the model. Overall, implementing MVC makes our code much easier to understand and maintain, due to the separated nature of models, business logic and views.

References

Nazar, N. (2021). Week 2 Object Oriented Analysis https://lms.monash.edu/my/
Nazar, N. (2021). Week 3 Object Oriented Design Principles https://lms.monash.edu/my/
Nazar, N. (2021). Week 4 Object Oriented Design Principles - II https://lms.monash.edu/my/
Nazar, N. (2021). Week 5 Software Design Patterns - I https://lms.monash.edu/my/
Nazar, N. (2021). Week 6 Software Design Patterns - II https://lms.monash.edu/my/
Nazar, N. (2021). Week 7 Software Architecture - MVC https://lms.monash.edu/my/
Nazar, N. (2021). Week 9 Refactoring https://lms.monash.edu/my/