51 In Class Competition Board

Bu Sun Kim
Kumar Bhargav Srinivasan

Vision: To be a convenient programming competition platform for faculty and students.

Description: A competitive programming platform where users can post coding competitions and participate in posted competitions.

PROJECT REQUIREMENTS

Business Requirements

ID	Requirement	Topic Area	Actor	Priority
BR001	Admin User should be able to verify faculty.	verification	Admin	High
BR002	Admin User should be able to verify student.	verification	Admin	High
BR003	Faculty should be able to track all the submissions	Submissions	Faculty	Medium
BR004	Student should be able to see all the submissions	Submissions	Student	Medium
BR005	Password should be atleast 6 characters	Login	Student/faculty	low
BR006	User Name should be alphanumeric	Login	Student/Faculty	low

User Requirements

ID	Requirement	Topic Area	Actor	Priority
UR001	Student/Faculty view all competitions posted	View	Student/Faculty	High
UR002	Student should be able to join a particular competition	Participate	Student	High
UR003	Student should be able to upload submission for competition	Submissions	Student	Medium
UR004	Student should be able to see results of his submissions for competition	Submissions	Student	Medium
UR005	Faculty should be able to post and start the competition	Post Competition	Student	High
UR006	Faculty should be able to upload test case for completion of the competition	Post Competition	Faculty	High
UR007	Faculty should be able to stop the competition	Post Competition	Faculty	Low
UR008	Admin should be able to allow/disallow student registered to compete	Authentication	Admin	Medium
UR009	Admin should be able to allow/disallow faculty registered.	Authentication	Admin	Medium

Nonfunctional Requirements

ID	Requirement	Topic Area	Priority
NF001	User account should be password-protected	Security	High
NF002	Normalized database,allowing for easy database changes	Database consistency	High
NF003	Competitions Posted should be visible realtime	Performance	Medium
NF004	Submission posted should be evaluated within a minute	Performance	Medium
NF005	Confirmation popups before and after submissions	User experience	High

- **Fully Implemented:**
  - BR003
  - UR001
  - UR002
  - UR003
  - UR004
  - UR005
  - UR006
  - NF001

    - NF002
    - NF003

- **Partially Implemented**
    - BR001: Endpoint exists on backend, admin page not created on UI.
    - BR002: Endpoint exists on backend, admin page not created on UI.

- **Not Implemented:**
    - BR004
    - BR005
    - BR006
    - UR007
    - UR008
    - UR009
    - NF004
    - NF005

Show your Part 2 class diagram and your final class diagram.

Please see page 5 for the Part 2 class diagram and page 6 for the final class diagram.

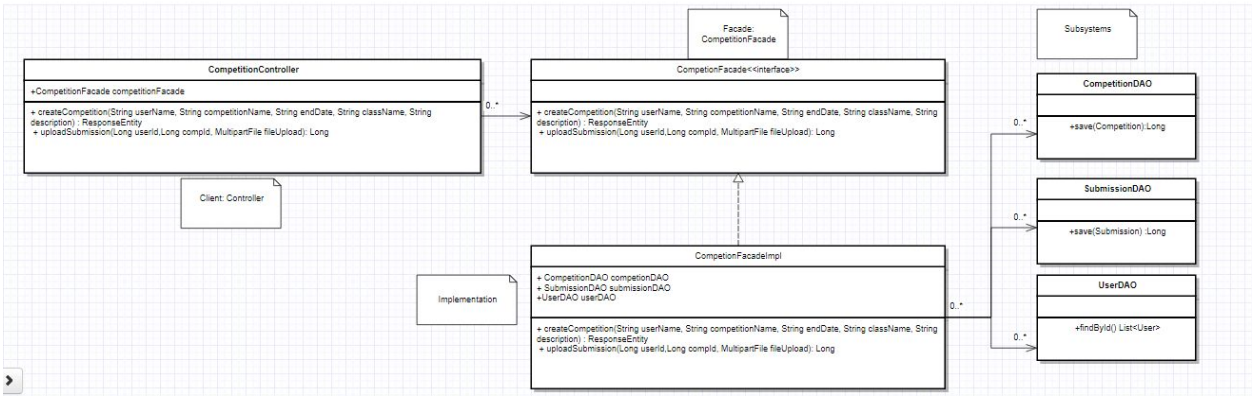
What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Our design changed quite a bit from part 2. We added the facade and factory design pattern, which changed the layout of the class diagram. In addition, originally our views were implemented as additional classes, but we pulled out the Frontend system to Angular and had the Frontend interact with the Java codebase through the controllers.

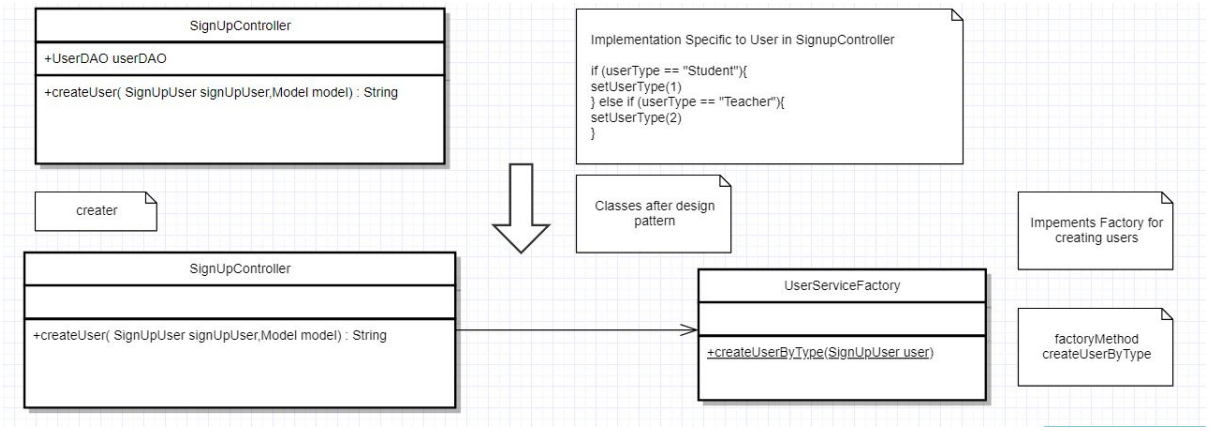
Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF).

Facade Pattern: Act as an Interface between client controller and Data Access Objects. DAO Objects are used by multiple controllers directly and as the DAO objects are subject to change, we need an abstraction layer which ensures that client is independent of the internal implementation at database layer and Hence Facade. This design pattern follows the open close principle and the Interface Segregation Principle which ensure that underlying Database implementation of the application flexible to change, without any change in the controller code, there by decreasing coupling.

Following are the diagram of classes which were involved:



We also implemented Factory Pattern: Create User Objects using factory. We had three types of users in the system and we had to have an Unified interface to create users such as Student and Faculty by setting appropriate type based on the request.

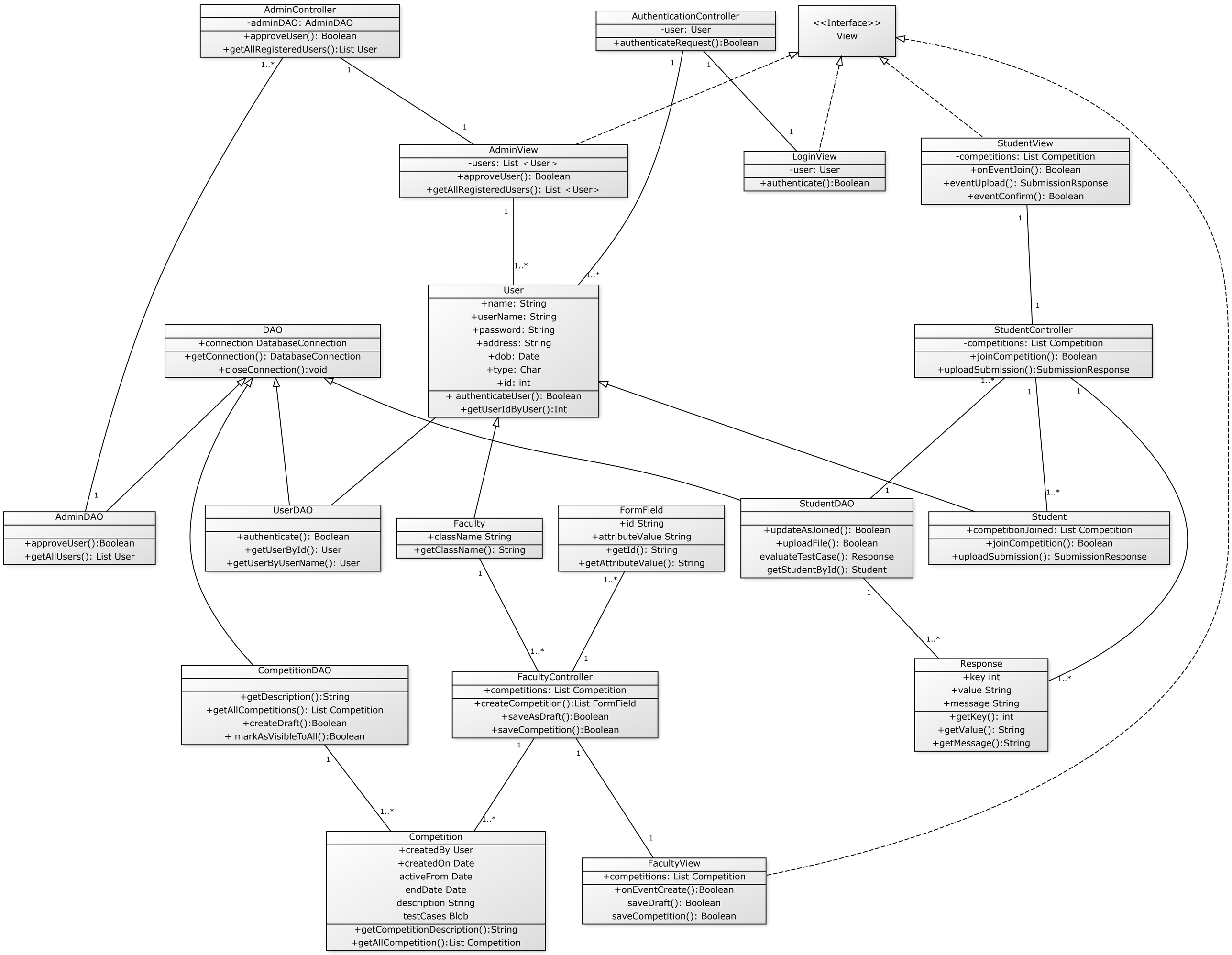


What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

- We learned to use Object oriented programming in developing production level web application.
- We started off learning in the class with object oriented principles, object oriented designs, design patterns and then refactoring, We were able to understand and appreciate the power of the OOP when we applied the concepts to the project.
- Class, Sequence, Activity diagrams helped understand different aspects of design and development.
- We understood the importance of requirement analysis and design which is underrated by typical programming mindset. we realized the amount of time spent was less in the implementation due to time spent in designing the system

right. Also, due to extensive designing of the project, there was a seamless transition to implementation.

- We kept improvising the class diagrams by using concepts learnt such as design patterns and refactoring which helped in improving quality,reusability and thereby avoid code smells, antipatterns.


AdminController
-adminDAO: AdminDAO
+approveUser(): Boolean
+getAllRegisteredUsers():List User
AuthenticationController
-user: User
+authenticateRequest():Boolean
<<Interface>>
View
AdminView
-users: List <User>
+approveUser(): Boolean
+getAllRegisteredUsers(): List <User>
LoginView
-user: User
+authenticate():Boolean
StudentView
-competitions: List Competition
+onEventJoin(): Boolean
+eventUpload(): SubmissionRsponse
+eventConfirm(): Boolean
User
+name: String
+userName: String
+password: String
+address: String
+dob: Date
+type: Char
+id: int
+ authenticateUser(): Boolean
+getUserIdByUser():Int
DAO
+connection DatabaseConnection
+getConnection(): DatabaseConnection
+closeConnection():void
StudentController
-competitions: List Competition
+joinCompetition(): Boolean
+uploadSubmission():SubmissionResponse
AdminDAO
+approveUser():Boolean
+getAllUsers(): List User
UserDAO
+authenticate(): Boolean
+getUserById(): User
+getUserByUserName(): User
Faculty
+className String
+getClassName(): String
FormField
+id String
+attributeValue String
+getId(): String
+getAttributeValue(): String
StudentDAO
+updateAsJoined(): Boolean
+uploadFile(): Boolean
evaluateTestCase(): Response
getStudentById(): Student
Student
+competitionJoined: List Competition
+joinCompetition(): Boolean
+uploadSubmission(): SubmissionResponse
CompetitionDAO
+getDescription():String
+getAllCompetitions(): List Competition
+createDraft():Boolean
+ markAsVisibleToAll():Boolean
FacultyController
+competitions: List Competition
+createCompetition():List FormField
+saveAsDraft():Boolean
+saveCompetition():Boolean
Response
+key int
+value String
+message String
+getKey(): int
+getValue(): String
+getMessage():String
Competition
+createdBy User
+createdOn Date
activeFrom Date
endDate Date
description String
testCases Blob
+getCompetitionDescription():String
+getAllCompetition():List Competition
FacultyView
+competitions: List Competition
+onEventCreate():Boolean
saveDraft(): Boolean
saveCompetition(): Boolean