OREGON STATE UNIVERSITY, COMPUTER SCIENCE

CS 361 Software Engineering I
HW 4: Design Assignment

STUDY HABIT PHONE APPLICATION

Brendan Jang, David Passaro, Casey Hines,
Christopher Teufel, Conner Rhea

November 10, 2019

PROPOSAL PARAPHRASE FROM CUSTOMER

The intention of this application is to aid students with their study habits. Students will enter their class information, and establish goals for each class. The goals include time allocation and desired grade. The application will make recommendations to adjust the time allocation based on the students current grade, remind the student when it is time to study, set a timer for the student during a study session which count down the time until a break and disable notifications from other applications during a study session.

There is also a store of study tips to aid the student in developing good study habits. Other tools will exist to help students prepare for tests, such as the ability to create flash cards, review materials, and organize notes either imported from Google Drive, or scanning in handwritten notes.

The application will integrate with common learning management systems (LMS) such as Canvas in order to remind students about due dates, and make recommendations based on performance. Recommendations will be tailored to the students' specific needs, such as test taking recommendations if the student is doing well on assignments, but not on exams.

1 UML Class Diagram

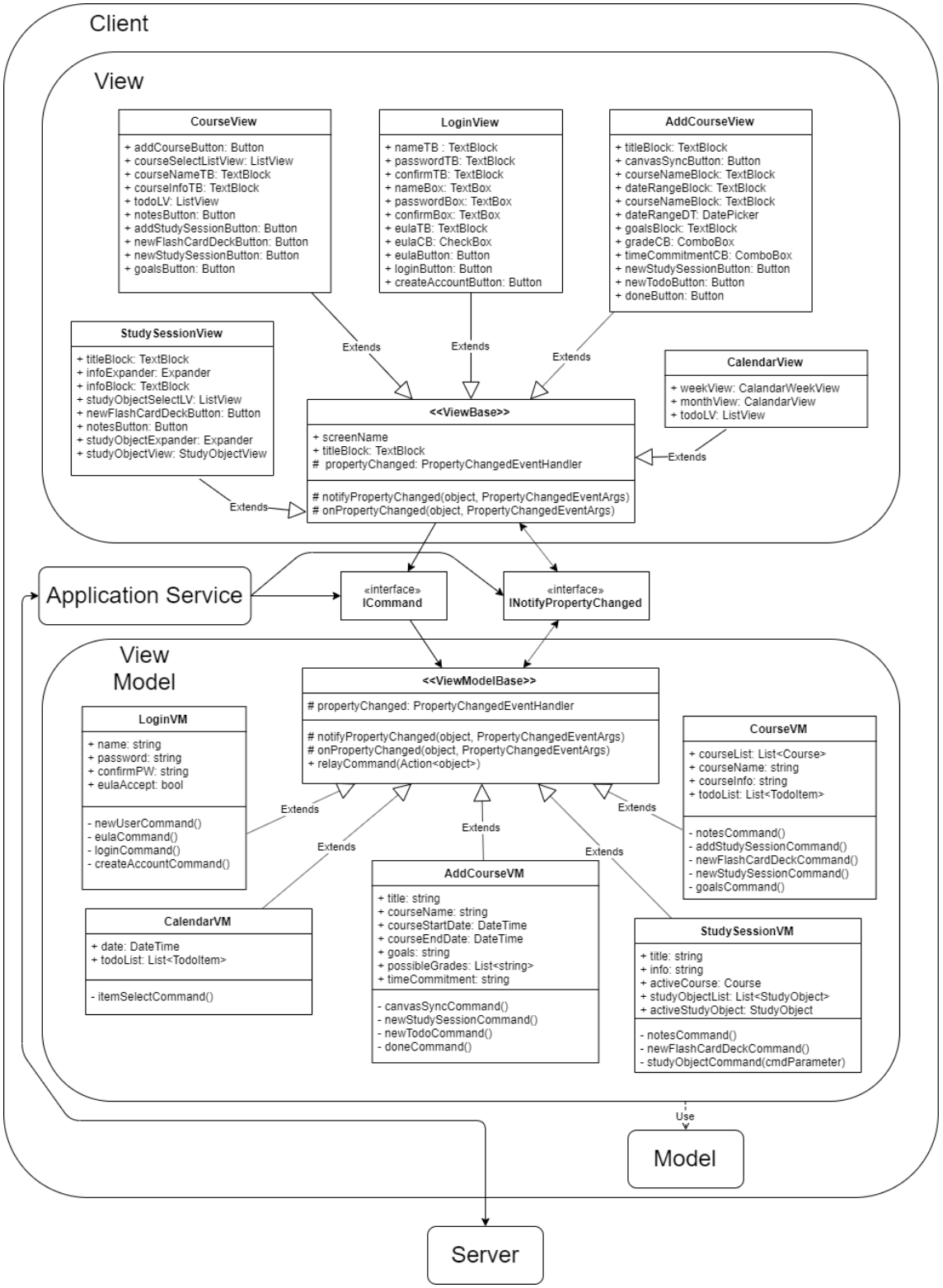

Figure 1.1: This is part of the UML class diagram for the client. Here, the View and View Model are shown along with the connecting subsystems.

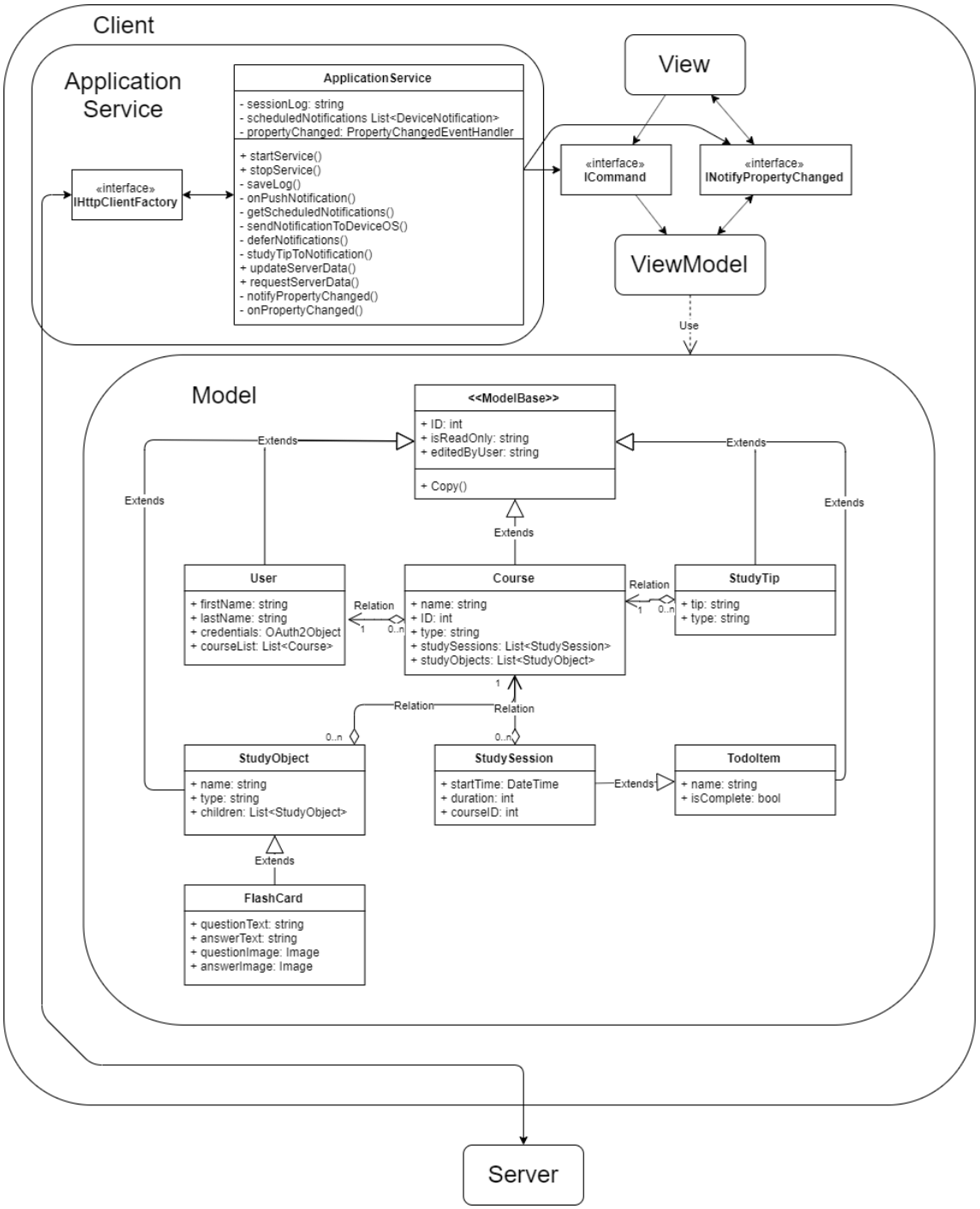

Figure 1.2: This is the second part of the UML class diagram for the client. Here, the Model and Application Service are shown along with the connecting subsystems.

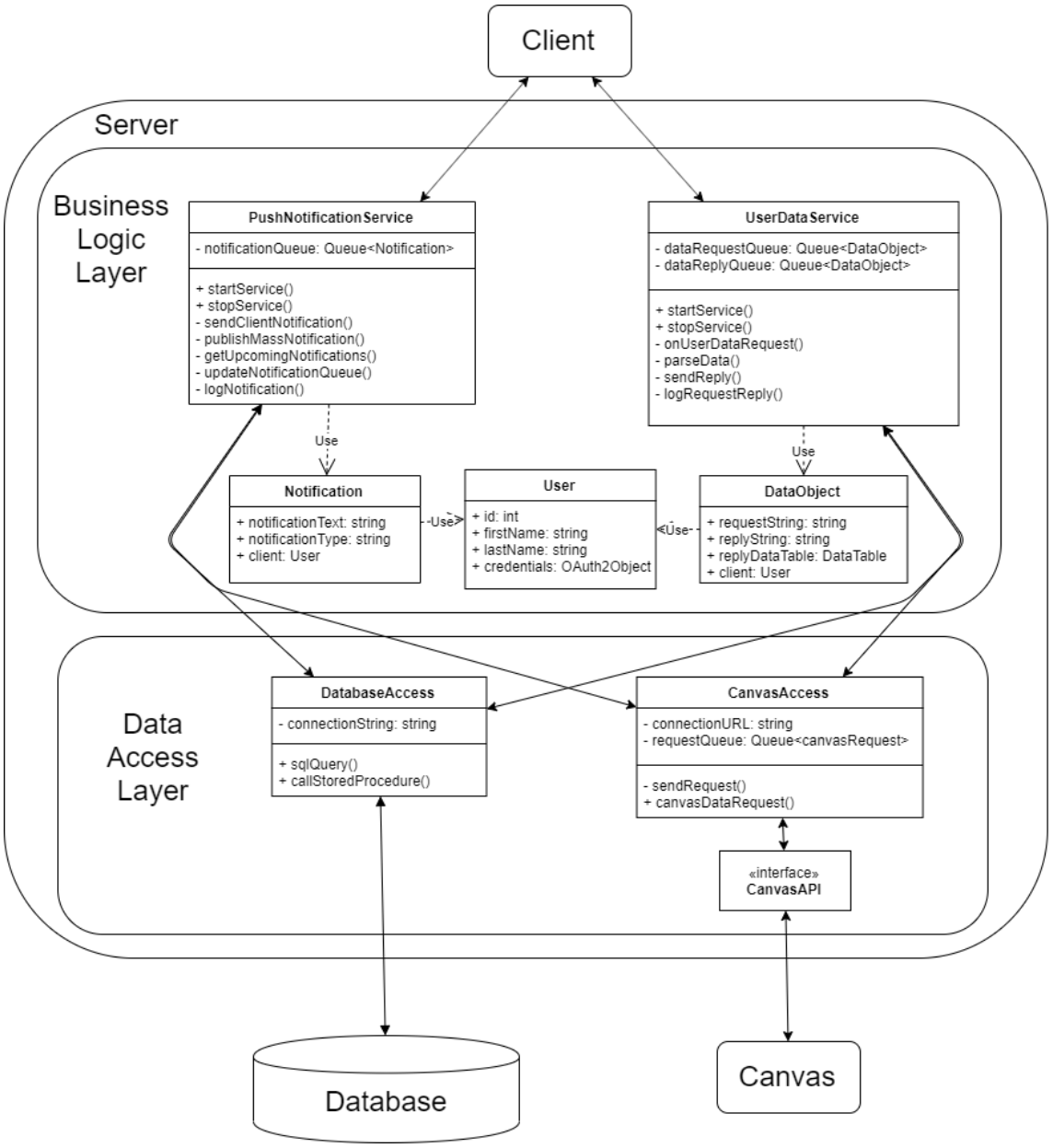

Figure 1.3: This is the UML class diagram for the server with connecting subsystems.

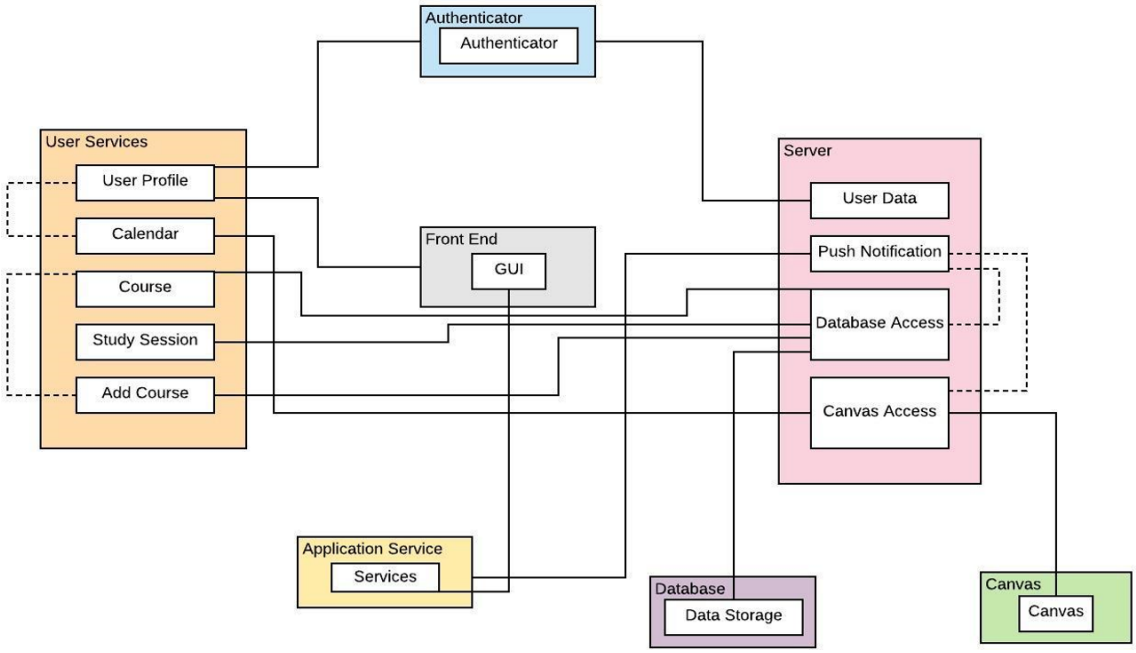

Figure 2.1

2.1 Coupling

When one module is involved in another module's concern.

- Content coupling
  - Add Course modifies Course.
  - Authenticator modifies User Profile access.
  - User Profile can modify Calendar.
- Common coupling
  - Study Session and Add Course both call on Database Access.
  - Study Session and Add Course both call on Database Access.
  - Course and Add Course both call on Database Access.
- Control coupling
  - User profile calls the authenticator.
  - Calendar calls on Canvas Access.
  - Course calls on Database Access.
  - Study Session calls on Database Access.
  - Add Course calls on Database Access.
  - Database Access calls on Data Storage.
  - Canvas Access calls on Canvas.
  - Services calls on Push Notifications.
  - Authenticator calls on User Data.
  - Push Notification calls on Database Access.
  - Push Notification calls on Canvas Access.
- Stamp coupling
  - Calendar would provide structured data to the user profile.

- Add course module provides structured data to the course module.
- User Data provides structured data to the authenticator.
- Authenticator provides structured data to the user profile.
- User Profile sends structured data to the GUI.
- Services send structured data to the GUI.
- Push Notification provides structured data to Services.

2.2 Cohesion

When a module is devoted to its concern.

- Functional/Informational Cohesion
  - The Authenticator provides security to the overall system.
  - The GUI provides an accessible interface for users.
  - User Services is used to send and receive user data.
- Communicational Cohesion
  - The Server's purpose is to provide relevant data requested from various services through the system.
  - The Application Services's purpose is to provide information to the GUI to monitor the status of various system functions.

3 Design Development Assessment

Incremental development would be best supported for the design of this system because of the general layout that we have planned. Ideally, the system has to be built in steps with overall functionality depending on the completion of the previous step. In this system, the server along with its necessary components (Database) and connections (Canvas) must be built first.

For the user to access the core features of the system, the user profile, calendar, course, study session, and add course classes have to be built next. Logically, the flow of our system development would be as follows: the back-end of our system would be developed first and the front end would follow. Once the system is operational, we could probably use iterative development to add more features or improve upon older ones.

4 Potential Design Patterns

4.1 Facade

Our application aims to provide a clean and easy to use GUI for our users to study for exams. The Facade Design Pattern is a perfect fit to provide a unified interface for our clients to easily access the different functions within. A Facade system will allow users to focus only on the things important to their study while a subsystem handles the connection to canvas and the data structures for the various studying materials. This way the system will remain easy to use for users of all ages without needing to worry about the complexities beneath the surface.

4.2 Observer

Two important systems in our application are keeping track of the time the user spends studying in order to meet their goals, and to periodically provide the user with tips and tricks

to make their studying more effective. To this end, implementing an Observer Design Pattern to monitor the system clock and keep track of how long a user does/doesn't study is integral to meeting our client's specifications. Watching for instances where the user stalls during studying will allow the application to interject if the user is struggling.

4.3 Adapter

As a result of our application relying on Canvas, a system with already existing functionality, we will need to use this Design Pattern to implement the link between them. In order to properly ensure that data is transferred between Canvas and users we'll need to make sure that we can interface with Canvas to download the necessary information for systems such as flashcards and accessing study materials provided through the class. Due to Canvas being a closed system and third party to our application, we'll need to make sure our communication is secure as we won't be able to modify the Canvas side.

4.4 Builder

Creating flashcards and other study tools in an integral part of our system. The Builder Design Pattern will allow for the creation of multiple complex structures for tools such as notes, flashcards and other study materials. By making all of these tools stem from a central class we can simplify the instantiation of new study material objects, and if the need to create new types of study tools arise we can use the generic class to create new representations.

5 Sequence Diagram of the Study Session Use Case

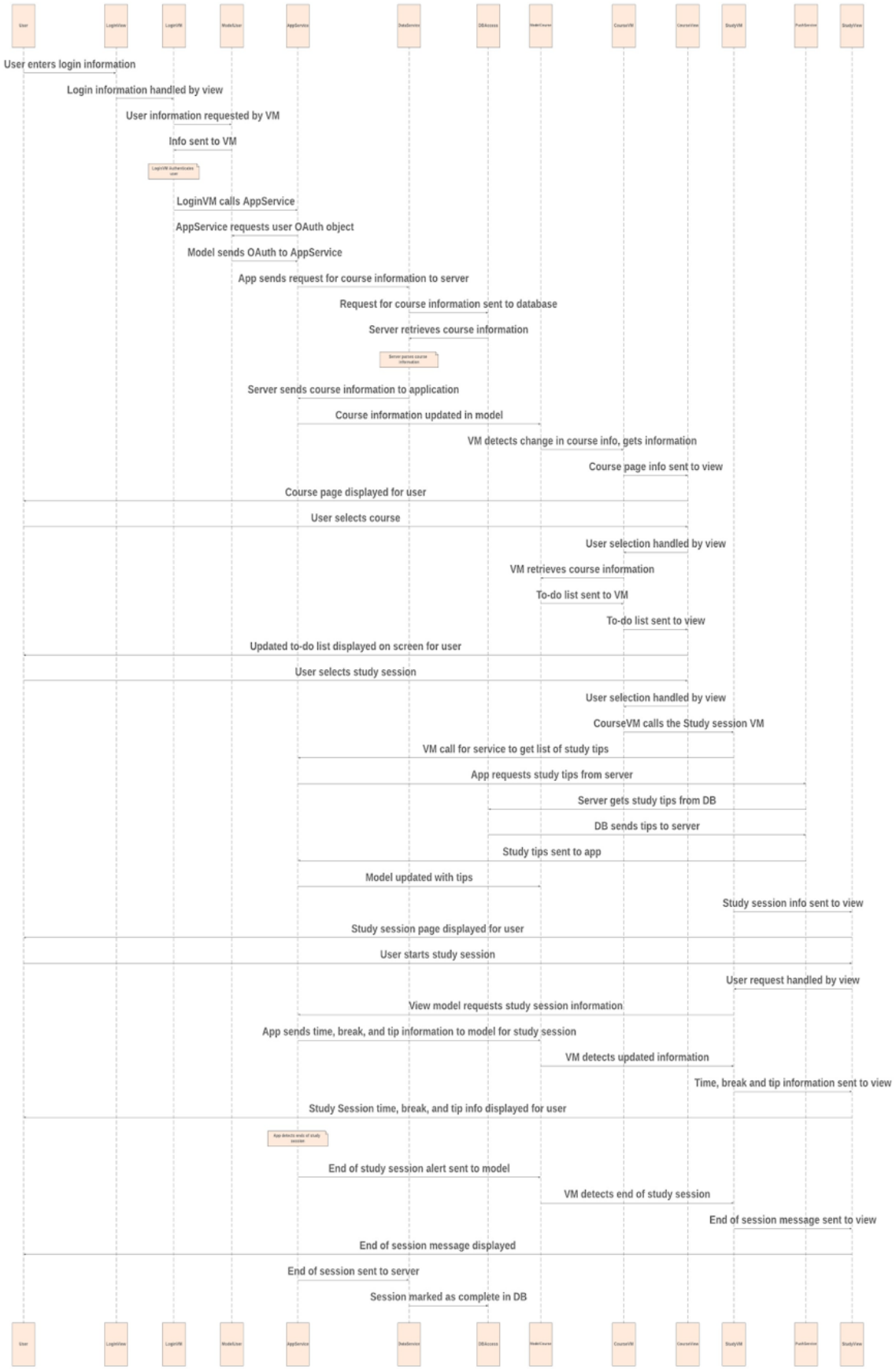

Figure 5.1

6 INTERFACES

There are four interfaces explicitly identified in the UML class diagram.

1. ICommand interface
   This interface is a command binding interface that is used to loosely couple user actions from view objects to command methods in the respective view model class.

   Contract: If a UI control that participates in command binding is triggered by the user, this interface will trigger the view model class (via an event) to call the command method that the control was bound to.

2. INotifyPropertyChanged interface
   The INotifyPropertyChanged interface aids in data binding, where A property in a view class is loosely coupled (again, via an event system) to a property in the respective view model class, visa-versa, or both.

   Contract: If view model class has a property that is modified, and is bound to a UI control property, the UI control property is also modified to reflect the change in the view model. This can also work the other direction.

3. IHttpClientFactory interface
   This interface constructs a secure HTTP client instance that connects to a server allowing for data transmission.
   Contract: A class that implements this interface can create HTTPS client instances used to make HTTP requests.

4. Canvas API
   This interface is provided by Canvas and used to communicate with Canvas. The contract here is simply each API call will yield a response from Canvas.

The following are some other potential interfaces. Many are specific cases related to the interfaces described above.

1. Preconditions: During login LoginView takes correct login information from the user as LoginView.nameTB and LoginView.passwordTB.
   Postconditions: The LoginVM will obtain the login information from LoginView as LoginVm.name and LoginVM.password. Contracts: LoginView.nameTB = LoginVm.name LoginView.passwordTB =LoginVM.password

2. Preconditions: While logged in user inputs course selection data in the Add Course View, stored as AddCourseView.courseName.
   Postconditions: The data is transferred to the View Model Base in AddCourseVM as AddCourseVM.courseName. Contract: AddCourseVM.courseName = AddCourse-View.courseName.

3. Preconditions: Study session has been input previously and has been signaled as imminent by the system clock.
   Postconditions: Business Layer Logic outputs a signal to Application Service. System will trigger alarm in host system.
   Contract: BLL.PushNotificationService.startService() triggers ApplicationService.send-

NotificationsToDeviceOS()

4. Preconditions: Course selection in the view has transferred data to AddCourseVM. courseName.
   Postconditions: AddCourseVM has triggered ApplicationService.updateServerData().
   Contract: AddCourseVM.courseName triggers ApplicationService.updateServerData().

5. Preconditions: User clicks on StudySessionView.newFlashCardDeckButton.
   Postconditions: StudySessionVM.newFlashCardDeckCommand() sends update information to Application Service.updateServerData()
   Contract: StudySessionView.newFlashCardDeckButton triggers StudySessionVM. newFlashCardDeckCommand()

6. Preconditions: User enters app, chain of events sets off BLLUserDataService.LogRequestReply() sending the login information to the DataAccessLayer.
   Postconditions: DataAccessLayer DatabaseAccess.sqlQuery called for verification of user login information.
   Contract:BLLUserDataService.LogRequestReply() triggers DatabaseAccess.sqlQuery.

7. Preconditions: User creates a profile and defines login information, LoginVM. newUserCommand() triggered.
   Postconditions: System will allow users to login to their profile with correct login information and save their credentials for future sessions, info sent off to ModelView User.firstName, User.lastName, User.credentials.
   Contract: LoginVM.newUserCommand() triggers ModelView User.firstName, User.lastName, User.credentials data.

7 EXCEPTIONS

1. User enters incorrect login information.
   System will show error message and allow three more chances for correct information to be given, else locking the account for 15 minutes and displaying a 'contact company' message.

2. User exits flash card UI without saving changes.
   System displays message "would you like to save changes?"

3. If a user attempts to connect to Canvas without a valid internet connection, the System should display a message indicating that the application cannot connect to the server as well as display an error message corresponding to the issue and recommending the user check their connection settings.

4. In the event of an internal system error such as a programmatic error, mathematical error, or any variety of other errors resulting from a bug or fault in the code should be handled through some form of error handling. The application should have a variety of error messages corresponding to different cases of internal errors and in the event of an error the application should offer for the user to send a bug report to us so that we can adjust the software.

5. If a user tries to remove or delete a class the app will remind users that deleting a course will remove all materials related to that course and ask again if they are sure. If a user confirms that they wish to delete the course, all saved materials corresponding to the class will be deleted to make space.

6. Server/DB Errors - If the user attempts to access information located on a server or database and there is an error on the server side, the app should be able to detect the status code from any server requests and display a message to the user that certain information is unavailable at this time. This means that the app should be checking status codes for all server responses. Additionally, if the server is unable to query the database due to a database error, the server should respond to the user app with the appropriate status code.

8 Team Member Contributions

UML Class Diagrams: Casey and Chris
Package Implementation: Brendan
Development Style: Brendan
Interfaces: David and Casey
Use case sequence diagram: Chris
Exceptions: Dave, Chris, and Conner
Create Final Draft: Casey