Oregon State University, Computer Science

# CS 361 Software Engineering I
# HW 3: Architecture

Study Habit Phone Application

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

November 3, 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 High Level Architecture

## 1.1 Layered Architectural Style

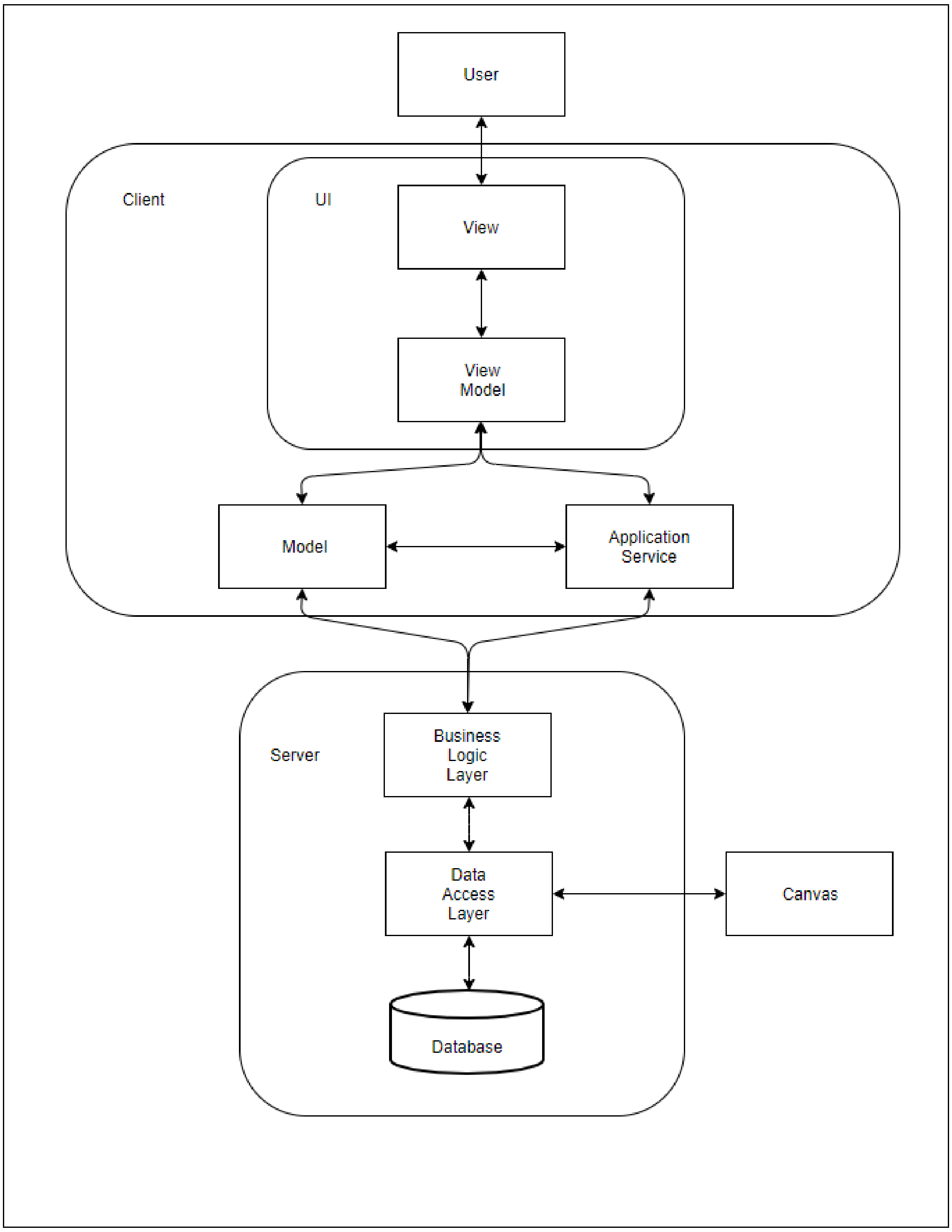

Figure 1.1: This is one of the proposed high-level architectures that uses a layered design style.

In the proposed layered architectural style, the client is decomposed into several layers. The view is the part of the UI that the user directly views and interacts with. This includes things like buttons, text I/O, artwork, etc. The view model contains the underlying objects and logic required by the view. The model is primarily a container for all the data needed at run-time. The application service runs in the background keeping the application data up-to-date by periodically synchronizing with the server and Canvas. It also propagates scheduled and push notifications from the server through the device OS.

The server is also decomposed into several layers. The business logic layer takes care of interpreting the requests from clients and constructs valid replies that are secure and easily consumed by the clients. The data access layer handles sending and retrieving data from each of the data sources. In this case the data sources are the system database (yet another layer), and Canvas.

### 1.2 Client-Server-Repository Style Architecture

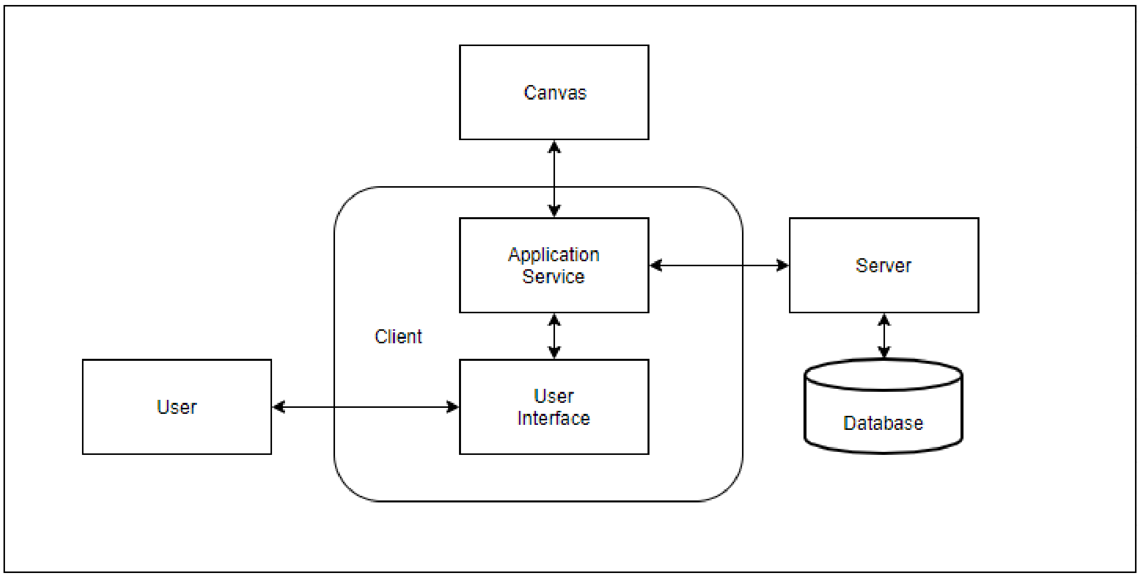

Figure 1.2: This is one of the proposed high-level architectures that uses a client-server and repository design style.

The repository style architecture uses a client-server design with services for storing and accessing data. A key attribute of this particular architecture is that Canvas connects directly to the client, not the server.

The client has two main components, the user interface and the application service. The application service is the same as it was described in the layered architecture. It runs in the background keeping the application data up-to-date by periodically synchronizing with the server and Canvas. It also propagates scheduled and push notifications from the server through the device OS.

# 2 Key Quality Attributes

### 2.1 Efficiency

#### *2.1.1 Efficiency for Layered Style*

- Normal use of the app would require pulling user data and resources from the server resulting in a latency period when the request is sent to the server and the response is

sent back to the client. However, this information would be in a format that is easily processed by the client.
- If Canvas syncing occurs when the app is in use, then there would be a greater latency period as the request for data would be sent to the server and then to Canvas, and the response from Canvas to the server and then back to the client. However, the server would be able to change any Canvas data into a format that is easily used by the client faster than the client would. However, Canvas interaction could be done while the app is not in use and stored with the rest of the user's information. This would cut down on latency time. Resource management of the server should be implemented in a way that does not interfere with normal requests from the client.
- Scaling would require an increase in server storage and processing capabilities as well as a greater need for Canvas API interaction management.

#### *2.1.2 Efficiency for Client-Server-Repository Style*

- Similar to the layered architecture, normal use of the app would require pulling user data and resources from the server and the same latency period would exist.
- Syncing user information would be more inefficient in this architecture as the client would have to get information from the Canvas API and convert it to a format that the server would use. This information would also have to be sent to the server for future use. However, less information would be pulled from the server resulting in a lower amount of information processing by the client.
- Scaling would require less of an increase in server capabilities as all API interaction is handled by the client and less information is being pulled from the server.

### 2.2 Integrity

#### *2.2.1 Integrity for the Layered Style*

- User would be authenticated by server which would need to be secure to protect any user information. Since Canvas interaction would take place via the server, the server would be required to store any authorization information (OAuth 2 according to Canvas API docs) as well. This only adds to the need for best security practices for the server.
- The server would also need to be updated with any change in the user's Canvas credentials.

#### *2.2.2 Integrity for Client-Server-Repository Style*

- Similar to the layered architecture, the user would still be authenticated by the server, meaning the server would still have to store authentication information for each user.
- Unlike the layered architecture, Canvas interaction is handled by the client. This means that any authorization information could be stored locally on the client side. Additionally, any change in the users Canvas information (classes, authorization, etc.) could be easily handled by the client without having to send information back and forth to the server. This makes the serves less of a target from a security standpoint.

### 2.3 Portability

#### *2.3.1 Portability for the Layered Style*

- The only change in the overall system from platform to platform would be a change in the client. The overall structure of the client would remain the same from platform to platform. How the views are implemented and how the application service runs would change with a different platform.

- Information sent to the client from the server would still be in a standardized format to be processed by the client.
- Running on multiple systems would have no effect on Canvas-server interaction as it all handled by the server.

### *2.3.2 Portability for the Client-Server-Repository Style*

- Similar to the layered architecture, the only change from platform to platform would be to the client. The overall structure would remain the same, but the implementation may vary from platform to platform.
- Interaction with the Canvas API would be similar from platform to platform as information would still be sent and received in a standard format.
- Information received from the server would still be in a standard format regardless of the platform.

# 3 Failure Modes

## 3.1 Canvas Data Failure

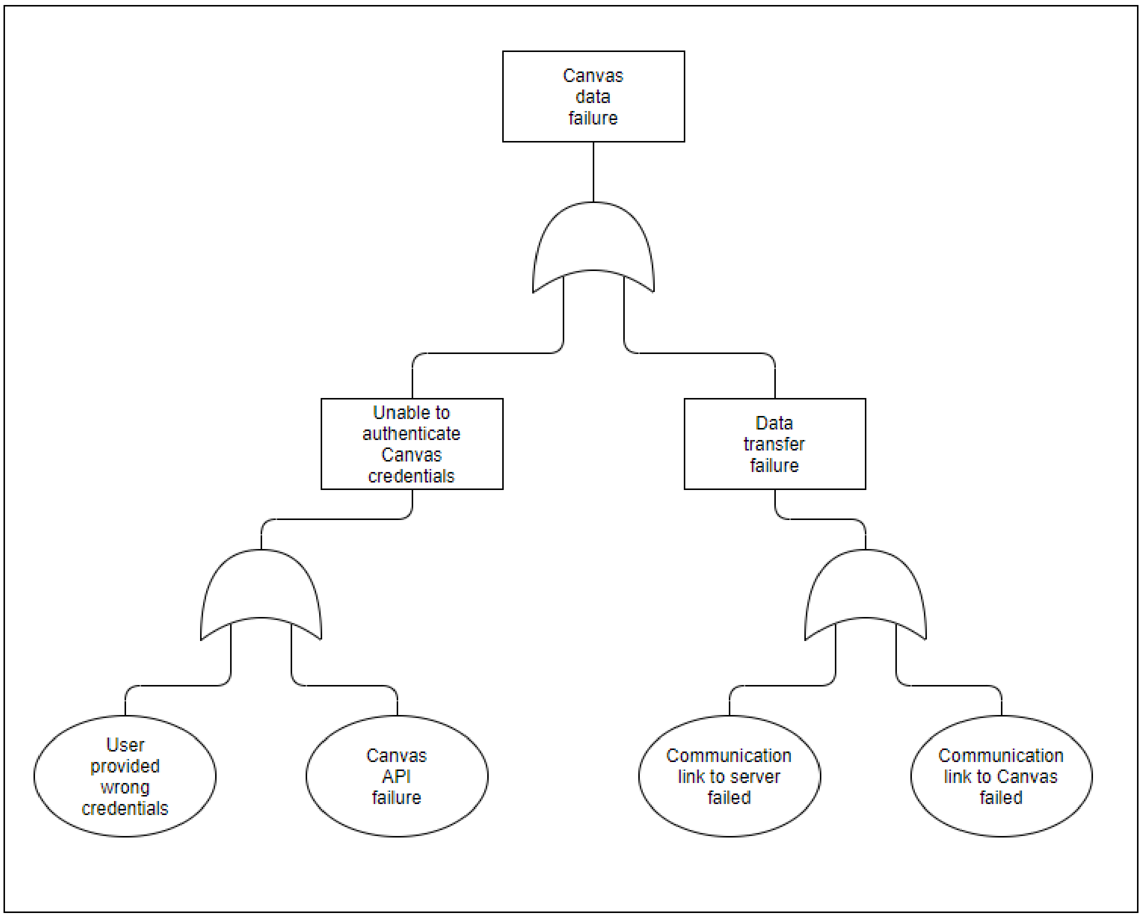
Canvas data failure
Unable to authenticate Canvas credentials
Data transfer failure
User provided wrong credentials
Canvas API failure
Communication link to server failed
Communication link to Canvas failed

Figure 3.1: This is a fault tree diagram that illustrates ways in which communication with Canvas can fail.

The Client-Server-Repository architecture is probably more prone to this type of failure because the communication link between Canvas and the server is likely to be more stable than the communication link between Canvas and the UI.

### 3.2 Failure to Add Course

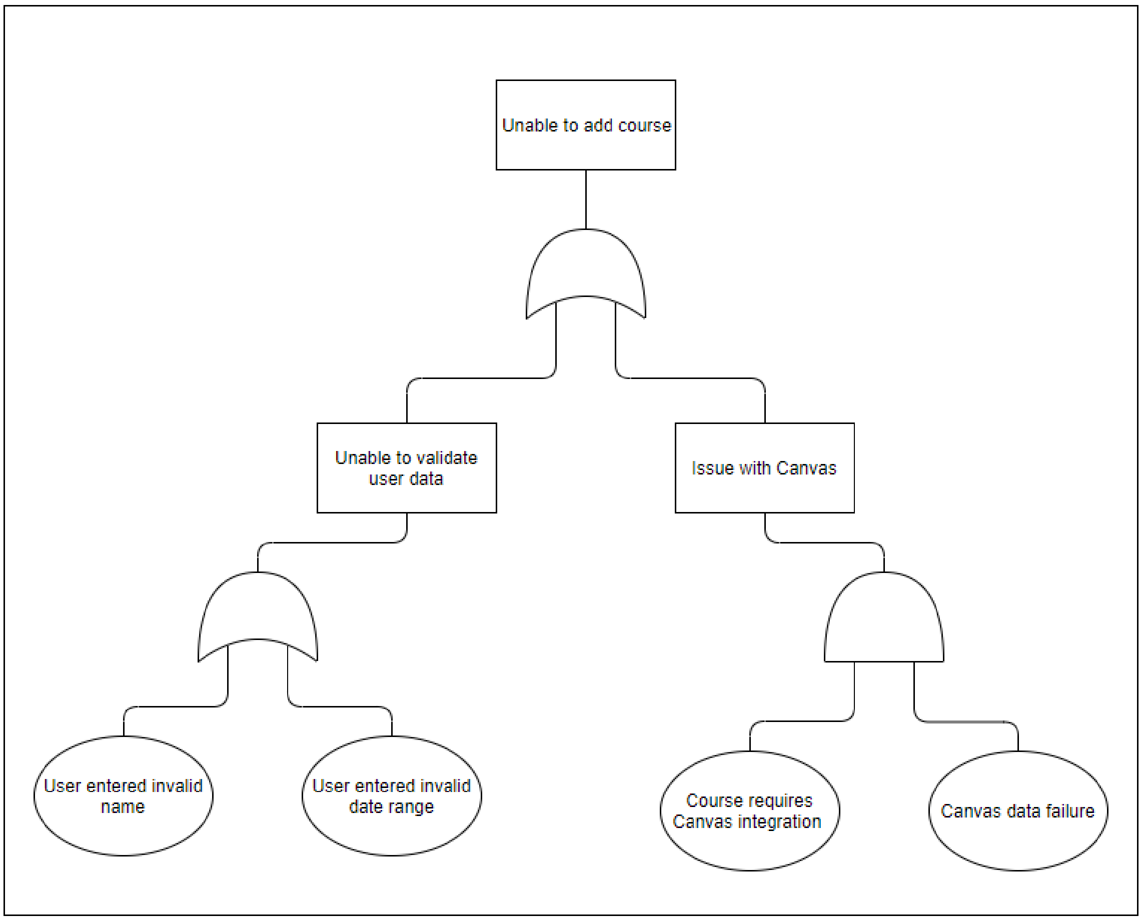
Unable to add course
Unable to validate user data
Issue with Canvas
User entered invalid name
User entered invalid date range
Course requires Canvas integration
Canvas data failure

Figure 3.2: This is a fault tree diagram that illustrates ways in which the process of adding a course can fail.

For this failure mode, the data validation branch of the tree doesn't depend on the architecture, however, a Canvas data failure is one of the leaves on the other branch, which was determined to be more susceptible to failure in the Client-Server-Repository Architecture.

## 4 Decomposition of Layered Architecture

## 4.1 Application Service Decomposition

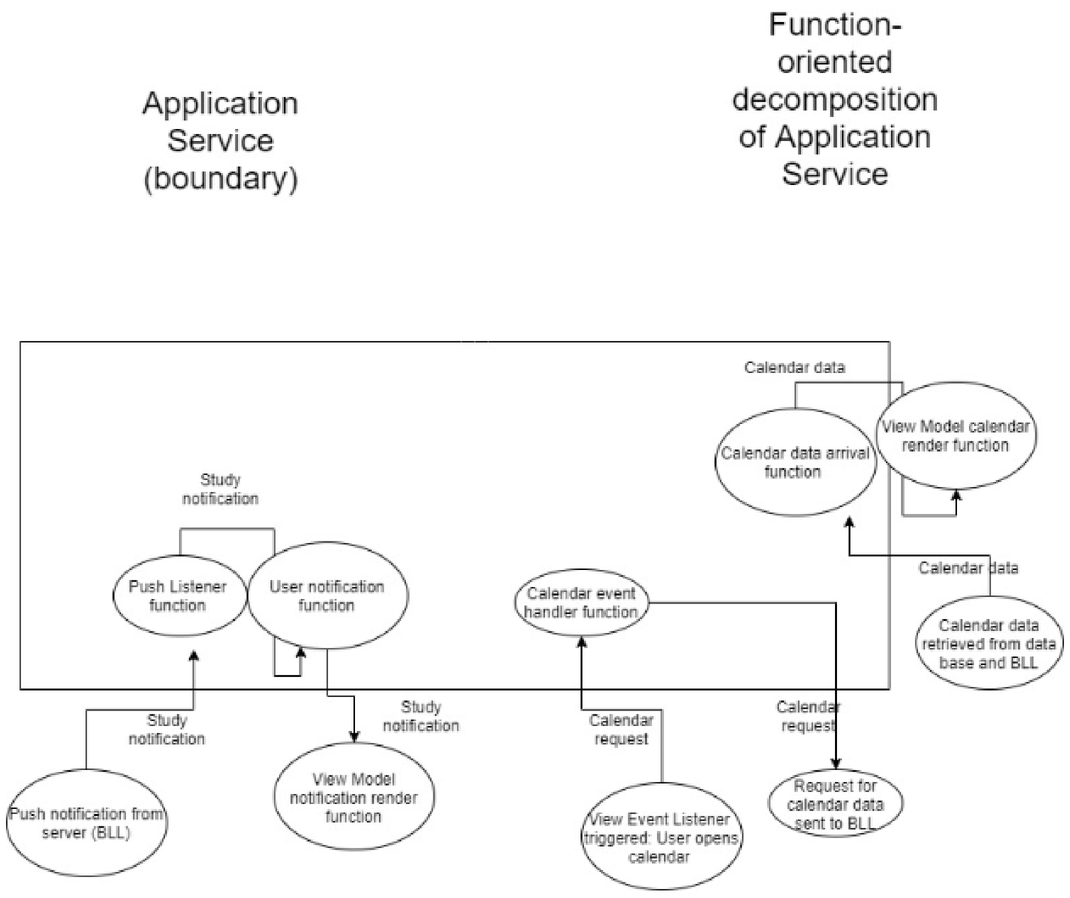
Application Service (boundary)
Function-oriented decomposition of Application Service
Calendar data
Calendar data arrival function
View Model calendar render function
Study notification
Push Listener function
User notification function
Calendar event handler function
Calendar data
Calendar data retrieved from data base and BLL
Study notification
Study notification
Calendar request
Calendar request
Push notification from server (BLL)
View Model notification render function
View Event Listener triggered: User opens calendar
Request for calendar data sent to BLL

Figure 4.1: This is a function-oriented decomposition of the Application Service which runs in the background on the client device. The Application Service handles notifications and synchronizing data with the server.

## 4.2 Business Logic Layer Decomposition

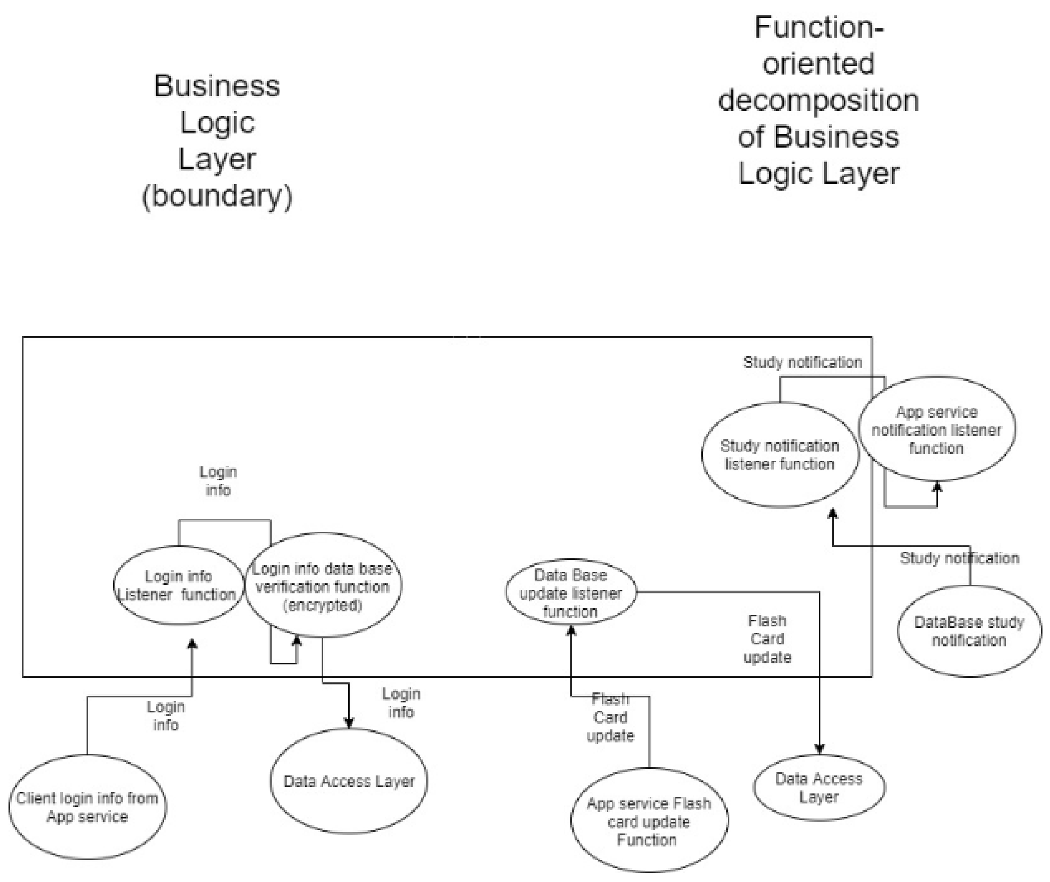
Business Logic Layer (boundary)
Function-oriented decomposition of Business Logic Layer
Study notification
App service notification listener function
Study notification listener function
Login info
Study notification
Login info Listener function
Login info data base verification function (encrypted)
Data Base update listener function
Flash Card update
DataBase study notification
Login info
Login info
Flash Card update
Data Access Layer
Client login info from App service
App service Flash card update Function
Data Access Layer

Figure 4.2: This is a function-oriented decomposition of the Business Logic Layer.

## 5 Architecture Validation

Several use-cases were analyzed in the context of the layered architecture.

### 5.1 Use Case 1: Creating Profile and General Use

- With the layered architecture, the user will create an account upon first opening the application. Once the account is created, the user will log in and gain authorized access to the system.
- Upon authentication, the system will create a secure connection from the client to the server.
- After the server confirms secure login, the user will have access to all the features of the application. The user will be able to select appropriate classes, view important dates, grades, and expected study hours through the GUI. The client will be in communication with the database and will transfer relevant information back to the application.
- With the information transferred from database to system, the user will personalize their calendar and input (or import) their schedule through the GUI which will then be saved on the system. The user will then have the option to start a study session based on their schedule which is then tracked by the system.

- The system will notify the user with relevant information such as subject related tips, optimized breakpoints, general study tips, and when session ends.
- User will then have the option to terminate session or start a new session. If user chooses to terminate, they will be logged out and all processes and connections will be stopped. If user chooses to start a new session, processes and connections are maintained and user is prompted to continue with last session information or input new study information.

### 5.2 Use Case 2: Math Flashcards

- With the layered architecture, the user has already created an account during the first login and has authorized access to the system. Once authenticated and connected to the server, the system will attempt to connect to Canvas using the student login information previously stored in the database.
- Once a secure and valid connection is established between the Data Access Layer (DAL) and Canvas, the Business Logic Layer (BLL) will request and receive relevant class information from Canvas. The received information will be formatted and sent to the client to be displayed in the GUI.
- At this point the user will be presented with their current classes and have the option to select the class they want to study for.
- After the user selects a class, they chooses the option to create a set of flashcards. The client sends a request to create a new flashcard set to the server, and the BLL creates an empty set in the database. The user is then presented with an interface to create the cards with questions and their respective answers.
- For each card the user creates, the data is sent to the server, processed in the BLL and stored in the database.
- After the user has at least one set of flashcards, they will have the option to start a flashcard study session, create another flashcard set, or terminate.
- If user chooses to start a flashcard study session, the client send a request to the server, the BLL fetches the data for that set and sends it back to the client where the set stored in a flashcard set object in the Model. This triggers the View Model to randomize the collection of flashcards and present them to the user through the View.
- During a study session, the Application Service will generate notifications with relevant information such as subject related tips, optimized breakpoints, general study tips, and when session ends. Much of this is contextual based on the class the user is studying for. This type of information is stored in the database and therefore requested through the server. If user chooses to create another flashcard set, the system prompts user to choose another class to study for and the process is repeated. If the user chooses to terminate session, they will be logged out and all processes and connections will be stopped.

### 5.3 Use Case 3: Study Session

- With the layered architecture, the user has already created an account during their first login and has authorized access to the system. Once authenticated and connected to the system, the system will attempt to connect to the database.
- Once connected to the database, the user selects a study mode through the GUI. The system will pull information based on the user's input from the database and will display it to the user through the GUI.
- The user will select the appropriate study session which will be tracked through the system. The user will then start the session.
- The system will then notify the user with relevant information such as subject related tips, optimized breakpoints, general study tips, and when session ends.

- User will then have the option to terminate session or start a new session. If user chooses to terminate, they will be logged out and all processes and connections will be stopped. If user chooses to start a new session, processes and connections are maintained and user is prompted to continue with last session information or input new study information.

## 6 Implications of Validation and Verification

One issue that will arise from relying heavily on the server and Canvas for the application is that those are both potential failure points that will make the app unusable or restricted. In the event of Canvas going down such as for maintenance or an outage, the app will be unable to validate the user's credentials or login and retrieve any information about the user's courses for studying. Additionally, if the app's server were to similarly go down, the user wouldn't be able to make use of the service at all. A potential solution would be to allow users to store some information locally and have an offline mode that doesn't require the user to login. This will allow users to potentially plan for outages or study with the materials they have on hand in a worse case scenario, although, that would have a storage cost for the app itself.

One benefit that the server approach gives our project is ease of storage and transfer. Having everything stored on the server for easy access means that using the Study Application has little-to-no storage requirements for the user or their device. Additionally, if a user has multiple devices with the app such as a tablet, smartphone, or other device capable of running the application, they will be able to access their study materials anywhere simply through login. This does require a connection to the internet in order to make use of the app, however by allowing users this freedom of movement can help them study in a wide variety of situations that are convenient to them. A potential feature that could be added could be some kind of data sharing or connectivity between devices on the same account, such as using one device to show flashcard sets, and another having the relevant information for the subject ready on-hand for the user to check their knowledge with.

## 7 Team Member Contributions

Architecture Dataflow Diagrams: Casey
Quality Attributes: Chris
Failure Modes: Casey
Further Decomposition: David
Validation: Brendan
Implications of Validation: Conner
Create Final Draft: Casey