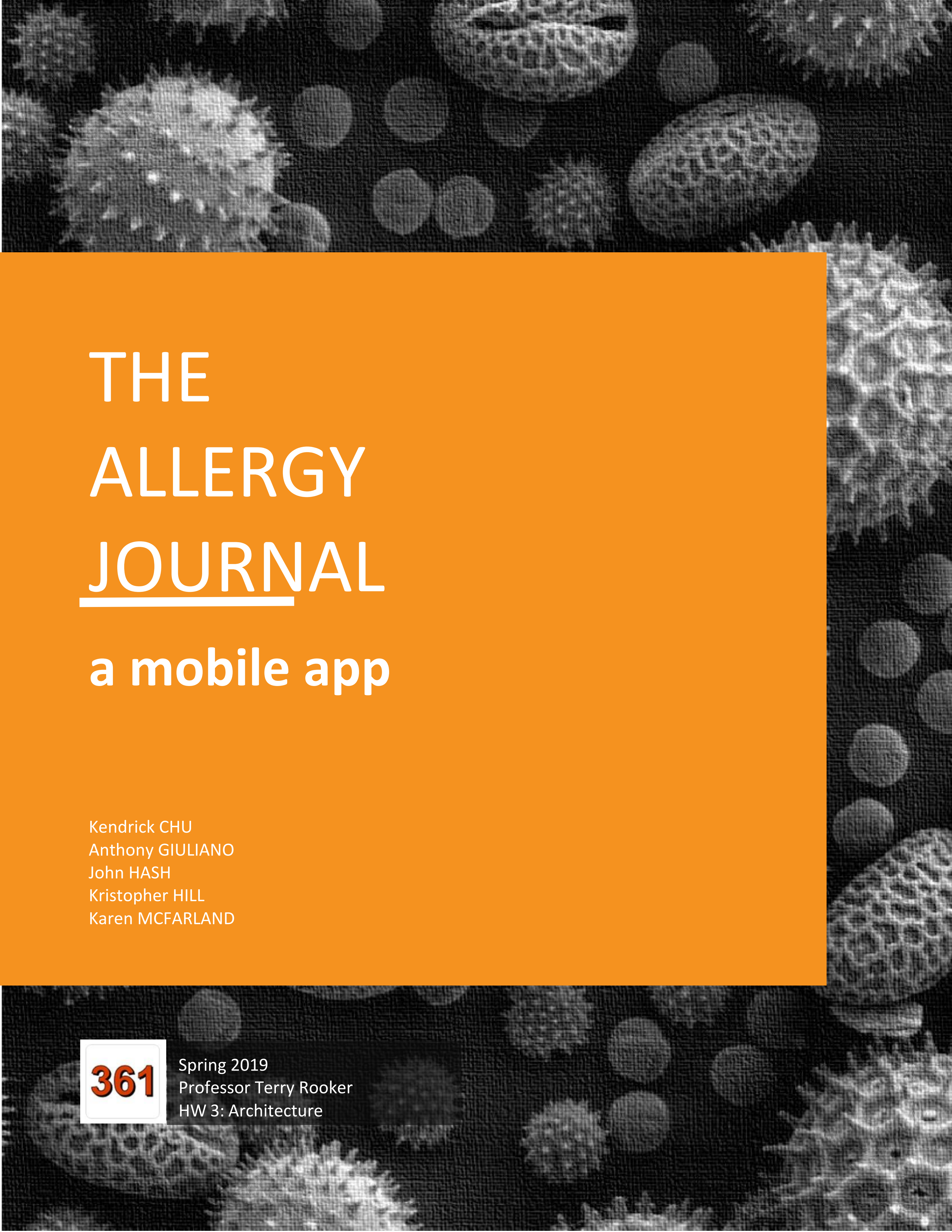

THE ALLERGY JOURNAL

a mobile app

Kendrick CHU
Anthony GIULIANO
John HASH
Kristopher HILL
Karen MCFARLAND

361
Spring 2019
Professor Terry Rooker
HW 3: Architecture

System Architectures

High-Level Architecture of Our System

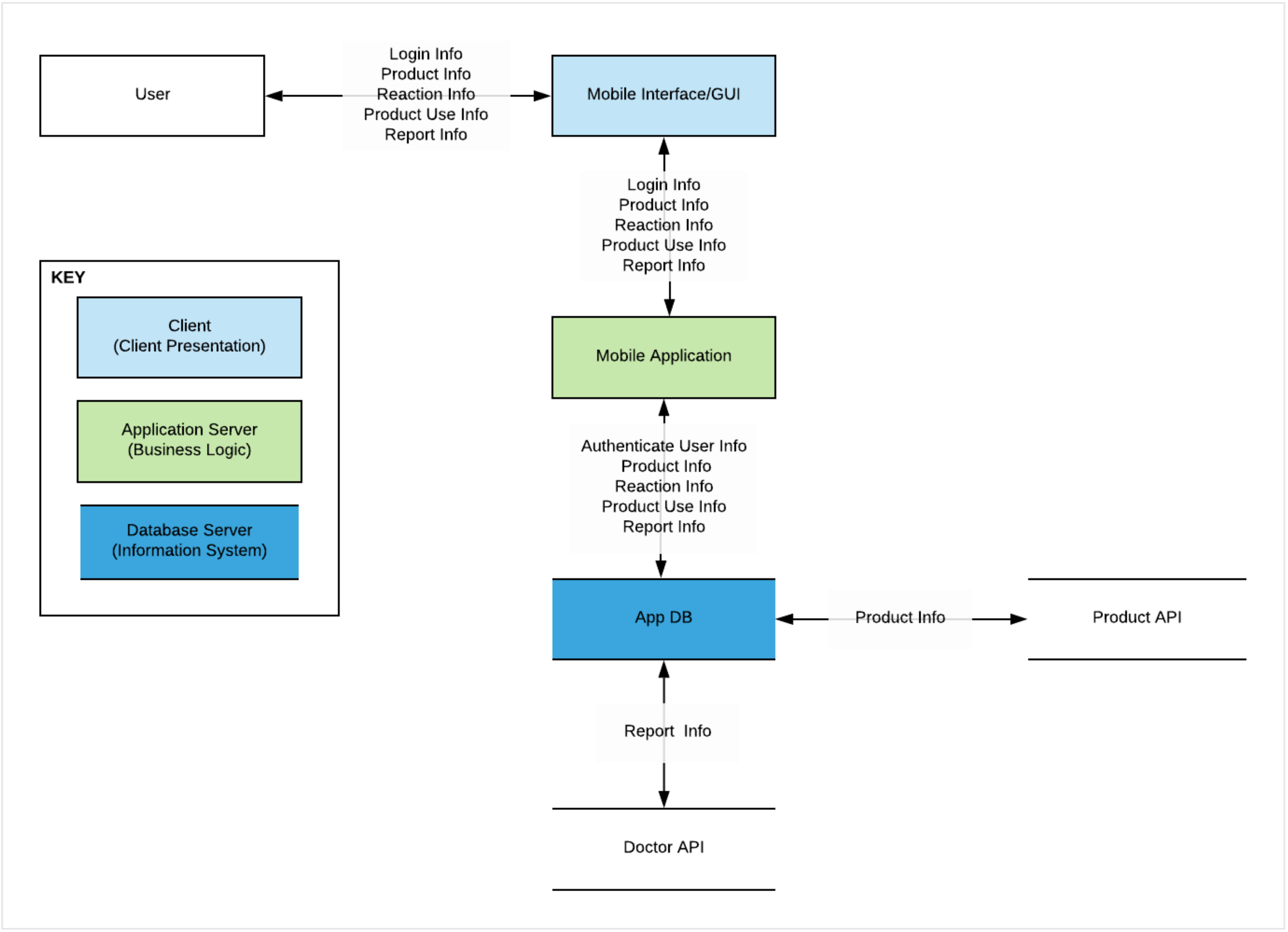

Figure 1. Level 0 Dataflow Diagram of *Client Server* Architecture

High-Level Architecture of Alternate System

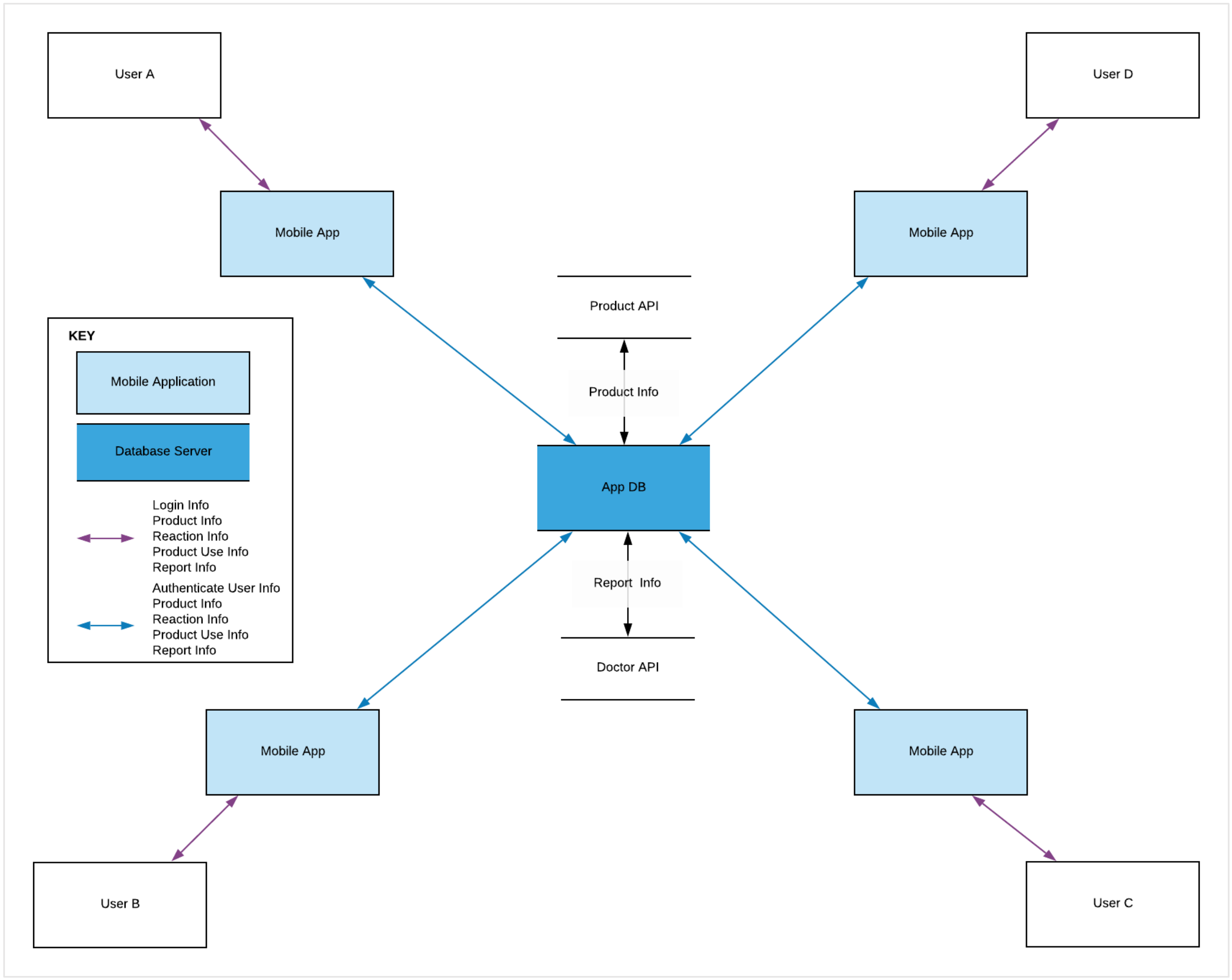

Figure 2. Level 0 Dataflow Diagram of *Repository* Architecture

Please note that we are aware that there are two primary styles of the repository architectural style. In order to have sharper contrasts, we chose the blackboard architecture over the database architecture for purposes of this assignment.

Key Quality Attributes

Usability

- Client-Server
    - System will operate like most other simple, mobile application interfaces. Internet connectivity constraints will be minimal because access is only needed when the user wants to push their data or receive a newly generated report.
- Repository
    - The repository architecture will likely not lead to any negative usability issues. The user will have to be informed that the nature of the application is slightly different than the client-server applications that they are used to. More privacy agreements will be needed. We may also need to further anonymize user data to ensure that information is not shared inappropriately across devices.

Integrity

- Client-Server
    - A user's anonymity will more likely be preserved in a client-server architecture, where the server does not initiate contact with clients or even "know" who they are. Users, operating through the client, decide when to send requests to the server for information. Overall, there is less, but more controlled, contact between the clients and the server/database in this architecture than in the repository architecture. As such, the opportunity for conflicts in the data are lessened.
- Repository
    - Data security is a more pressing issue with the repository architecture. Because of the blackboard type of repository, the clients will be in constant contact with the repository, and therefore, other clients as well. Extra caution is needed to protect anonymity of the users because the devices on the client side will be "checking out" data from the repository to do computational work. Under this architecture, data consistency is a primary concern for when clients retrieve data from the data store and then update data before sending it back to avoid conflicts.

Reliability

- Client-Server
    - With nearly all computation being carried out on the server, the client-server model may be slightly less reliable than the repository architecture. A fault could lead to error propagation that would affect all users then need to be updated centrally.
- Repository
    - Consistency will be a major issue with the repository architecture type. This architecture distributes computations across many devices. It might be expected that even though slight errors could occur, by having many devices working on improving the data store in a blackboard fashion, that the collective wisdom of the system may overcome some errors.

Efficiency

- Client-Server
  - The client-server architecture would take a lot of the burden off of the user's device by leaving nearly all computation to the server. As a result, performance would be nearly uniform for all users. Storage requirements on the client end are minimal. Furthermore, keeping copies of the data consistent is improved by doing computational work on the server and updating the data on the server.
- Repository
  - A repository architecture would be much more of a burden on the user's device on the client side because clients are taking data from the store, performing a computation, updating the data, and sending it back. User devices would need to be able to store (if only briefly) potentially large amounts of data from the data store while doing computational work. Network burden would also increase for both client and server.

Failure Modes

Failure Mode 1: Allergen report Not Available

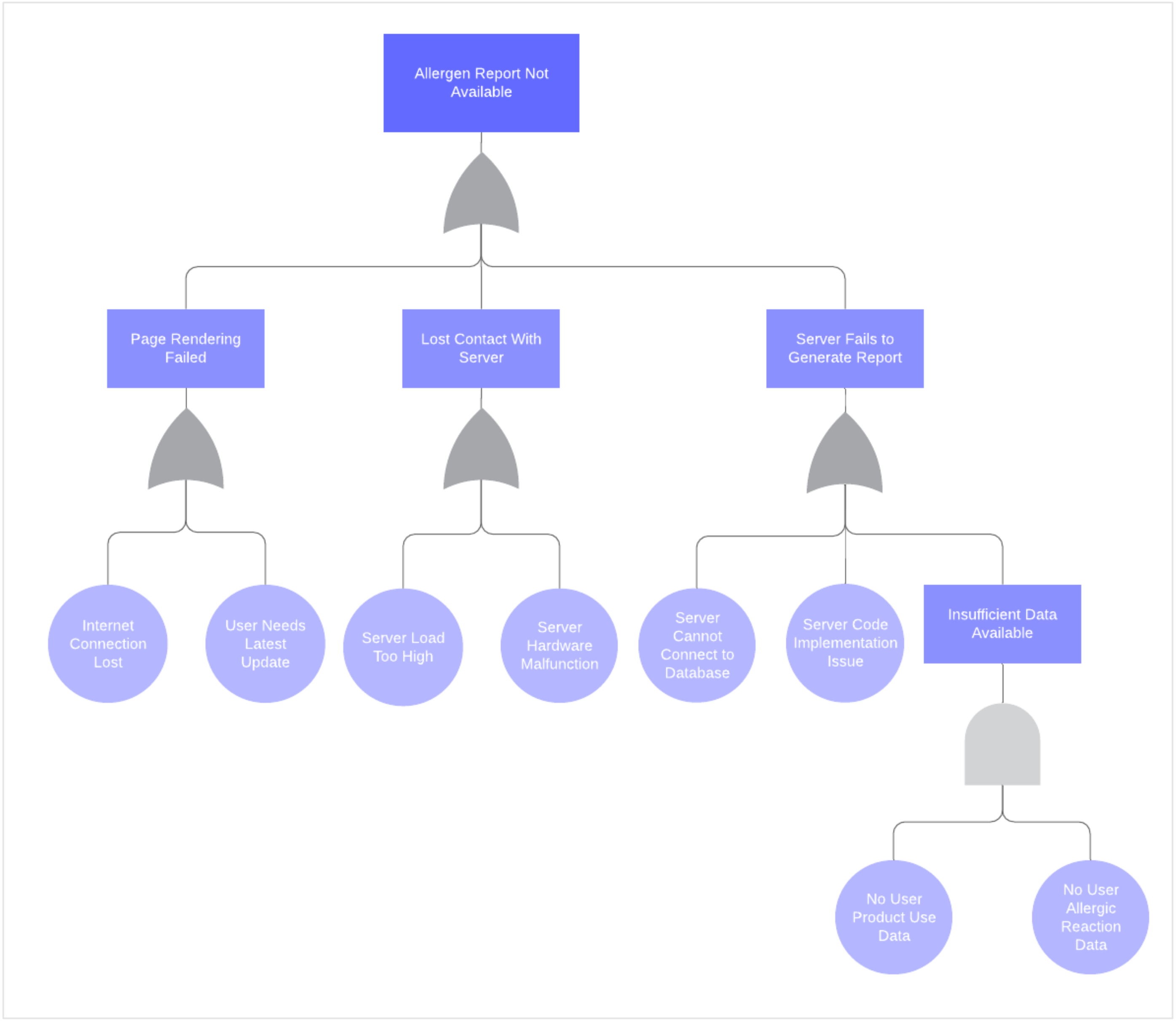
Allergen Report Not Available
Page Rendering Failed
Lost Contact With Server
Server Fails to Generate Report
Internet Connection Lost
User Needs Latest Update
Server Load Too High
Server Hardware Malfunction
Server Cannot Connect to Database
Server Code Implementation Issue
Insufficient Data Available
No User Product Use Data
No User Allergic Reaction Data

Figure 3. Failure Mode 1: Allergen report Not Available

The client-server architecture is more likely to fail. Under this architecture, the server load will be high. Furthermore, the most likely reason that a user will be unable to access new allergen reports will be because of high server traffic or a server malfunction, either physically or through a fault in the code on the server. However, in the repository architecture, the server/data store will only be handling a small percentage of the overall computation being done. As such, the burden will be much lower. Also, under the repository blackboard style, the user will have large amounts of data stored on their devices, and barring page rendering issues on their end, some report information may still be available.

Failure Mode 2: User Privacy Compromised

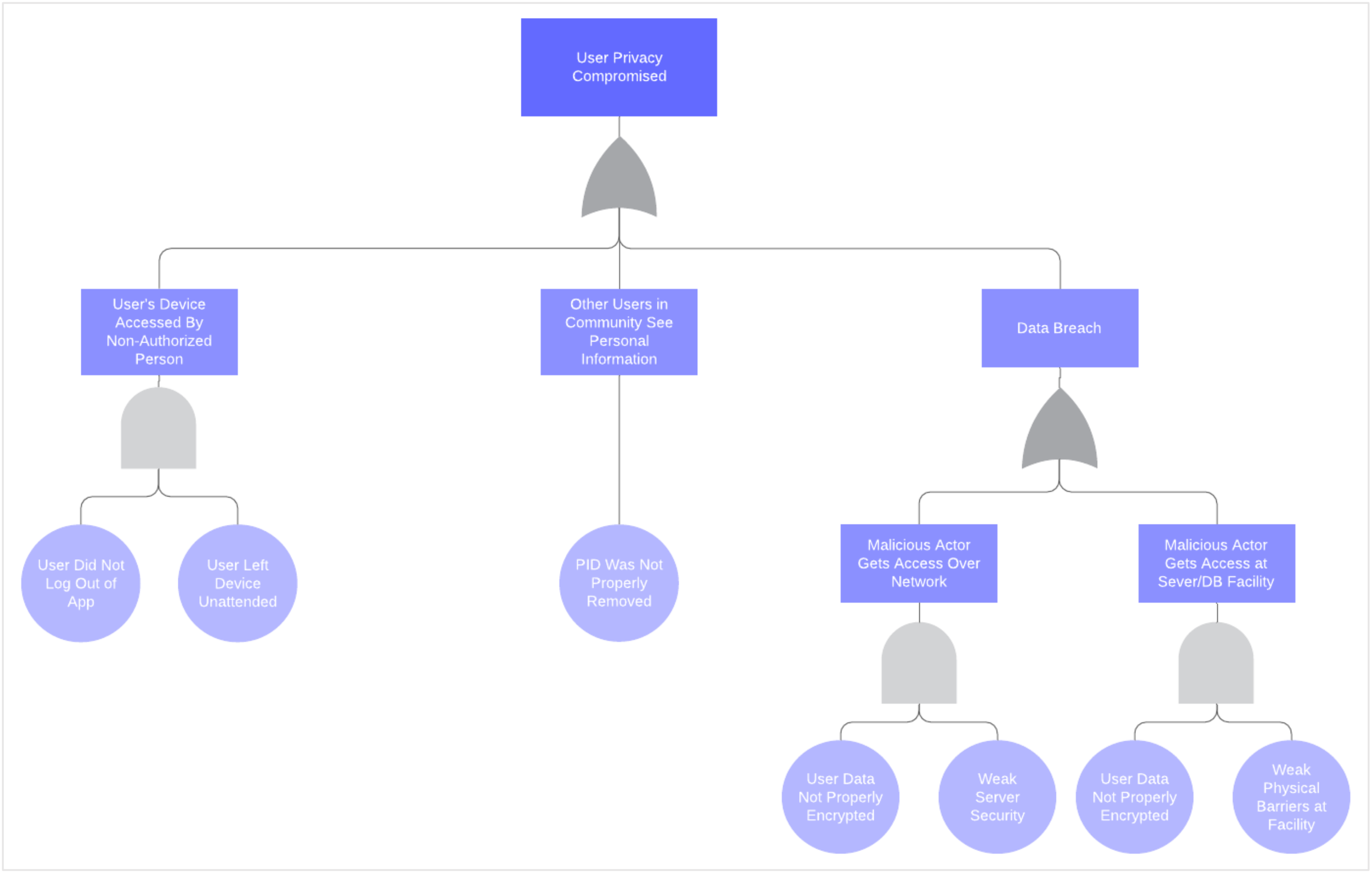
User Privacy Compromised
User's Device Accessed By Non-Authorized Person
Other Users in Community See Personal Information
Data Breach
User Did Not Log Out of App
User Left Device Unattended
PID Was Not Properly Removed
Malicious Actor Gets Access Over Network
Malicious Actor Gets Access at Sever/DB Facility
User Data Not Properly Encrypted
Weak Server Security
User Data Not Properly Encrypted
Weak Physical Barriers at Facility

Figure 4. Failure Mode 2: User Privacy Compromised

Regarding user privacy, the repository architecture is more likely to fail. Neither the client-server architecture or the repository architecture can prevent a user's failure to secure their own device. In the repository blackboard style, however, the user will have a significantly increased amount of data on their device. If personally identifiable data (PID) is not properly scrubbed before it is checked out by user device's, large amounts of PID could be compromised.

Architecture Decomposition

Based on our quality and failure assessment, we selected the client server architecture for further decomposition. The Level 1 decompositions for the Mobile Interface/GUI and Mobile Application are shown below.

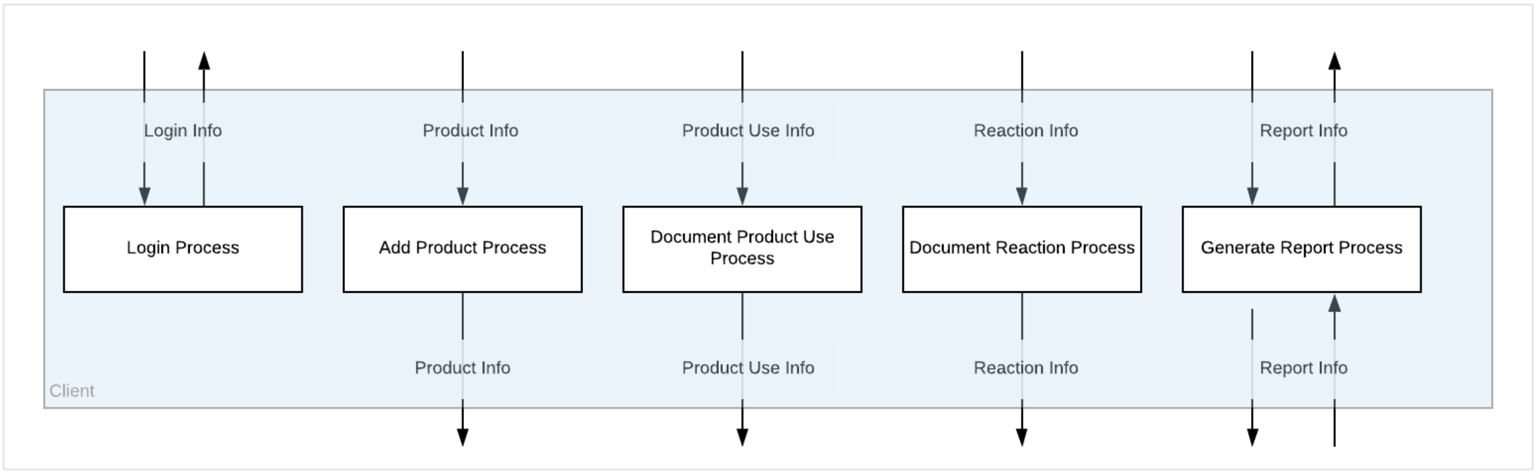

Figure 5. Level 1 Dataflow Diagram of Mobile Interface/GUI

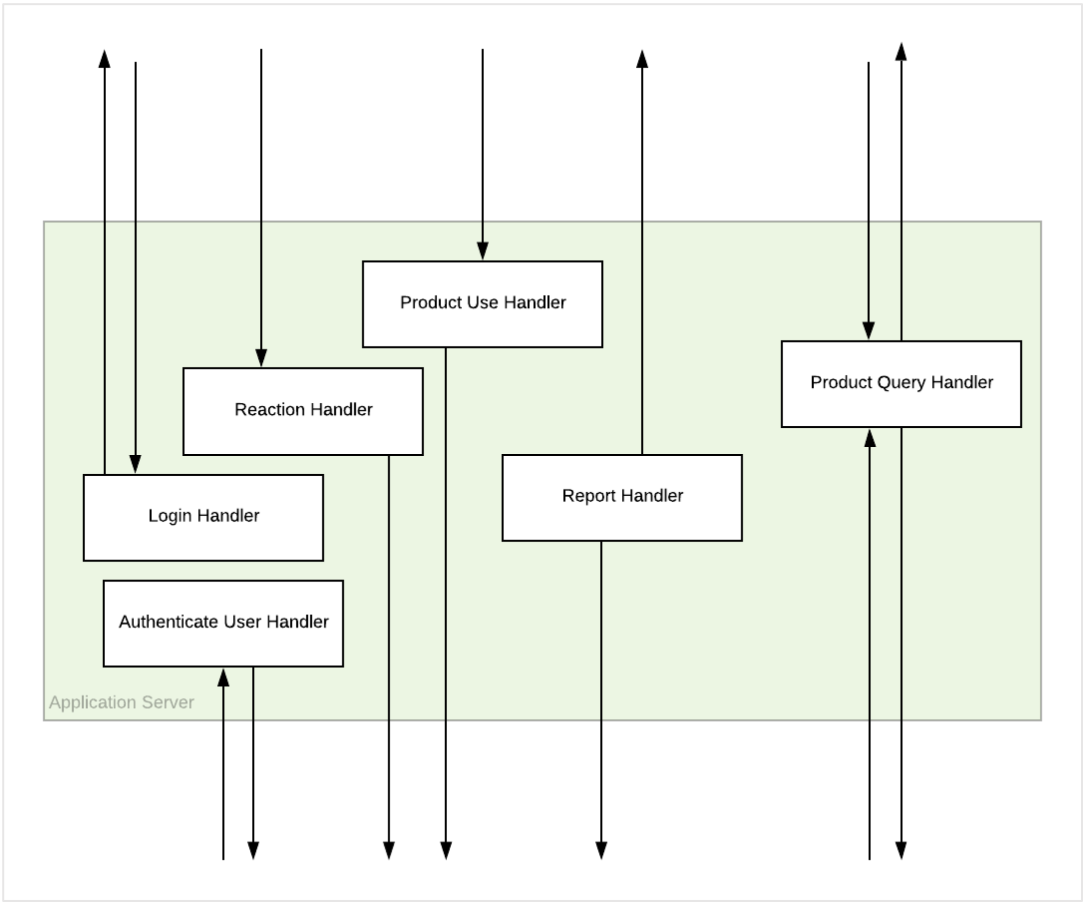

Figure 6. Level 1 Dataflow Diagram of Application Server

Validation By Use Case Walk Through

USE CASE 1: Entering a Product

- In the client-server architecture, the user will be authenticated by the server after entering their credentials in the app’s graphical user interface (GUI) and the client contacts the server.
- The user enters information through the application interface about the product they wish to add. The client sends the request to the server which runs a database query that checks if it is an acceptable product in the system.
- The server response includes the product match, which is viewed through the interface, and the user confirms that the product details are correct. If confirmed, another request is sent to the server for the product to be added to the user’s profile.
- If the user’s product does not match an approved product in the database, the user initiates an administrative approval request. The client sends the request to the server, where it waits for another client on an administrator's end to request the user petitions.
- If approved, the product is added to the product database and to the user profile.

USE CASE 2: Documenting Instance of Product Use

- In the client-server architecture, the user will be authenticated by the server after entering their credentials in the app’s graphical user interface and the client contacts the server.
- Through the GUI, the user navigates to the interface for documenting product use, where a new product instance is created.
- The client sends a request to the server to the get the list of products associated with user’s profile. The interface is populated with a list of the user’s products, from which the user chooses. Date and time are entered and added to the product use instance. The UI provides options for the user to select body regions applied, and after selected, the information is sent from the client to the server to be stored in the user’s profile database.

USE CASE 3: Documenting an Allergic Reaction

- In the client-server architecture, the user will be authenticated by the server after entering their credentials in the app’s graphical user interface (GUI) and the client contacts the server.
- Through the GUI, the user selects the option for documenting a new “Reaction Case”.
- The GUI then presents the user with a display to indicate the region of the body that had the reaction and the user selects the affected region of the body from a standardized list.
- After selecting the region, the GUI also presents the user with the option of including a photo of the reaction.
- The GUI also presents the user with a display to identify the symptoms that are being experienced. The client sends a request to the server to get a list of symptoms from which the user can choose. The user is presented with the option of seeing photos of other reactions in order to help determine the reaction being reported. If this option is selected, the client sends a request to the server to get a collection of images from which the user can choose. If the user selects one of the images, the corresponding symptom is saved locally.
- When information about the affected body region, user symptoms and any optional photos taken by the user have been collected, the user is asked to choose a date and time for the reaction being reported. The current date and time will be proposed as the default and the user has the option to change this information.

- The information that will be sent to the server is presented to the user for confirmation.
- After user confirmation, this collection of information on the user's reaction is sent from the client to the server to be stored in the user's profile database.

USE CASE 4: Allergens and Analytics Reports

- In the client-server architecture, the user will be authenticated by the server after entering their credentials in the app's graphical user interface and the client contacts the server.
- Through the GUI, the user selects the option for viewing the reports of allergens they've already logged.
- The app client then makes a call to the server that stores the database of information for this app. Specifically, it requests:
    - products from the user's product use table that are identified as giving users in the app's wider community allergic reactions, ordered by how many instances of allergic reactions those products have been recorded as having.
    - ingredients from the short list of products already requested that are identified as giving users in the app's wider community allergic reactions, ordered by how many instances of allergic reactions those been recorded as having.
    - products from the wider community of users that are identified as giving community users the same allergic reaction that was recorded by this specific user, ordered by number of instances of allergic reactions, limited to top ten.
    - products from the wider community of users that are identified as giving community users the same allergic reaction that was recorded by this specific user and also used by the user, ordered by number of instances of allergic reactions.
    - products from the wider community of users that are identified as having the same use as the product(s) recorded by this specific user in this instance but excluding the products in this instance themselves, ordered by number of users that are using the products.
- The app client takes the information in the response from the server and presents the information to the user in the GUI in lists labeled, respectively:
    - "Products Likely Causing Reactions"
    - "Ingredients Likely Causing Reactions"
    - "Products With Similar High Reaction Rates"
    - "Products You Use To Be Careful Of"
    - "Products You Might Want To Use Instead"
- The app client GUI has an option for user to click labeled, "Share Info with Community." If user clicks the button, the app GUI prompts user to select whether to also share report with an allergist/dermatologist, then the app client makes a call to the server to insert all non-personally-identifiable data related to this particular allergic reaction instance to the appropriate tables in the database:
    - product used
    - body part product was used on
    - symptoms of allergic reaction
    - (optionally, if taken) photo of allergic reaction
- Finally, user can click option in app GUI to go back to the main screen or can simply close out of the app.

Implications

The client-server architecture we have described here makes extensive use of the server side and leaves little for the client side other than mainly accepting data input from the user and rendering reports. As such, one potential downfall is that if the server cannot be contacted the application is almost non-functional but for a few exceptions, such as being able to access summary data about known allergens and the ability to view the most recent reports.

Listed here are some features that could be added to allow the user to still use the application without server connection. These suggestions are primarily based on adding more responsibility to the client side without diminishing the role of the server:

- Allow the user to send reports directly to the doctor from their device. This would bypass the need for the server to be active for the doctor to get health information, but at the same time, security issues would increase because most features that protect the user's PID are on the server. Furthermore, without the server as an intermediate, doctors and their hospital/clinic system may be unwilling to work directly with clients.
- Requests (such as allergen instances or product use instances) to the server could be queued up on the client end. When a connection is resumed, the requests could be automatically sent.
- A low-security authentication could be added to the client side to allow the user to login and queue up requests. Once connection with the server has been re-established, full authentication could be finished before requests were sent.

Usability Considerations

For each additional client-side feature that is added, we are putting more burden on the user to keep track of available features, and this could have a negative impact on the usability. We want the application to be as simple as possible. It is already difficult enough to incentivize people to keep records like- these, so we feel that in the interest of usability, keeping maximum burden on the server side is optimal.

Efficiency Considerations

This is an application for everyday users, who have smartphones with a wide range of capabilities. Some users could likely take on more computational and storage burden on the client side if they have the latest, most powerful devices. In the interest of serving the needs of as many people as possible, keeping the burden on the server side will not "bog down" personal devices and is more efficient for the users.

Reliability Considerations

The entire design of this application predicates on a user base of potentially millions of people, and a server failure results in loss of access for everyone. We plan on using a cloud solution to host our database server and with the increasing reliability of such systems, we feel that this is an acceptable risk. Furthermore, this application, though important, is not critical for users. Finally, the probability of server failure is much less than the probability of the failure of an individual user's device.

Final Considerations

This architecture review has helped the team identify gaps. Specifically, we were able to really think through and clarify which parts of the system the product API and doctor API would interact. As a consequence, we also realized that we needed an authentication process between the application server and the database server in addition to our login authentication.

Contributions of team members

	Kendrick CHU	Anthony GIULIANO	John HASH	Kristopher HILL	Karen MCFARLAND
High-level architecture DFD	X	X	X	X	X
Alternate high-level architecture DFD		X			X
Quality attributes discussion			X		X
Fault tree diagrams			X		
Fault tree explanations			X	X	X
Lower-level dataflow diagrams		X	X	X	X
Architecture validation			X	X	X
Implications			X		X
Stitch together document					X
Review final draft		X	X	X	X
Submit document					X