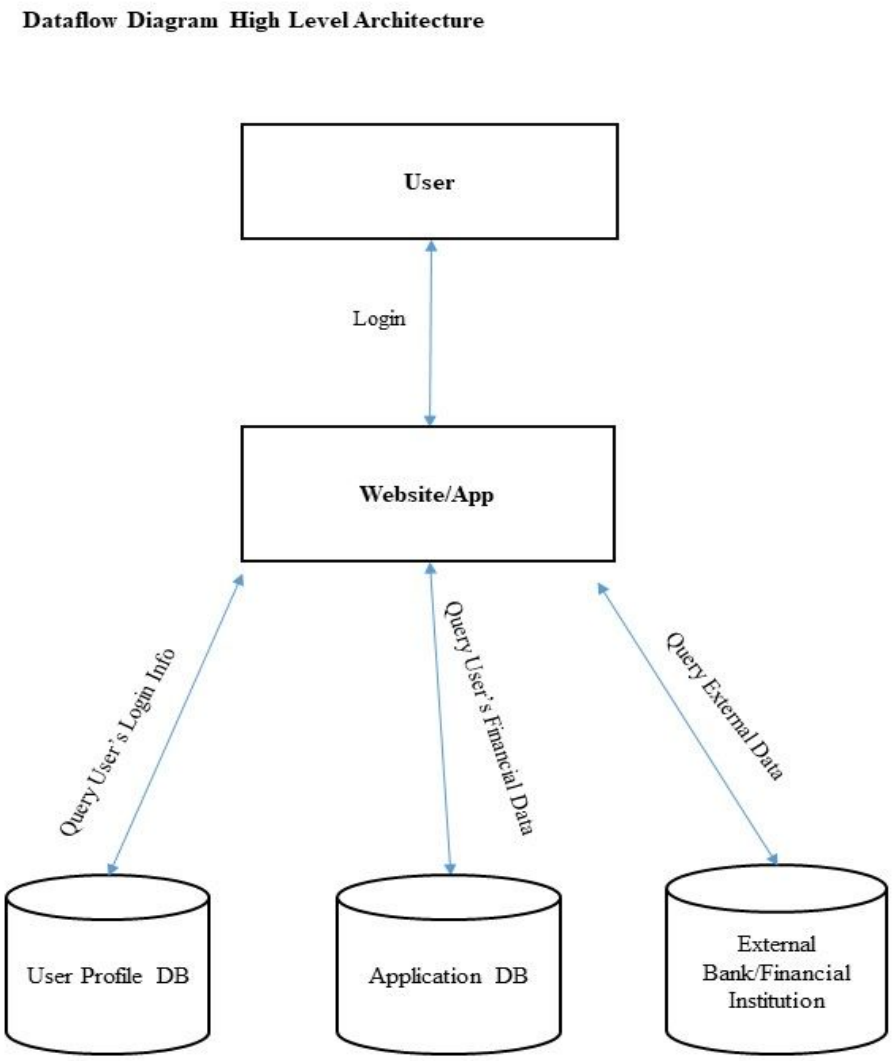
Homework 3

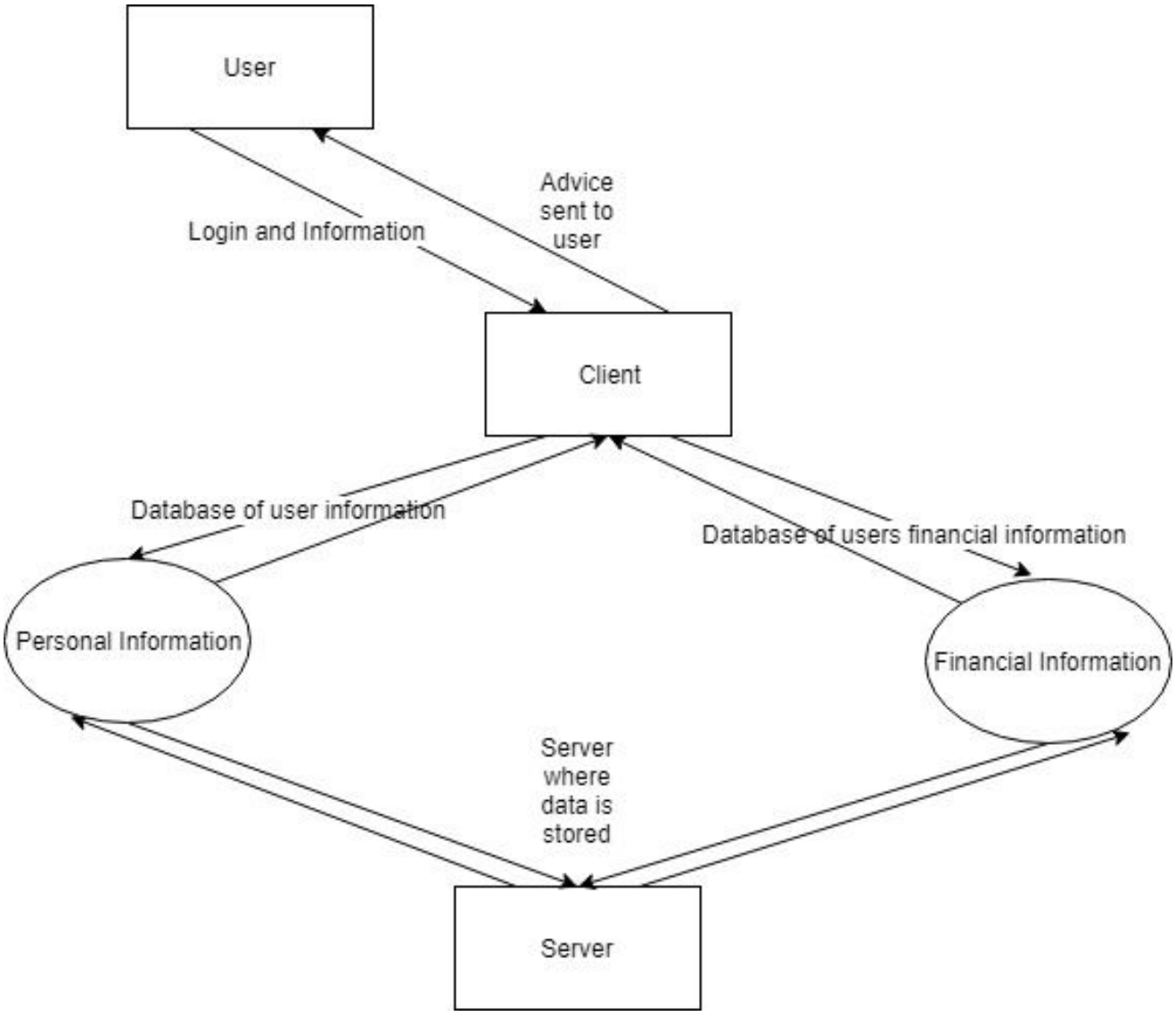
Group 10
Class: CS-361

I. Dataflow diagram: high-level architecture



II. Dataflow diagram: alternate high-level architecture

Client-Server high level architecture alternative.



III. Two failure modes & draw fault trees for each

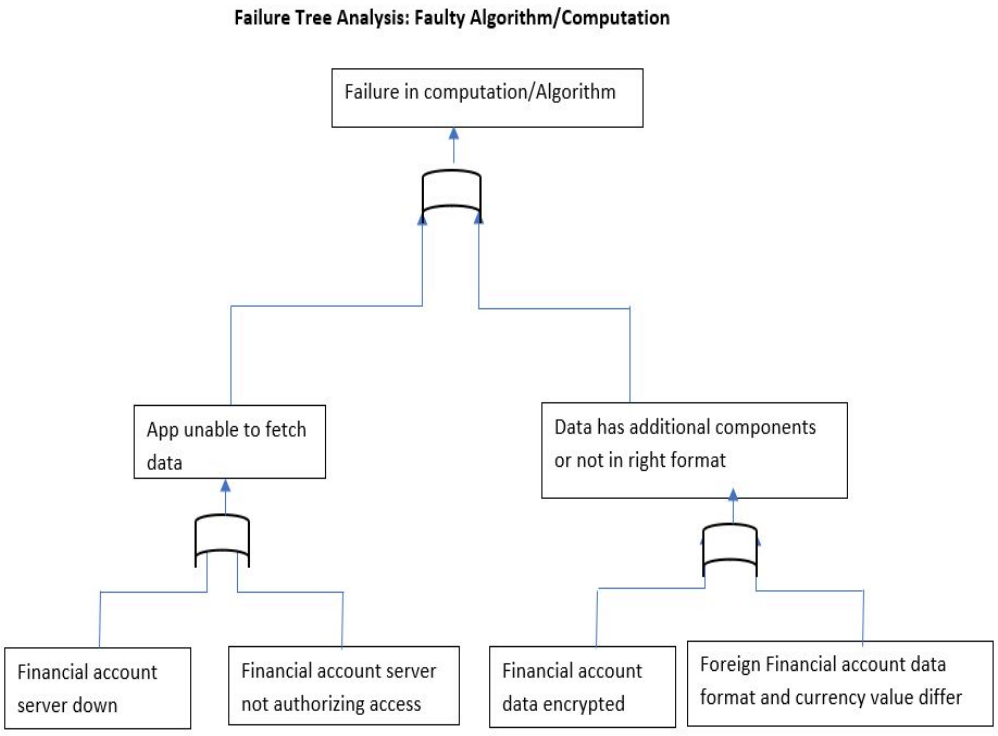
1. Faulty Algorithms/Computations: The algorithm or set of formula doesn't work for all the possible inputs.

There can be several reasons for failure in algorithm. The app might not be able to fetch data due to the server for the financial account being down or financial account might not allow the authentication via different app it might be the case the user can access account only thorough original website. Without getting data our app algorithms or computation cannot be carried out.

Also,since we are getting financial accounts from different sources for each user.

If the user has foreign financial accounts the data format will differ also the currency conversion factor needs to be taken into account to get accurate results.

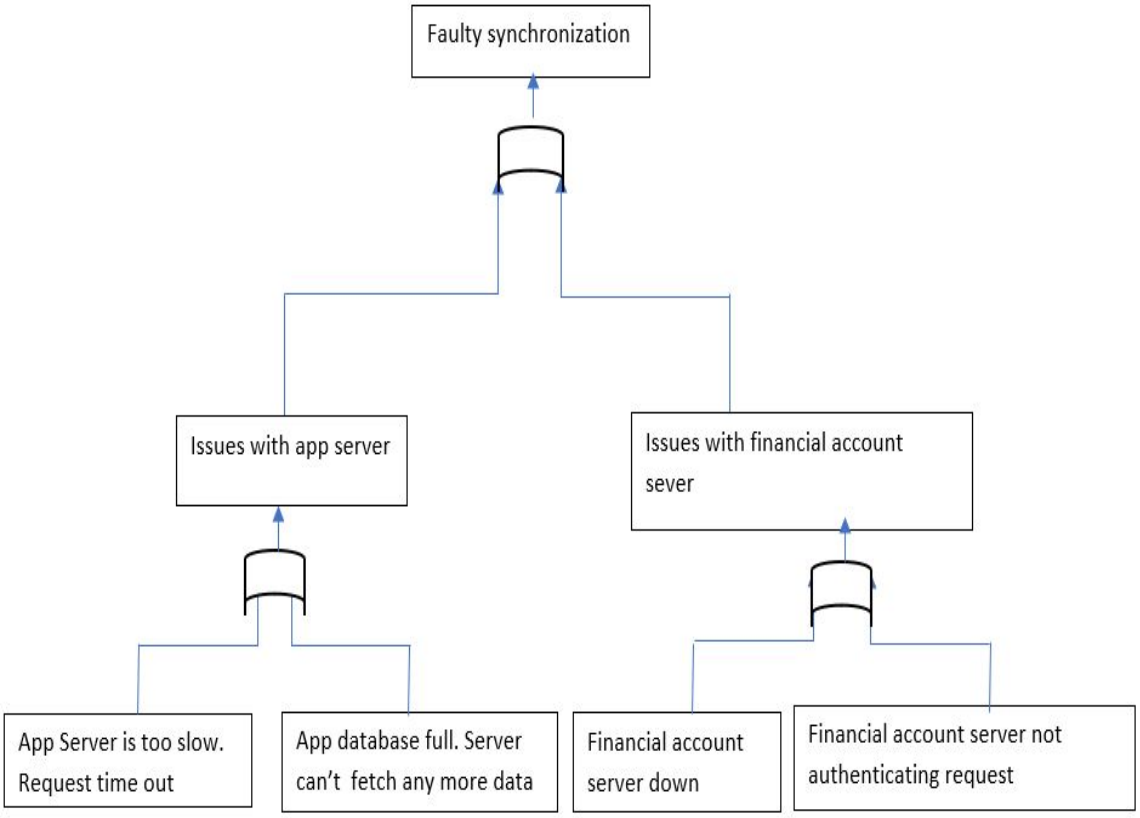
While extracting data some fields required might return undefined values based on how the information is stored in that website. This can occur due to encryption of financial data into the accounts.It can cause unwanted behaviour in our app.



2. Faulty synchronization: The parts of the system are not in sync with each other . The app needs to fetch information in real time. The app can fail to get

information from the financial accounts which can cause delay in getting information regarding pending payments.There can be some issues hindering the synchronization in the app. Issues with app server or issues with financial account server. In case of problem with app server. App server might be slow. It can result in request getting timed out. App's database might be full there is not enough space to store information.Also, the financial account server can also cause problems like financial account server might be down or the server might not authenticate request. Thus, if the spending charts are generated for the user after computing the financial transactions. If all the financial data is not transferred for any reasons the financial charts would be incorrect and would provide an incorrect analysis for the user spending.

Failure Tree Analysis: Faulty synchronization



Failure Mode: Customer's personal financial data is compromised

Some of the most high-profile software security failures (so to speak) that have been profiled in the media over the recent past has been the compromise of private user's data. Entities in the news have included Facebook, Equifax, Google Plus, Orbitz, British Airways, Uber, and many, many more.

These compromises (in essence) result from either poor security (as in the case of Equifax), or an active attack (as was the case in British Airways). Some of the largest events (involving hundreds of millions to trilions of records) were the result of the service being "hacked" by an active, malicious user. However, simple poor security measures (as in the case of the 2018 Facebook data breach) can also impact millions of people.

Given that this application is intended to collect, analyze, and store end-user financial information, maintaining security for this data must be of the highest priority.

In this fault-tree analysis, we are going to focus not on the case of a single-user compromise, but instead on wider-system data breach. These can be divided into two types of security failures: one resulting from an active attack, and one resulting from a system-wide (passive) failure.

One active-attack example is that of a Phishing attack. For such an attack to be successful, not only must an active agent be attempting to gain entry to the system's data, but the user (and/or staff) must be uninformed or unaware of this type of attack and/or what sort of thing to look for. This is (primarily) and human-and-system security issue, and is (primarily) mitigated with both education and other methods such as two-factor authentication and login confirmation.

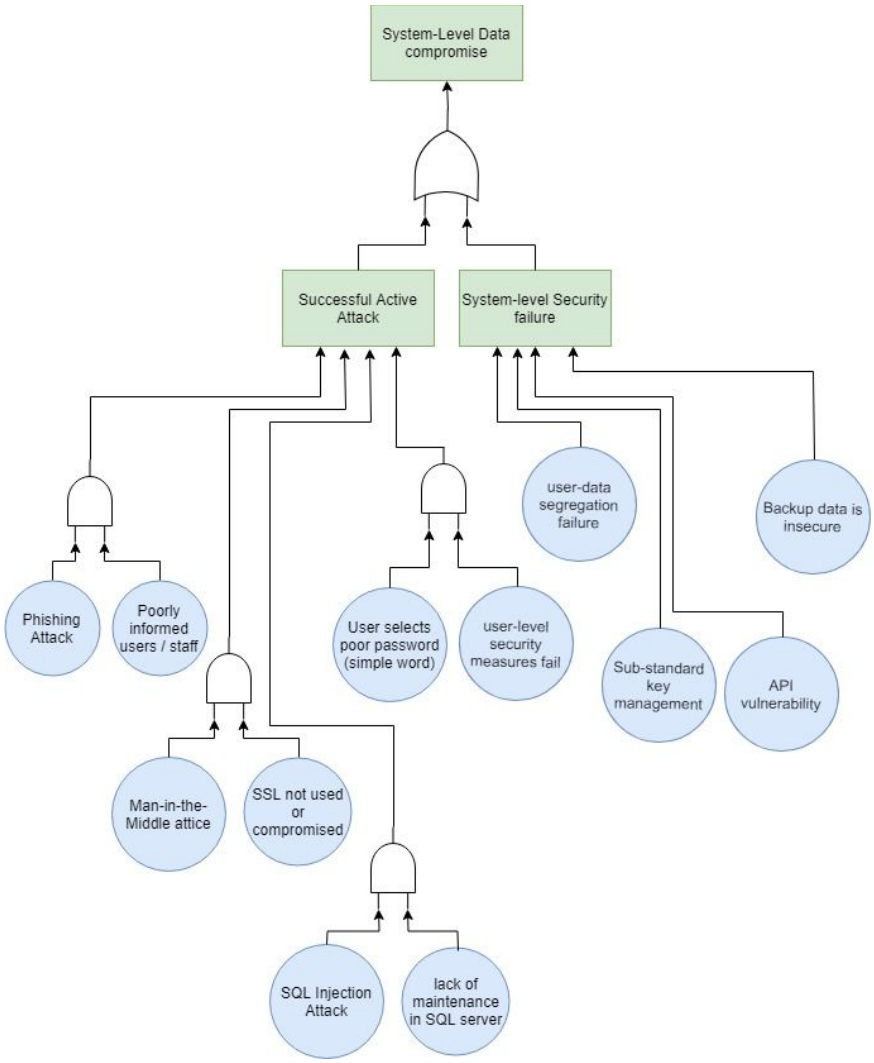
An example of a system-level (passive) failure would be one where the user-data is not properly segregated (in a secure manner). This would (potentially) allow one

user to access another user's data - either purposefully or accidentally - either scenario must be considered.

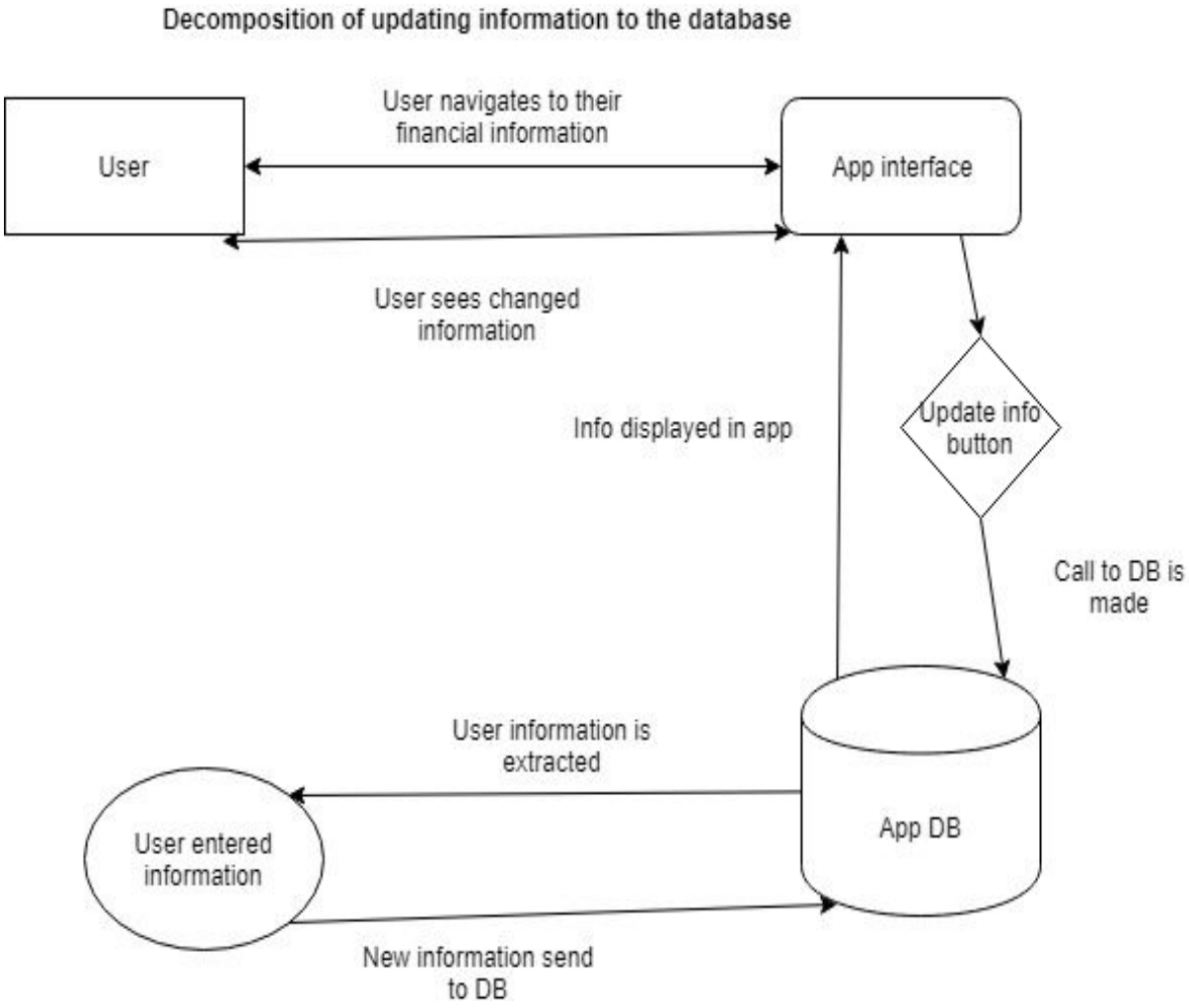
Solving each of these types of events will require further analysis of each scenario, and what infrastructure needs to be put in place to mitigate each.

The diagram below is not exhaustive, but should be viewed more as a work-in-progress. As methods for (both active and passive) system-data breaching become more complex, more complex scenarios must be identified and considered.

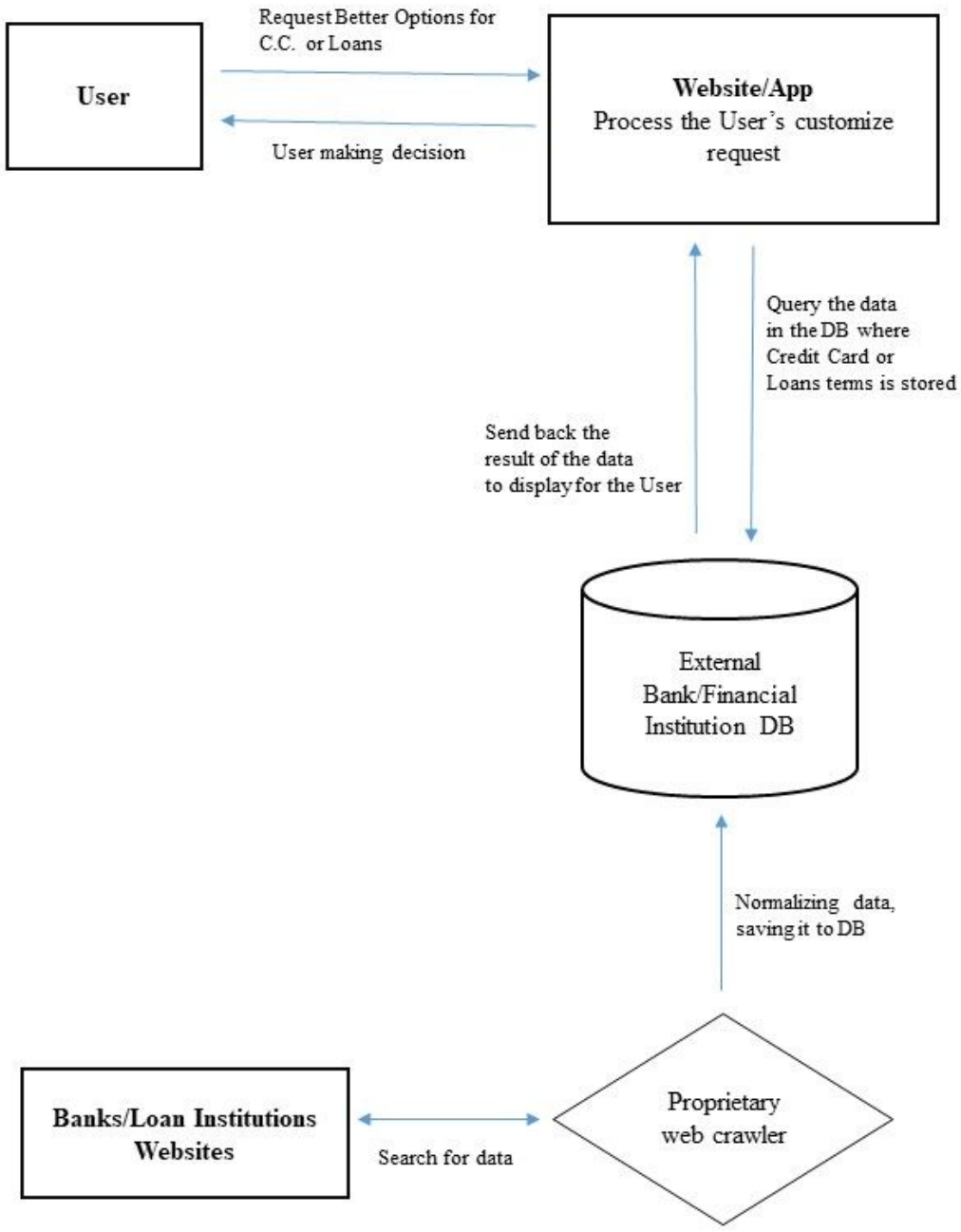
Failure Tree Analysis for a System-Level Data compromise:



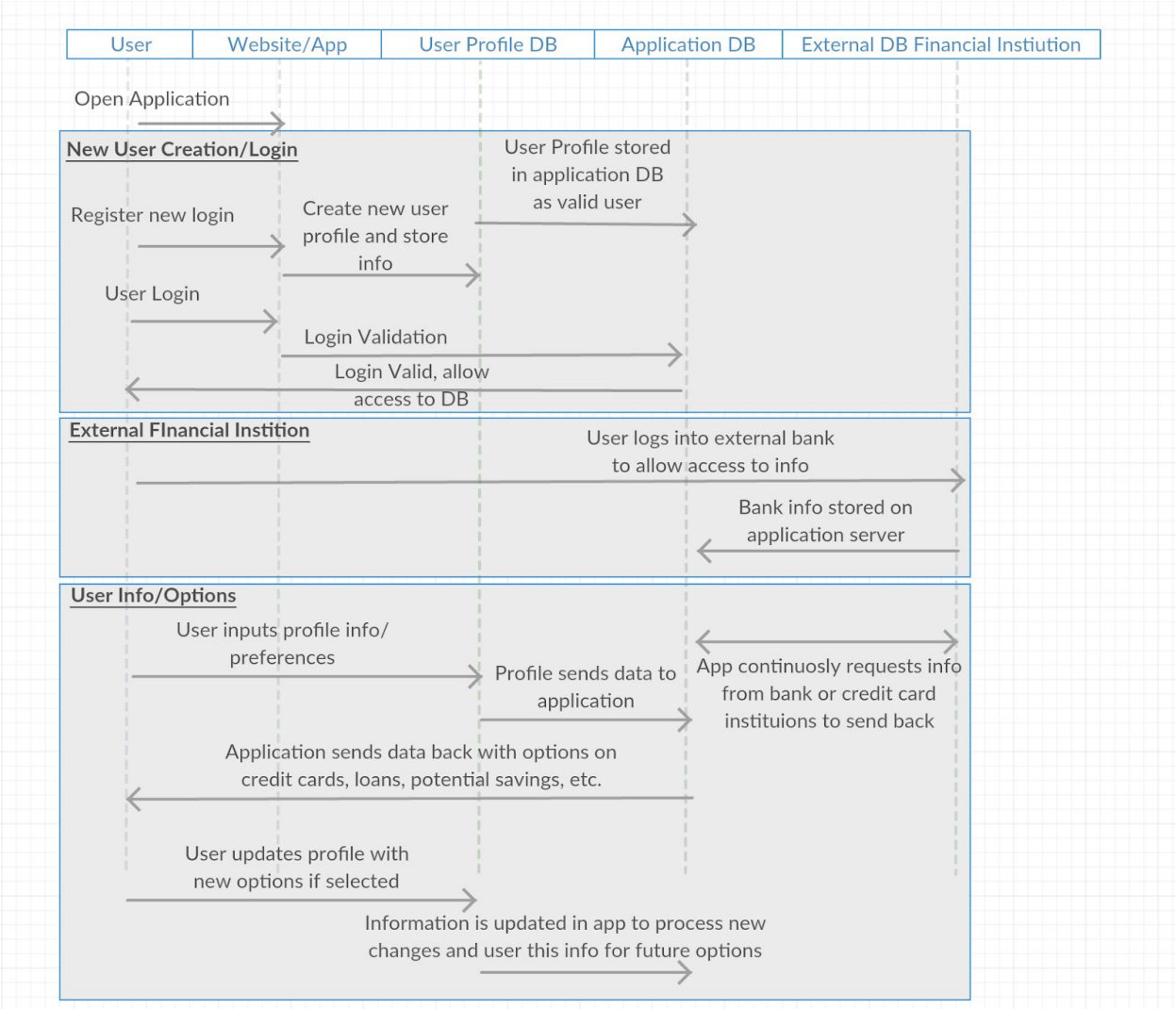
IV. One architecture - further decomposition



Low level dataflow diagram of "Better Options" feature



V. Walk through the use cases (validate architecture)



Between all use cases, there are similar steps in the process. Users will create a login and this information will be validated to the application DB which will then be stored as within the User DB. After creation, the user will log into the application DB which in turn allow access to the profile. The profile will then be filled with information that is stored within the User DB. If external bank information is wanted in processing accurate and up to date account info, then the user will be prompted to login into the financial institution's server to allow Check Please to access it. All this info will be sent to the application DB to allow the processing of options based off the user's intended use purpose of finding better

solutions to debt or options for financial stability. As the application processes the potential options, it will be relayed back to the user and have them update their choices so that the application can continue to update for other future options.

VI. Explain the implications

The results of our validation efforts have revealed some real concerns about security of the user's personal data, as well as the connections to the user's financial institutions. Building and maintaining trust with the application's user-base is absolutely critical to the success of our product.

With this knowledge, we would revise our architecture to include more secure user-login methods. We would most certainly enable two-factor authentication, email verification, and possibly other account verification methods.

We may also want to consider exactly how much customer's financial information is stored locally on the user's device, as well as how much is stored on our product's servers. It could be considered much more secure to simply have a pointer to a location (remotely, with authentication) where financial information is stored.

While these modifications would improve user data security, it is very possible that these modifications would result in performance degradation. If nothing else, server-to-server communication becomes the critical element that determine overall performance. This is in stark contrast to the alternative of keeping all user-data local; all required data is immediately available but also more vulnerable. These options need to be considered and balanced carefully.

VII. Team Member Contributions:

- Aseem Prashar: Modes of Failure (faulty computations / synchronization)
- Forrest Allen: Dataflow Diagram #2, further decomp.
- Thach Vo: Dataflow Diagram #1, further decomp.
  Khoa Phan: Use cases walk-through
- Michael Volz: Modes of Failure (data compromise), Explain the implications