Presentation order

1. Introduce Team/Lead into app – Lauren
2. Login – Ahmad
3. Design – Hishaam
4. Main Window – Hakim
5. Update User – Abad
6. Create User – Lauren
7. API Connection – Ali

Slide/Pictures

1. Intro- headshot
2. Login
3. Different UI (Light/Dark)
4. Main Window/Main Window with Selected user
5. Update User Form/User Form showing pw unlock
6. Create User Form/User Form showing privilege combo box
7. Something for API?? Or wrap up page

**Use the sub-topics in each featured topic to write a script for the presentation**

INTRO (**Lauren**)

“Hello everyone, my name is Lauren Rivier. I am team lead for App Dev. Our team, which consists of Ali, Abad, Ahmed, Hishaam, and Hakim, are creating an app for the Pet Best company. This application will be responsible for managing employees, customers, services and appointments, schedules, reports, everything that’s important behind the scenes to run a successful business. So, we are excited to show you our progress so far. I’m now going to hand it off to Ahmad who will talk a little bit about the Login form, which is the first point of contact that the user has with the application.”

LOGIN (**Ahmad**)

* EXISTING USER ON SERVER
* ERROR HANDLING
* EXCLUSIVELY OWNER/EMPLOYEE
* WATERMARK
* FEATURING LOGO

“Hello everyone, my name is Ahmad Khazal, I will be discussing the login portion of the application. If there is an existing user on the server, they will be able to login into our application. Our login form consists of a username and password. When these credentials have met, the user can login and view the application. If an admin logs in, they can manage to check for errors on our application and make some changes. Owner and the employees will be able to login and view customer requests. The application will only to be accessed by employees or admins.”

DESIGN (**Hishaam**)

* COLOR PALLETTES (DARK AND LIGHT)
* FONTS

“Hey, my name is Hishaam and I’ll be walking you thru the early stages of our application’s design as well as briefly discussing the style guide. As we began the development of our application, we knew we wanted a clean, minimalistic design that prioritizes usability w/o sacrificing functionality. Having decided on C# as our programming language of choice, it was only fitting that we build our application using WPF, a powerful graphical sub-system known for its modern user interface. ”

MAIN WINDOW (**Hakim**)

* LIST BOX
* UPDATE USER + CREATE USER BUTTONS
* USER MUST BE SELECTED TO UPDATE

“Hi, I’m Hakim. After a verified user logs in, they are taken to their main page depending on their privileges which are passed through the API for each user. Admins are taken to a page where they are greeted with a welcoming message, followed by a list of all the users. Under that, the admin can choose from 3 buttons, Create User, Logout, and Update User. The Update user is disabled until the admin chooses a user from the list box above. When a user IS selected, the button becomes usable and upon pressing it, the admin is taken to the Update User page. Unlike the Update User, the Create user button does not require a selected user. It simply takes you to the Create User page. Lastly there is a Logout button that will terminate the user’s connection and take them back to the login page. When an employee logs in, they are taken to a similar home page with a welcome message and a logout button that also terminates their connection and sends them back to the login page. Customers, however, are not permitted to use the desktop application and are greeted with a message box that notifies them of that.”

UPDATE USER (**Abad**)

* FIELDS REQUIRED
* PASSWORD CHECK BOX
* FILLS WITH SELECTED USER
* UPON SAVE, USER IS UPDATED AND LISTBOX IS UPDATED

“Hello everyone, my name is Abad Meza and I’ll be talking about the Update User feature of our application. When a verified admin or employee is logged in, they can click a name from the users’ listbox and the “Update User” button will be available to update a profile. Clicking the button will open the UserProfile form with all the fields needed to change information. Fields include textboxes for First Name, Last Name, Email and Username. The form also has a Privileges drop box with the 3 different credentials that a user can be granted. The 3 credentials are Admin, Employee, or Customer. There is also a checkbox next to the password field. If it is checked, it will allow the user to also change the password. The before mentioned fields are all filled with information entered when first creating the profile. If the verified user needs to update any of the fields with new information, the user simply selects a field and edits the text. When the user is content with any of the changes they made, all the user must do is press save. Instantly, the profile is updated with the new changes in the users’ list box. Close the “Update User” form and you’re back at the main form ready for the next action.”

CREATE USER (**Lauren**)

* SAME FORM UTILIZED AS UPDATE USER
* FORM WILL BE EMPTY
* FIELDS REQUIRED
* UPON CREATE, USER IS CREATED AND UPDATED IN LIST BOX

“Similar to the Update User button, the Create User button will take you to the same User Profile form. We wanted the user to have the ability to create not only employee and admin accounts, but client accounts as well. Not everyone has access to the internet or computer or are tech savvy, so we wanted to keep this in mind. So, this time around the form will be blank, forcing the user to fill in all fields that are required. Once the user hits create, the user list box will be updated, and a user will be created. We talked about how our application can login, create user, and update user, but we didn’t really go in depth with how that all works. Ali will be able to explain a little bit more about how our application communicates with the API’s.”

API (**Ali**)

* REST APIs
* JSON
* POST
* ASYNC/AWAIT

“Hello, I’m Ali Hazime and I’ll be talking about the process we used to connect to the API that backend set-up. So, let’s start off with the big question, what is an API? An API, or Application Programming Interface is a very broad term that can cover a lot of things. In fact, we could have an entire presentation on what an API is alone. So, for the sake of this presentation, It’s best if you think about a general API as a tool that allows two different software to communicate with each other. The specific type of API we had to work with is called a REST API, now this is a bit more specific so it’s easier to explain. A REST API is basically like a function but attached to a web server. You pass it arguments, and if those arguments agree with its parameters, it passes back data. We used the POST method to pass information because this app will have a constant flow of sensitive data moving through it, so we don’t want any leaks. You might be wondering, if REST APIs and functions are so similar, what’s the point in using a REST API. Well the key difference is that a REST API is much more accessible for everybody. Given the nature of this project, we have multiple teams of people all pulling from a public collection of data. Using a REST API makes it so that everybody is accessing the data through one channel, instead of us having to connect to the database directly and use our own private functions. Now we used the REST API to get user information to be used in our application, so logically, the next thing we need to do is store all that user information inside of an object. That would be really easy, if the API returned its data in C# notation, which it does not. The API returns all its data in JavaScript Object Notation, also known as JSON. So, the question becomes, how do we take JSON values and store them in a C# object. To put it simply, we used a package. A package that contains a function that decrypts JSON values into C# values. Then we set our objects equal to the decrypted JSON and we had our spiffy new users. The last thing I want to talk about is Async. I’m not going to get into all the details of how we implemented Async, just what it is and why we used it, because it’s an integral part of our UI. If you’re familiar with multi-threading in Java, this is familiar territory. Usually when you have a program, it works a little bit like this. You have a set of processes that need to run, and they all run one after another. Async, or Asynchronous programming, allows processes to run simultaneously. Let's say I have four processes, A, B, C, and D. Synchronous programming would mean that in order for B to execute, A must finish, so on and so forth. However, with Async, we can say hey all of you run at the same time. If every process takes 2 seconds, regular programming would take 8 seconds where Async would take about 3 at the cost of a bit more processing power. There are two major reasons we’re using Async in our program. The first is that we don’t know what kind of processes we’re going to have to implement in the future. There might be some long, arduous task that needs to be completed every time the app is launched. Using Async prepares us for that event if it ever needs to happen. The second is that it gives more power to the user over the UI. Usually in a visual application, when a process starts, the whole app freezes and waits for the process to finish. With Async, the UI won’t freeze because the processes are independent. This gives the user the ability to multitask and run different processes at once if need be, instead of having to wait for one thing to finish and be stuck the whole time. Now let’s step away from UI, and focus more on the power behind it, with the Backend team”