Preparation:

Nicholas: prepares application on device
Tyler: prepares slides and loads up project on simulator

Slide 1:

Introductions
Nicholas: “Hey I’m Nick”
Tyler: “Hey, I’m Tyler and our project is Indesyn”

Slide 2:

Tyler: “If any of you remember, our project deals with the process of connecting designers with consumers. The problem at hand is that finding an interior designer is not streamlined, making it hard for people to get the help they need. Furthermore, not many platforms exist where connecting the two is a smooth experience that also brings together a variety of interior designers from all over the world.

Nicholas: “As far as what we saw this app accomplishing, we envisioned our application being a platform for users to find an interior designer, while on the other hand, establishing a hub for designers to connect with users and display their work.

Tyler: “Now moving on to our product definition, ultimately, the sole purpose of this app was so people could revitalize the spaces that they live in. Many people do not have this design eye we have talked about so much and allowing designers to step in and utilize their trained eye on part of the user makes this app unique.

Nicholas: “Our platform will accomplish this by allowing the designer to display their past work while also allowing efficient communication between user (or NID) and designer (or ID).

Tyler transitions to the demo slide

And now we will move to our demo

Slide 3, Demo:

Tyler: **Launches app**

Nicholas: “So this is our sign-in screen. You can see a Google sign-in button at the middle of the screen which allows us to sign into our app using google credentials. Tyler will now click the button and sign in with his login info. After we sign in, we are brought

to our profile set up screen. The text input fields are auto filled using information from the user info object we receive from signing in with Google. For this demo, we will demonstrate two example cases: first, an NID signing up and then an ID signing up. In both cases, after choosing their desired role, they would click continue upon which they will be brought to the correct screen So, now Tyler will choose ‘user’ and click continue.

** Tyler clicks continue as a User**

Tyler:
“As you can see, whenever a **user** signs up for the app successfully they will be brought to the page that contains all the active designers on the app; we can scroll through a list of them now, which is implemented through the use of a FlatList.

**Explain FlatList **

Designers are pulled randomly from a backend call to the database. From that call we can grab all the associated data pertaining to each designer, such as their names, and profile pictures. And now we will show you how connecting with a designer works.

Tyler selects random designer

When Nicholas mentions icons on the ID’s profile, click them as he mentions them

Nicholas: “We are now brought to the designers page. You can see their chosen profile image, their name in the header, a connect button to the right of that which, on press, will bring the NID to the request page, a short bio provided by the designer, a briefcase icon which will display a scrollable containing designer chosen images, and a comment icon which will bring up designer reviews. If we click the first review, a modal pops up expanding the review further; you have the ability to scroll in the modal in order to read the entire review. Additionally, if we go back to the designer's images, you can click on an image and a modal will pop up expanding the image and providing a brief description.

Tyler: So now we are going to demonstrate how the actual interaction occurs. Nick right now is on this account for purposes of the demo. After clicking connect, you will be brought to this page which allows you to submit images of what room you want redesigned or need advice on as well as an initial message. The key thing here is you can’t send the request until the text field is filled out and the images selected, which I will demonstrate now.

Tyler fills out request

Tyler: “There is a max of 4 images because this is the max amount the library lets you choose from. As you can see, once everything has been filled out, you get a preview of the images you chose and the send request button now pops up. **Once you click the button, an indicator pops up letting you know the images are being uploaded to the database and being formatted for gifted chat**. As you can see, all the images are now rendered with the text message and you can await the response from the designer. Also if you want to, you have the choice to upload more images. And now Nick will type out his response”

Nicholas: **Say outloud what the message is**

Tyler: “Then once the interaction is over, and I will type “Thanks,” you end it by leaving a review by clicking on the button on the top right. It then takes you to a review page where you can give them a rating and leave a message. ”

Nicholas: As you can see, after submitting the review, the NID is now brought back to the designers profile. They have the ability to take a look at the designers profile again or go back and browse other designers.

Nicholas: So, that wraps up the NID portion of the demo, now Tyler will reload the app and we will do a test run of someone trying to sign up as a designer.

Tyler reloads the app bringing us back to the home screen

**
The designer would sign up with google credentials, we will have Tyler log in again.
Wait for Tyler to log in
and have their information autofilled on the profile set up page. This time, we continue as a designer. So, we are now brought to the designers profile page. Everything is essentially the same as an NID viewing an ID’s profile, but now the ID has the option to edit their bio and upload / delete images. If you click the settings icon on the top right, a list pops up with some options, we want to edit our bio so Tyler will click that.
Tyler clicks edit bio You can now click the area next to your profile image in order to edit your bio. Notice, instead of the settings icon being on the top right, save is now there; so after we finish editing our bio, we click save.

Tyler: Okay now I will briefly show how deleting and uploading images work. Essentially, what is different for the designer as opposed to an NID is that holding down one of the images that you would like to delete brings up a different type of modal. Instead of the normal modal we saw prior as an NID, a delete button appears and if we click it, that portfolio item is now gone. And then to upload a photo you click the top left and choose a photo. An upload modal now appears and you can fill in a description and click upload and now the portfolio item is added to the existing images.

Tyler: Now we will show the interaction from the designer pov. If you saw earlier, in the settings tab, we can set our status to active and allow users to connect with us. And now a green border appears on your profile picture showing that you are active and people can connect with you. Nick will now message me and I will get an alert that someone wants to interact with me and get my advice.

Nicholas: “I am currently running the app on my phone. **Show chat to class** So now I will message Tyler.

Show chat to class, then message chat

Tyler: As you can see, a modal appears and I can choose to decline by swiping left or accept by swiping right. So once I swipe right I can now see the chat and start interacting. Then once a designer is finished giving advice and the interaction is over, I can click End and end the conversation.

So, that wraps up the demo portion of the presentation.

Tyler transitions back to the slides

Slide 4:

Tyler: “As you can see, our tech stack has not changed that much. We are still using the MERN stack with the inclusion of Firebase. And these are the react libraries that we used which are Gifted Chat, which is how we are able to upload the images and get in contact with a designer. We used React Native Image Crop Picker, which is how we were able to upload the images, and lastly we used react native elements for a majority of the UI work. Nick will be talking more about that during the technical talk segment.”

Slide 5:

Nicholas: “So, now moving on to work distribution. On my end, I handled the implementation of Google sign in, the screen creation and navigation by means of React Native Elements / React Navigation, and the styling across both IOS and Android. Tyler implemented Gifted Chat in order to support image uploading, the dynamic between GiftedChat and Firebase, the request and reviews pages which involve the NID connecting with the ID, and the database and backend functionality.

Slide 6:

Tyler: “ One of the biggest challenges was in fact Gifted Chat. Because our entire app is reliant on this framework, it took an extensive amount of research and I quickly found that the documentation was extremely scarce. There were a couple of videos that showed you how to structure the firebase configuration code with Gifted Chat to get the app working. however, there were barely any resources that showed how to deal with other media types such as images and videos. Therefore, image uploading and sending multiple messages at the same time to another person took quite a bit of trial and error because of the sparse documentation. The way I was able to send multiple image messages as you saw in the demo was by creating a few functions that essentially parsed the request by extracting each image from the request. Then compressing those files through the use of a property from the React Native image crop picker library, and then uploaded all of the images first, and finally sent the message last to the database so it was formatted in the way you saw in the app. And that took a lot of trial and error because the docs and articles I read didnt concisely explain the parsing factor. ”

Nicholas: Another challenge was styling for the app. Styling in general is pretty tedious, especially when you're dealing with nested navigation and just making sure your content is laid out properly on every screen. When moving across Android and IOS not everything cleanly ports over. Layout issues and other small things such as supported fonts to specific OS component props also come into play. Since we decided to demo our app on Tylers side, the workflow for the front end consisted of me laying the groundwork for what we wanted the interface to look like on Android and then coordinating with Tyler to get the look right on IOS. We tried to adhere to general flexbox layouts so it resized to suit various screens and only used set padding when needed. By accounting for the look on both IOS and Android we also take advantage of what React Native is known for, its cross platform play. Furthermore, I knew of React Native but have had little experience with it so getting over that initial learning hump posed a mini challenge. Additionally, google sign in was an issue for us in the

beginning. It was mainly issues with initial setup but when we discovered firebase that process was greatly simplified.

Slide 7:

Tyler: “On to lessons learned, as a result of this being our first time creating an entire project in React Native, it really taught us a lot about the nuances of it. From small things such as how components could be passed down differently from normal react, and the way in which the navigation was controlled. The navigation factor was interesting because of the way it differed from web apps. Normally on web apps everything is stack based, however, with mobile you had to nest these navigation and tab controllers which took some time getting used to.

Nicholas: “I realized the importance of content layout between Android and IOS and by extension, the importance of having entire teams for each OS. Earlier, we got into some of the issues we had with styling between both mobile operating systems, but those issues are nothing compared to what an industry developer might encounter. With the amount of platform specific options to deal with, I can see why having teams for both may be ideal.

Tyler: “Image compression was one of the initial things we were worried about because storing all these images in a database could become costly if we have thousands and thousands of images stored. However, one of the things we learned is that there is an abundance of libraries that can compress files for you and allow you to store them in a database in an extremely intuitive way. All I had to do with the react native image crop picker library was add an object to the method that sends the image with a specified compression level. The idea of data compression is interesting to me and even made me want to look up general encoding algorithms such as Huffman encoding and seeing how they could be applied.

Nicholas: Furthermore, we could have clearly identified all of the features that we wanted. As we progressed with the project, we sometimes got caught up in the number of features that could have been potentially added. This project in particular is relatively feature oriented; but we wanted to provide our baseline of supporting a connection between NID and ID while also providing some of the standard features associated with services of this nature, such as profile customization.

Tyler: “Lastly, learning about real time databases was extremely useful. Because we chose not to use sockets, we had to find a way to update and create information in real time while allowing that information to be pulled just as fast. Firebase’s real time

database in conjunction with GiftedChat showed us just how powerful these databases can be and is definitely something I want to look into in the future."

Slide 8:

Tyler: "Speaking about the future, one of the key things I learned about myself while coding this project is that I thoroughly enjoy coding in Swift way more as opposed to React Native. The amount of times we had to reinstall dependencies and delete dependencies and run the command pod install was absurd. Now this can be and is most certainly a result of us not being experts on the matter, but it is something I will most definitely not miss.

Slide 9:

Questions?

Beginning of technical talks

Slide 10

Tyler: "Okay guys moving on into the next part of our presentation, the technical talks, I will first start off by explaining to you what Singletons are. Then Nick will follow up and talk to you guys about react native elements.

"Starting off, as you see in front of you, singletons are a design pattern. But what exactly is a design pattern? To put it simply, design patterns are a list of best practices you would use for certain programming scenarios that you see over and over again, like how to create objects of a certain type without having to understand all the implementation details, such as an iterator.It is a set of guidelines for solving a common software scenario that cannot be broken if you decide to conform to it and this is where singletons can get complex, which we will touch on later. But you now might be asking yourself the question, what is a singleton?"

Slide 11:

"Well, the idea of a singleton is actually pretty easy to understand. It is a design pattern that restricts the instantiation of a class to one object, meaning that once that object is created, it will exist throughout the application's lifecycle.To give a real-world example think of it like an operating system that has only one file system. Everything on that operating system has access to that file system in some sort of way such as when you log onto your Mac and you can use the finder which has access to all your files that are under your account. It's kind of like a shared resource in that way such as in the picture

you see in front of you. All these different objects can access that Singleton at once just like multiple users on a mac can access their part of the file system. And this moves us into the next property of Singleton's which is that the instance of the class that is created is going to be accessible globally, almost like a fancy global variable. It's not quite the same but we'll get into that in a bit.

Slide 12:

"Moving on, the purpose of the singleton is to control object creation, limiting the number of objects to only one. The singleton allows only one entry point to create the new instance of the class. As you see in the picture, again driving home the point of global access, all these different clients have access to that one Singleton instance through the property getInstance(). And once you have that access, you can call upon any method or variable within that instance and use it throughout the app. This can be extremely useful when you need to control the resources of a software system such as database connections or sockets."

Slide 13:

"Adding on to why using it can help you on a project, singletons can be good for protecting shared resources. An example of this is creating a database class to connect to a database through one connection, or one instance. Through this one instance you can control load balancing of the database, and manage unnecessary connections to the database because there exists only one instance, so if there are any errors you know exactly where to look. **And they support one object performing some type of app wide coordination such as the aforementioned database example or an API, which we will touch on in a slide or two.**

Slide 14:

"So now moving on to the actual implementation we had in our own project, we used it for gifted chat and the integration of firebase. As you see in front of you, you have a normal JavaScript class where there's a Constructor and I included one of the methods because the other methods aren't that important and I just wanted to show you that the init method was simply to set up the connection to Firebase, however, one of the most important aspects of this Singleton pattern is those two lines of code right there. We're defining a property on the fire class called shared, which is a convention that is usually used with singletons, to be the sole instance that will be used across the app. And then in that second line of code we're simply exporting the class to allow it to be imported and used in other files as we will see in the next slide.

Slide 15:

“Okay so after importing the fire class in this file we then have access to all of that classes methods and variables through that fire.shared property. In our project for gifted chat, every time a new message is sent by an NID, that .on method is run and updates the database with the new messages and formats it in a way that gifted chat wants. And then in the componentWillUnmount function, whenever somebody exits the chat it simply cuts the connection to the database. And again Singletons are a design pattern so they're not language specific. And to drill this point home we will look at 2 classes here in Swift.

Slide 16:

“Here are two examples of API manager classes that expose the singleton instance via its shared property, as convention. You use API.shared.makeAPIRequest() to access the API through a single, unified instance. This allows you then to manage API calls serially, and thus have cleaner architecture, showing how Singletons can be extremely useful.

Slide 17:

“On to some pros and cons of singletons. First you can Implement their interfaces and by this I mean because Singletons are a design pattern you don't have to use them in a specific way, you can use them anyway you want actually it's just a matter of conforming to those rules that there is one instance and it can be accessed globally. Two, they do not take up space in the global namespace. If you remember earlier when I said they are like a fancy global variable this is what I meant. Because they do not take up space in the global namespace but still have that global characteristic, it makes it a lot easier to share certain resources across the app without hindering performance. However, as a result they do share global mutable state. And with this, it can make a program’s state unpredictable as any object throughout your application, if used incorrectly, can change the state at any time making the state unreliable. And if you can't have reliable state, then the application as a whole will be unreliable. And lastly, they can cause unit testing to be cumbersome. By this I mean It may be difficult to unit test the client code of the Singleton because many test frameworks rely on inheritance when producing test objects. However, since the constructor of the singleton class is private, you will need to think of a creative way to mock the singleton. And thus, as the software complexity increases, it will limit the flexibility of the program as more of these tests have to be written.

Slide 18:

“And now here at the end, what I hope you all take away from this is that singletons are a very popular and commonly adopted pattern because of simplicity. It can be dumbed down to these two pieces of clipart. On the one hand, you have global access anywhere throughout your app and on the other, only one instance of a class is exposed to its clients or other objects. And that wraps it up, thanks.

Nicholas’ technical talk; React Native Elements

Slide 19:

“Ok, so now we’re moving on to our other technical talk on React Native Elements

Slide 20:

“So, what is React Native Elements? It is a cross platform UI toolkit used to integrate components into your application. Components include Avatars, Sliders, Cards, Button Groups, and many more. These components are common to most UI’s but not standard enough to be included in the React Native core component list; by standard I mean not every interface will need a rating bar or slider for example but a high percentage of interfaces will need some way to capture text input or support image rendering, with the latter being supported by the core component list. You would use React Native Elements in conjunction with React Natives core components and API’s in order to build a UI of your liking. You are not limited to the core component list from React Native; on the website they offer resources to help find other libraries and they offer some brief support on library compatibility. So this library is not the only option to work with, as I’ll touch upon in the coming slides.

Slide 21:

“Alright so, what role did this play in our project? To start off, some of the components used in building up our UI were, avatars, ratings, icons, and dividers. If you remember from the demo earlier, avatars were used to display the designer chosen image, star based ratings were next to each designer, icons such as the briefcase icon helped in displaying designer images, and dividers could be seen separating content on the reviews and “find a designer page”. Using these components provided by React Native Elements helped us easily translate our mockups into working UI components. Before this project, I knew of react native but had little experience with it. Using this library in conjunction with react natives core component list greatly helped productivity in terms of getting components out to interact with.

Slide 22:

“So, now lets see how to implement some components. First thing you would need to do is make sure react native elements is installed; either npm install react-native-elements or yarn add react-native-elements. Next, in your file, you would import the components you need; as you can see in the image on the top left, I import divider, avatar, rating, badge, and icon from react native elements. These other images show the components in action. Over here on the top right is the starbased rating which has a number of prop specific options associated with it like readonly, image / star size , and the color of the star. Icon on the bottom left has associated prop options to specify the icon you want to use and avatar has options to specify the source image, border thickness and overall size of the avatar. Each component does have more props to choose from and can be seen on the react native elements website. From here you would normally wrap your component in a view or other wrapping tag in order to position your component where you want it. For example, the rating component is wrapped in a view because I want to position it relative to the designer's name and the icon component is wrapped in a touchable opacity so it provides some visual feedback upon clicking but also allows us to freely position it. So, overall component interaction is easy, it’s just a matter of layouts and knowing the component specific props.

Slide 23:

“So what are some other options available to us and why use it? There are a number of other component libraries such as NativeBase, Material Kit, React Native Paper, and the list goes on. Some libraries may offer the same components (with different prop options) while some provide others. Ultimately, it comes down to which library is best suited for your needs. You want to avoid using multiple libraries for different components because that's not good practice. This could lead to code bloat, conflicting issues between libraries and an inconsistent style since the components would differ. Whether it's a class project or something outside of the classroom, you want your interface to be consistent and on the code side of things you don’t want to be micromanaging components from different libraries and the associated differences between the two. With that being said, we chose react native elements because it had everything we needed for our UI and it was easy to get started with as you saw in the previous slide.

Slide 24:

“So what are some takeaways after using the library? We only used a small subset of what the library had to offer so you could definitely build a robust interface by utilizing

the many components it has available. Like I've mentioned, it's really easy to implement, and I feel like it has a good set of starter components for quickly building up a UI. It’s easy knowing what the components do and knowing the props for each but the content layout is pretty tedious.

Slide 25:

Technical questions?