Indesyn

Nicholas Samaroo, Tyler Torres
Hunter College
CS 499 Capstone Spring 2020
Final Presentation
5/18/20

01 Problem

- The process of finding an interior designer is not very streamlined.
- There also lacks a diverse platform that connects people with a variety of interior designers.

02 Vision

- Creating a platform for users to find an interior designer, on the other hand, establishing a hub for designers to connect with users and display their work.

03 Product Definition

- The ultimate goal is to liven up the interior space of our users
- Connecting people to those with the trained eye and giving them a platform to receive the aid they need

- Our platform accomplishes this end goal by displaying designer's past work to exhibit expertise
- Allows both parties to interact efficiently

Demo

Tech Stack

MERN

mongoDB

Express JS

React Native

node JS

Firebase

Firebase

React Libraries

React Native Image Crop Picker

REACTNATIVE ELEMENTS

Work Distribution

Nicholas

- Implemented Google Authentication
- Screen creation and navigation
- Styling across both IOS and Android

Tyler

- Implemented Gifted Chat and functionality to send multiple images as their own messages
- Request and Review pages pertaining to GC
- Set up Firebase to interact with GC
- Database and backend functionality

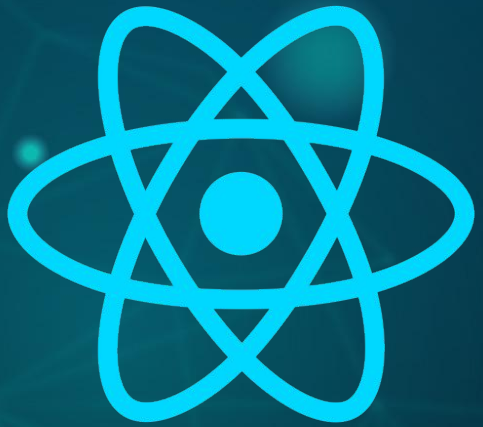
Challenges

- Gifted Chat
  - Documentation is sparse
  - Not many tutorials relating to our specific work
  - Trial and Error (lots of it)
- Styling
  - General styling and uniformity between Android and IOS
  - New to react native
- Google Authentication

Lessons Learned

- First time using React Native, thus, we learned a lot about the technology and its nuances
- Importance of general layouts for use between IOS and Android devices
- Image upload and being able to compress files
- Don’t get caught up in smaller features; identify key small features
- Learned more about databases and how different types are more efficient for specific projects
    - Firebase’s realtime database

For the Future



VS

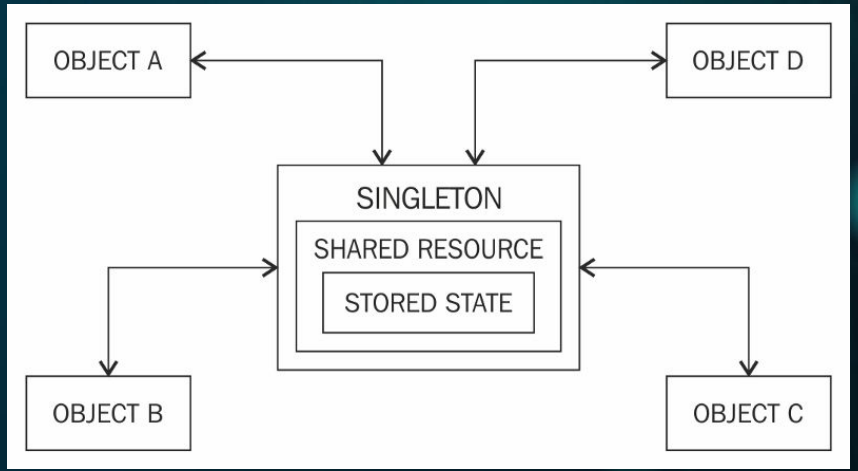
Questions?

Singletons

A Design Pattern

What is a Singleton?

- A singleton is a class in which only one instance exists throughout the application's lifecycle
- The instance of the class is accessible through the class as a property or method of that class, making it **globally** accessible



Purpose

The purpose of the Singleton class is to **control object creation**

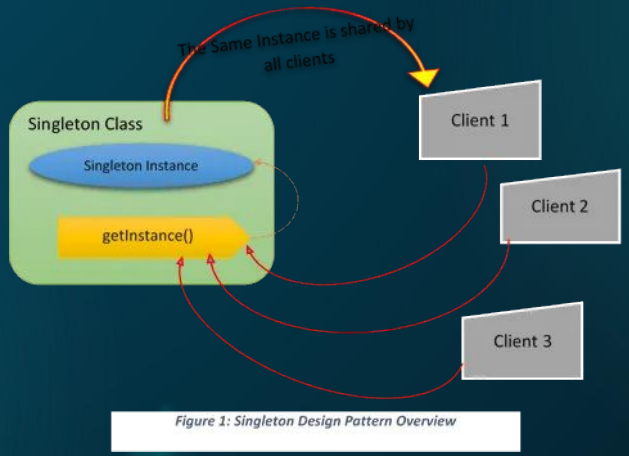


Figure 1: Singleton Design Pattern Overview

Singletons are often useful where you have to control the resources, such as database connections or sockets.

Why use it?

Singletons can be good for protecting shared resources, providing access to some system which contains only one instance of an object, supporting one object which performs some type of app-wide coordination

Implementation

```
class Fire {
  constructor() {
    this.init();
    this.observeAuth();
  }

  init = () => {
    if (!firebase.apps.length) {
      firebase.initializeApp({
        apiKey: 'AIzaSyAj5mfd9eMZmvcSfZyNav9fjCe1i_v17GM',
        authDomain: 'indesyn-3fb48.firebaseapp.com',
        databaseURL: 'https://indesyn-3fb48.firebaseio.com',
        projectId: 'indesyn-3fb48',
        storageBucket: 'indesyn-3fb48.appspot.com',
        messagingSenderId: '929469838203',
        appId: '1:929469838203:web:68a15f4b6a9aec735a03de',
        measurementId: 'G-W7VEZCJPC1',
      });
    }
  };
```

```
Fire.shared = new Fire();
export default Fire;
```

Usage of Singleton

```
componentWillUnmount() {
  Fire.shared.off;
}
```

```
Fire.shared.on(message => {
  this.setState(previousState => ({
    messages: GiftedChat.append(previousState.messages, message),
  }));
```

```
on = callback => this.database.on('child_added', snapshot => callback(this.parse(snapshot)));

off = () => {
  this.database.off();
};
```

Usage of Singleton

Examples of API manager classes that enforce the one instance rule

```
class NetworkManager {

    // MARK: - Properties

    static let shared = NetworkManager(baseURL: API.baseURL)

    // MARK: -

    let baseURL: URL

    // Initialization

    private init(baseURL: URL) {
        self.baseURL = baseURL
    }

}
```

```
class API
{
    static let shared = API()
    var isRequestPending = false

    private init() { }

    func makeAPIRequest()
    {
        if isRequestPending {
            return
        }

        isRequestPending = true

        // Make the API HTTPS request...
    }

    func onReturnAPIRequest()
    {
        isRequestPending = false

        // Do something with request data...
    }
}
```

Pros vs Cons

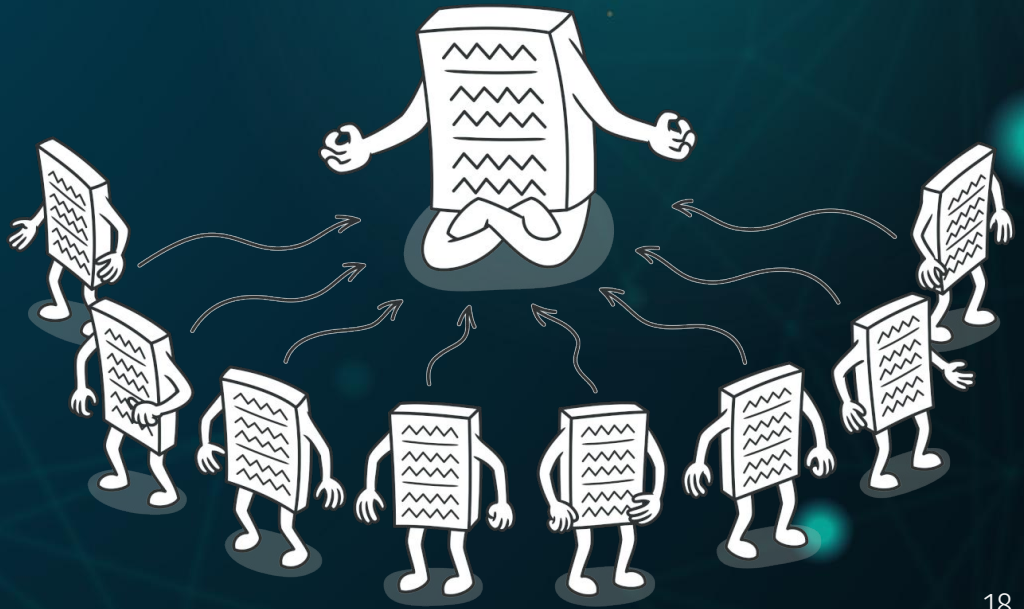
- Can implement interfaces
- Do not take up space in the global namespace

- Shares global mutable state
- Can cause unit testing to be cumbersome

Singleton Takeaways

Global Access

Single Instance



React Native Elements

UI Toolkit

What is React Native Elements?

- A cross platform UI toolkit used to integrate components into your application
- Components include Avatar, Slider, Cards, Button Groups, etc.
- Essentially, components common to a UI but not standard enough to be included in the React Native core component list
- Would use this in conjunction with standard React Native components and API'S

Role it played in our project

1. Avatar, Rating, Icon, and Divider were some of the components used in building up our UI
2. Helped me easily translate our mockups into working UI components

Implementation

```
import React, { Component, useEffect } from "react";
import { Divider, Avatar, Rating, Badge, Icon } from "react-native-elements";
import {
  StyleSheet,
  View,
  Text,
  Image,
  ImageBackground,
  TextInput,
  StatusBar,
  TouchableOpacity,
} from "react-native";
import { disableExpoCliLogging } from "expo/build/logs/Logs";
import { SafeAreaView } from "react-native-safe-area-context";
import { ScrollView } from "react-native-gesture-handler";
```

```
<View style={styles.rating}>
  <Rating
    type="custom"
    imageSize={35}
    ratingColor="#3498db"
    ratingBackgroundColor="transparent"
    readonly
  />
</View>
```

```
<TouchableOpacity>
  <Icon name="account-multiple" type="material-community" size={50} />
</TouchableOpacity>
```

```
<Avatar
  size="large"
  containerStyle={{ borderWidth: 3 }}
  rounded
  source={{
    uri: "https://i.picsum.photos/id/1027/2848/4272.jpg",
  }}
/>
```

Other options / Why use it?

- There are a number of other component libraries such as NativeBase, Material Kit, React Native Paper, etc.
- Some libraries may offer the same components (with different prop options) while some provide other component options
- Ultimately, it comes down to which library is best suited for your needs
- React Native Elements is easy to get started with

Takeaways

We used a small subset of what the library had to offer; could definitely utilize the many components in the library to build a robust application

Really easy to implement
Good set of “starter” components for quickly building up an interface

Technical Questions?