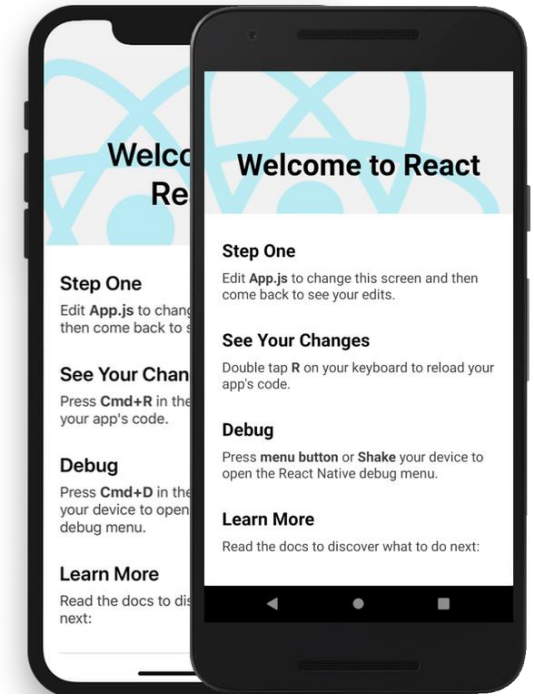
Android

iOS

React Native

React Native

1. Теория



React Native combines the best parts of native development with React, a best-in-class JavaScript library for building user interfaces.

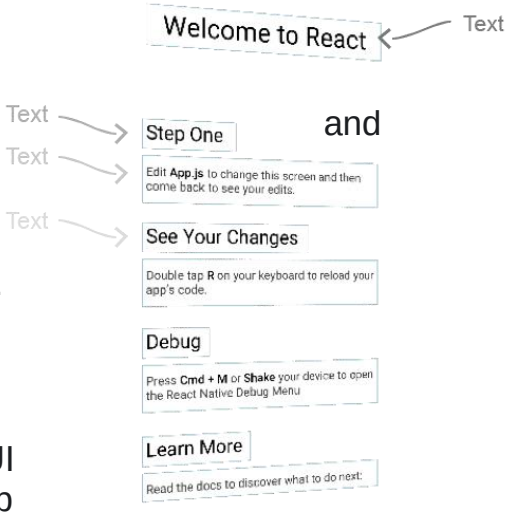
Use a little—or a lot. You can use React Native today in your existing Android and iOS projects or you can create a whole new app from scratch.

Written in JavaScript—rendered with native code

React primitives render to native platform UI, meaning your app uses the same native platform APIs other apps do.

Many platforms, one React. Create platform-specific versions of components so a single codebase can share code across platforms. With React Native, one team can maintain two platforms and share a common technology—React.

Native Development For Everyone



React Native lets you create truly native apps doesn't compromise your users' experiences. It provides a core set of platform agnostic native components like View, Text, and Image that map directly to the platform's native UI building blocks.

Seamless Cross-Platform

React components wrap existing native code and interact with native APIs via React's declarative UI paradigm and JavaScript. This enables native app development for whole new teams of developers and can let existing native teams work much faster.

Expo is a framework and a platform for universal React applications. It is a set of tools and services built around React Native and native platforms that help you develop, build, deploy,

and quickly iterate on iOS, Android, and web apps from the same JavaScript/TypeScript codebase.

Tools and Services

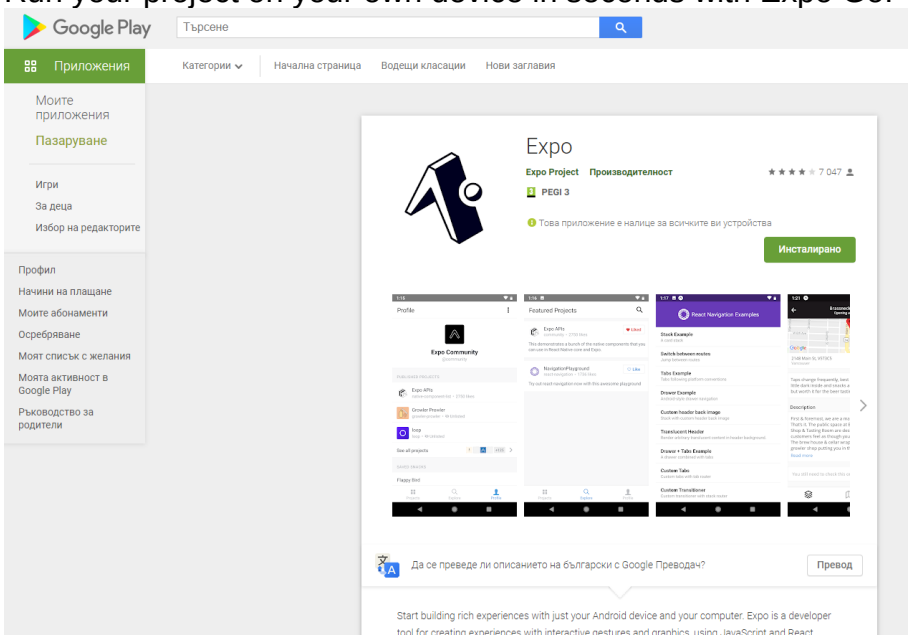
Expo tools and services empower you to create incredible apps using Expo CLI and the Expo SDK, the Expo Go app, our cloud build and submission services, and Expo Snack.

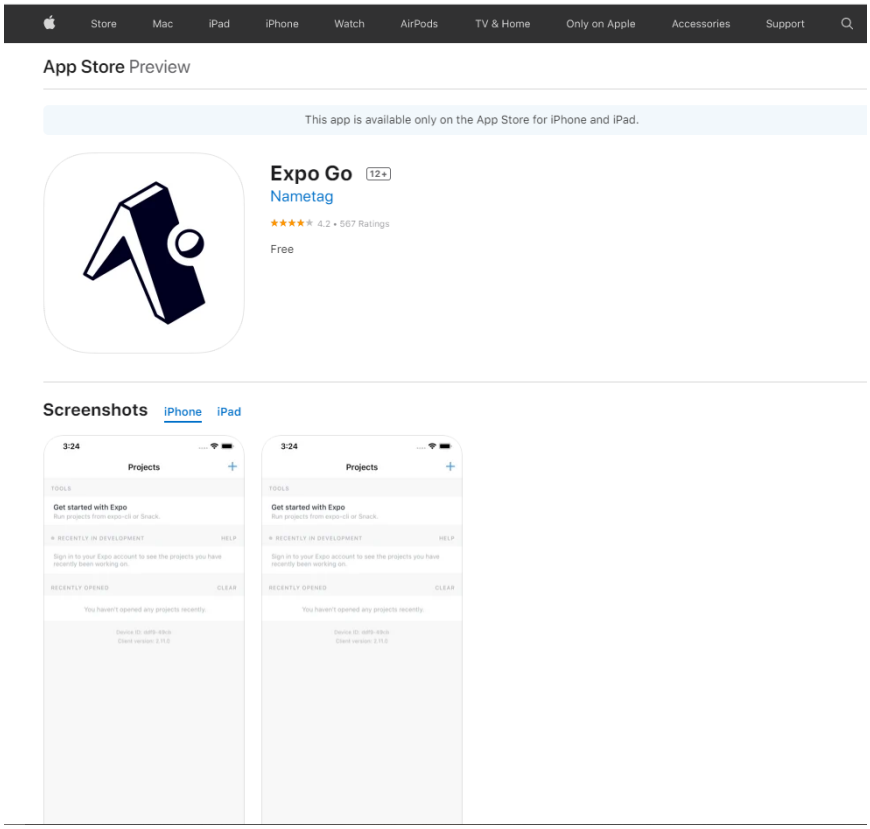
Develop with Expo's CLI and SDK

Install Expo CLI to create and run your project. Then use our SDK's APIs and components to build a fully-featured application.

Run your project with Expo Go

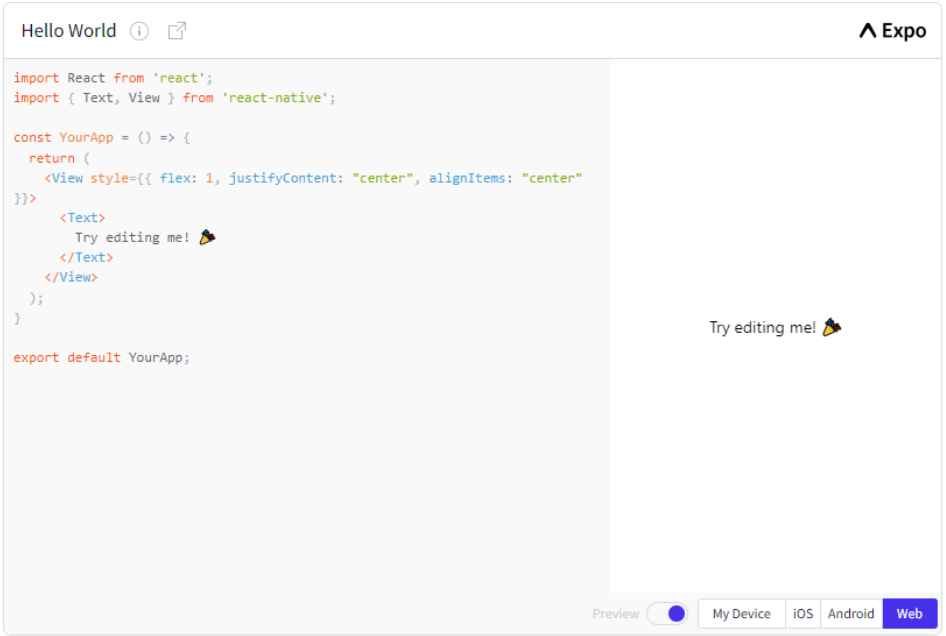
Run your project on your own device in seconds with Expo Go.





Interactive examples

This introduction lets you get started immediately in your browser with interactive examples like this one:



https://reactnative.dev/docs/getting-started

The above is a Snack Player. It’s a handy tool created by Expo to embed and run React Native projects and share how they render in platforms like Android and iOS. The code is live and editable, so you can play directly with it in your browser. Go ahead and try changing the "Try editing me!" text above to "Hello, world!"

Function Components and Class Components

With React, you can make components using either classes or functions. Originally, class components were the only components that could have state. But since the introduction of React's Hooks API, you can add state and more to function components.

Hooks were introduced in React Native 0.59., and because Hooks are the future-facing way to write your React components, we wrote this introduction using function component examples. Where useful, we also cover class components under a toggle like s

```
import React from 'react';
import { Text, View } from 'react-native';

const HelloWorldApp = () => {
  return (
    <View style={{
       flex: 1,
       justifyContent: 'center',
       alignItems: 'center'
     }}>
     <Text>Hello, world!</Text>
    </View>
  );
}

export default HelloWorldApp;
```

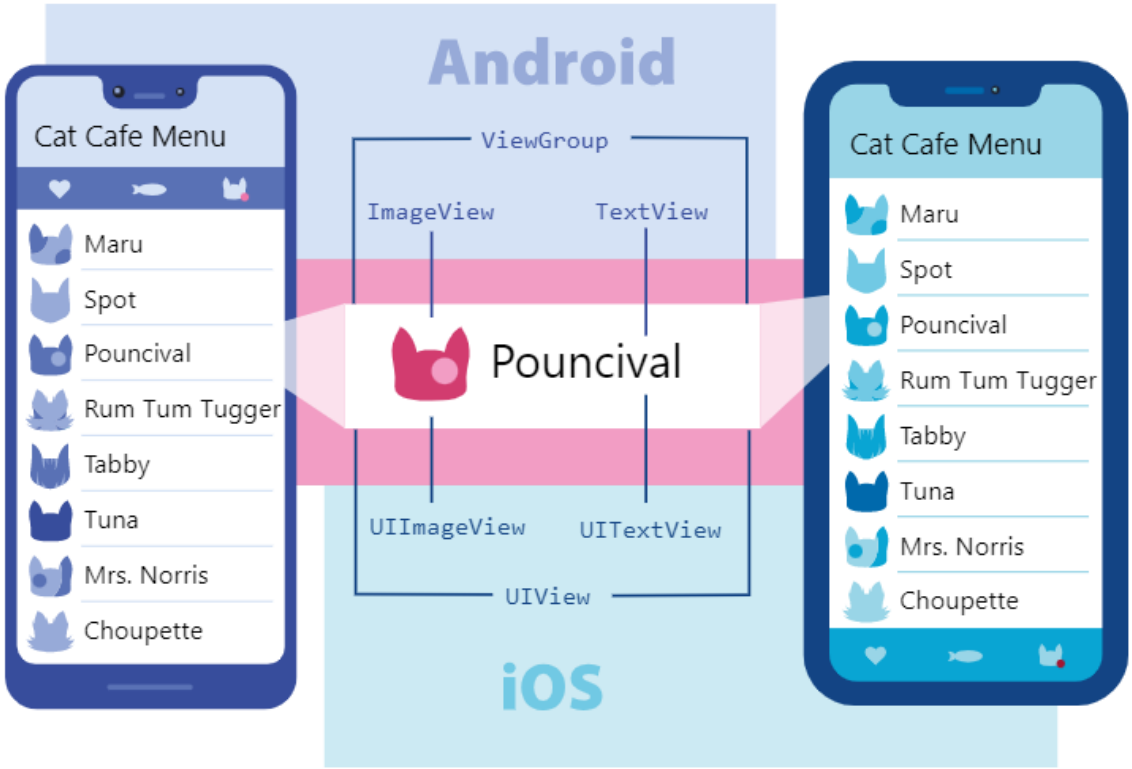
```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class HelloWorldApp extends Component {
  render() {
    return (
      <View style={{
         flex: 1,
         justifyContent: "center",
         alignItems: "center"
       }}>
       <Text>Hello, world!</Text>
      </View>
    );
  }
}

export default HelloWorldApp;
```

Core Components and Native Components

React Native is an open source framework for building Android and iOS applications using React and the app platform's native capabilities. With React Native, you use JavaScript to access your platform's APIs as well as to describe the appearance and behavior of your UI using React components: bundles of reusable, nestable code. You can learn more about React in the next section. But first, let's cover how components work in React Native.

Views and mobile development

In Android and iOS development, a **view** is the basic building block of UI: a small rectangular element on the screen which can be used to display text, images, or respond to user input. Even the smallest visual elements of an app, like a line of text or a button, are kinds of views. Some kinds of views can contain other views. It's views all the way down!



Just a sampling of the many views used in Android and iOS apps.

Native Components

In Android development, you write views in Kotlin or Java; in iOS development, you use Swift or Objective-C. With React Native, you can invoke these views with JavaScript using React components. At runtime, React Native creates the corresponding Android and iOS views for those components. Because React Native components are backed by the same views as Android and iOS, React Native apps look, feel, and perform like any other apps. We call these platform-backed components **Native Components.**

React Native comes with a set of essential, ready-to-use Native Components you can use to start building your app today. These are React Native's **Core Components**.

React Native also lets you build your own Native Components for Android and iOS to suit your app's unique needs. We also have a thriving ecosystem of these **community-contributed components.** Check out Native Directory to find what the community has been creating.

Core Components

React Native has many Core Components for everything from form controls to activity indicators. You can find them all documented in the API section. You will mostly work with the following Core Components:

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
<View>	<ViewGroup>	<UIView>	A non-scrollling	A container that supports layout with flexbox, style, some touch handling, and accessibility controls

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
<Text>	<TextView>	<UITextView>	<p>	Displays, styles, and nests strings of text and even handles touch events
<Image>	<ImageView>	<UIImageView>	<img>	Displays different types of images
<ScrollView>	<ScrollView>	<UIScrollView>		A generic scrolling container that can contain multiple components and views
<TextInput>	<EditText>	<UITextField>	<input type="text">	Allows the user to enter text

```jsx
import React from 'react';

import { View, Text, Image, ScrollView, TextInput } from 'react-native';

const App = () => {
  return (
    <ScrollView>
      <Text>Some text</Text>
      <View>
        <Text>Some more text</Text>
        <Image
          source={{
            uri: 'https://reactnative.dev/docs/assets/p_cat2.png',
          }}
          style={{ width: 200, height: 200 }}
        />
      </View>
      <TextInput
        style={{
          height: 40,
          borderColor: 'gray',
          borderWidth: 1
        }}
        defaultValue="You can type in me"
      />
    </ScrollView>
  );
}

export default App;
```

Handling Text Input

TextInput is a Core Component that allows the user to enter text. It has an onChangeText prop that takes a function to be called every time the text changed, and an onSubmitEditing prop that takes a function to be called when the text is submitted.

```
import React, { useState } from 'react';

import { Text, TextInput, View } from 'react-native';

const Translator = () => {
  const [text, setText] = useState('');
  return (
    <View style={{padding: 10}}>
      <TextInput
        style={{height: 40}}
        placeholder="Type here to translate!"
        onChangeText={newText => setText(newText)}
        defaultValue={text}
      />
      <Text style={{padding: 10, fontSize: 42}}>
        {text}
      </Text>
    </View>
  );
}

export default Translator;
```

Using a ScrollView

The ScrollView is a generic scrolling container that can contain multiple components and views. The scrollable items can be heterogeneous, and you can scroll both vertically and horizontally (by setting the horizontal property).

This example creates a vertical ScrollView with both images and text mixed together.

https://reactnative.dev/docs/using-a-scrollview

Using List Views

React Native provides a suite of components for presenting lists of data. Generally, you'll want to use either FlatList or SectionList.

The FlatList component displays a scrolling list of changing, but similarly structured, data. FlatList works well for long lists of data, where the number of items might change over time. Unlike the more generic ScrollView, the FlatList only renders elements that are currently showing on the screen, not all the elements at once.

The FlatList component requires two props: data and renderItem. data is the source of information for the list. renderItem takes one item from the source and returns a formatted component to render.

This example creates a basic FlatList of hardcoded data. Each item in the data props is rendered as a Text component. The FlatListBasics component then renders the FlatList and all Text components.

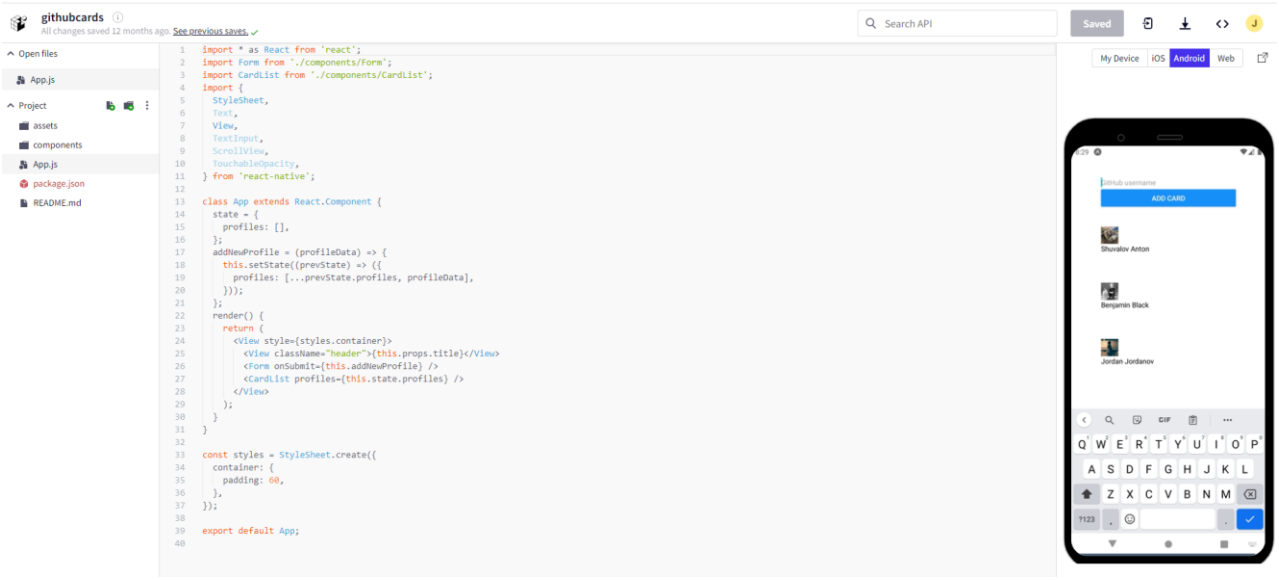
https://reactnative.dev/docs/using-a-listview

2. Github Cards in React Native

Re-write Github Cards project https://github.com/profjordanov/githubcards

Use Snack - React Native in the browser (expo.dev)

Reference: https://github.com/profjordanov/hybrid-app-development/blob/master/07.%20React%20Native/githubcards.txt



3. Student Admin

Your task is to implement application "Student Admin" that interacts with a REST API to display student information and allows users to upload a profile picture.

Requirements:

1. Set up a React Native environment with all the necessary dependencies.

2. Implement the following features using React Native:

- Display Students:
  - Fetch student data from the API using a `GET` request.
  - Display the list of students with their names as clickable elements.
  - On clicking a student's name, navigate to a detailed view page that shows their:
    - Full name
    - Faculty number
    - Email
    - Username

- Profile picture (if available)

- Capture and Upload Picture:

  - Add functionality to take a photo using the device's camera.

  - Display a preview of the captured photo before uploading.

  - Enable users to upload the photo alongside the student's faculty number and a password using a `PUT` request.

3. Use Basic Authentication when making API requests.

  - Encode the username and password (`guest:guest`) to Base64 and include it in the `Authorization` header for all API requests.

Design Recommendations:

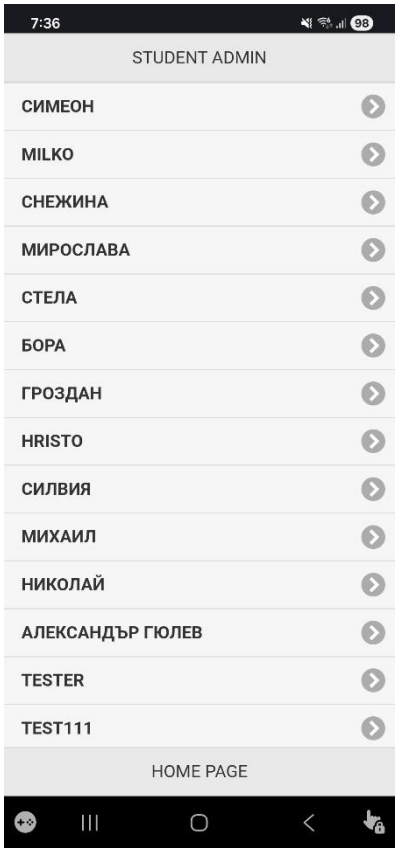
- A home page listing all students, each linking to a details page.

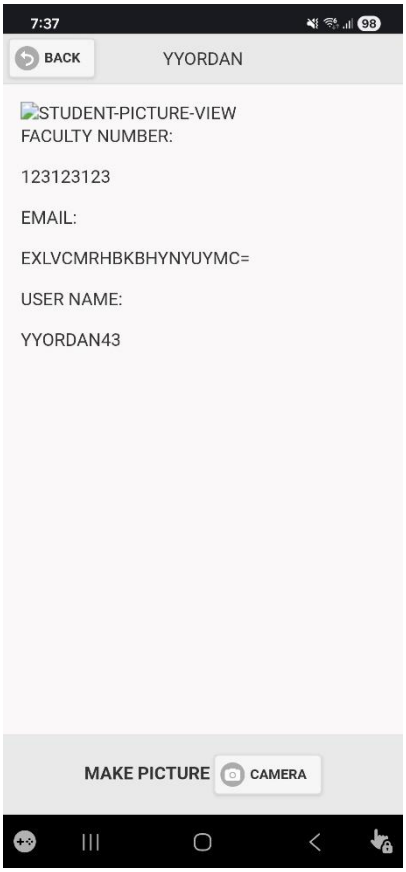
- A button to capture a profile picture that opens the camera.

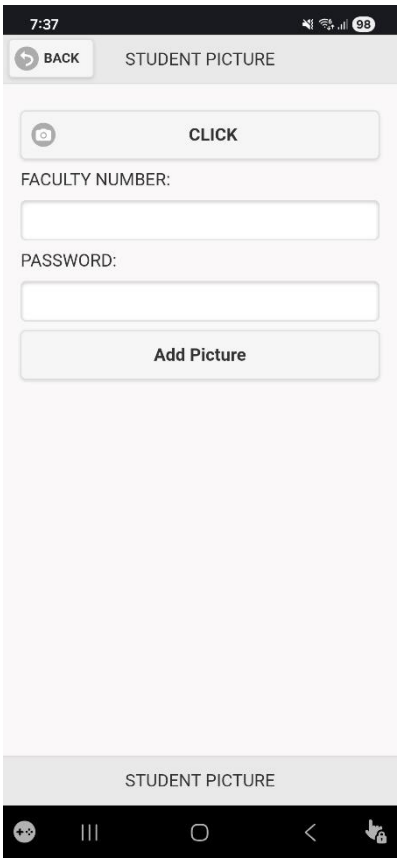
- An input field for the faculty number and password, as well as a button to upload the captured image.

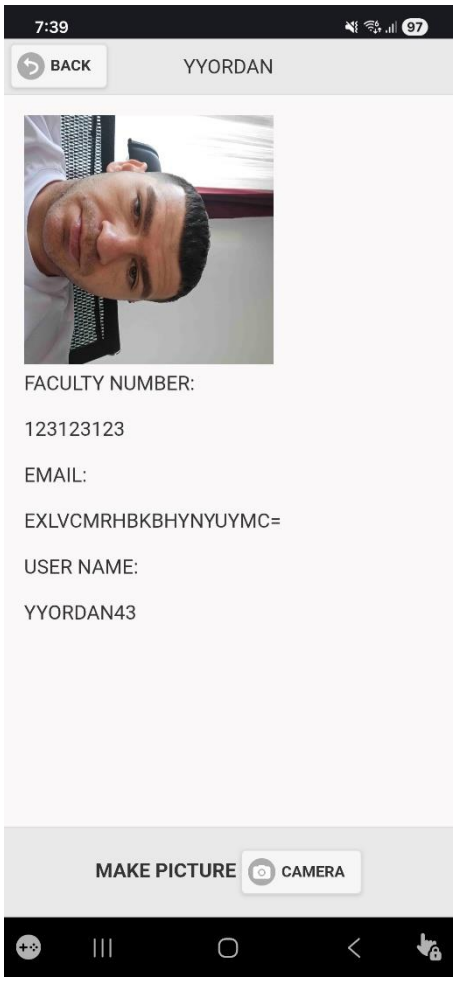
Ensure the app is responsive and works on both Android and iOS.

Focus on implementing good coding practices and creating a user-friendly interface! Happy coding! 🎉


7:36
STUDENT ADMIN
СИМЕОН
MILKO
СНЕЖИНА
МИРОСЛАВА
СТЕЛА
БОРА
ГРОЗДАН
HRISTO
СИЛВИЯ
МИХАИЛ
НИКОЛАЙ
АЛЕКСАНДЪР ГЮЛЕВ
TESTER
TEST111
HOME PAGE


7:37
BACK
YYORDAN
STUDENT-PICTURE-VIEW
FACULTY NUMBER:
123123123
EMAIL:
EXLVCMRHBKBHYNYUYMC=
USER NAME:
YYORDAN43
MAKE PICTURE
CAMERA


7:37
BACK
STUDENT PICTURE
CLICK
FACULTY NUMBER:
PASSWORD:
Add Picture
STUDENT PICTURE


7:39
BACK
YYORDAN
FACULTY NUMBER:
123123123
EMAIL:
EXLVCMRHBKBHYNYUYMC=
USER NAME:
YYORDAN43
MAKE PICTURE
CAMERA