Development for Mobile, Wearable and Smart Devices

Red Capes

Today

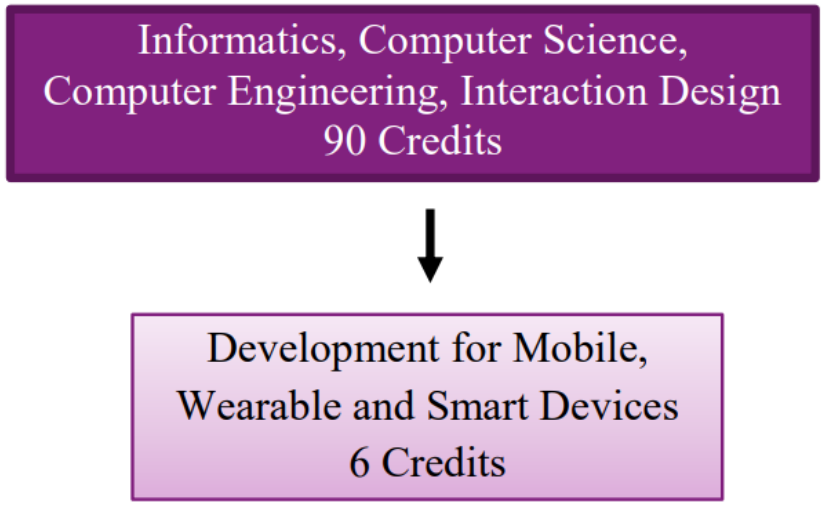
Vue Native introduction

- ✓ Prerequisities
- ✓ Installing Node, Expo & Vue Native
- ✓ Starting a new project
- ✓ Running the project on your phone
- ✓ Basic Vue Native: Components & control structures

Instructions can be found at vue-native.io/docs/

Red Capes

Course Prerequisities



This course will rely on basic knowledge about

- ✓ HTML
- ✓ Javascript
- ✓ CSS

There are introductory courses on these topics here:

HTML | CSS | Frameworks | Javascript

Red Capes

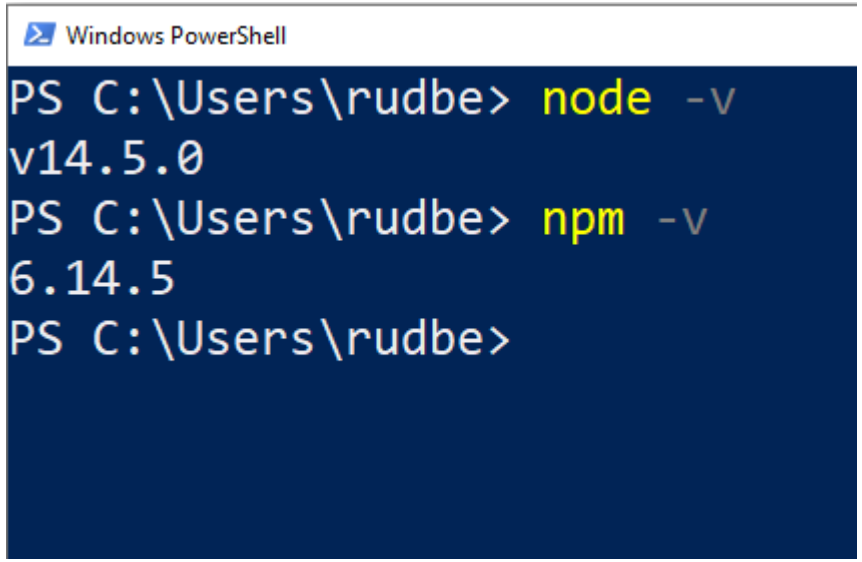
Installing Node, Expo & Vue Native

Node.js

- ✓ JavaScript runtime built on Chrome's V8 JavaScript engine
- ✓ Used for creating and running our Vue Apps
- ✓ Download and install manually from nodejs.org

Red Capes

Installing Node, Expo & Vue Native



Test Node JS after installation

- ✓ Open a new terminal / Powershell window
- ✓ Type *node -v* and press enter
  - ✓ A version number should be returned
- ✓ Type *npm -v* and press enter
  - ✓ Another version number should be returned

Red Capes

Installing Node, Expo & Vue Native

Vue Native is a framework to build cross platform native mobile apps using JavaScript

Vue Native CLI

- ✓ Command Line Interface for creating and running Vue native applications

- ✓ Install with the terminal
  - ✓ *npm i -g vue-native-cli*

Red Capes

Installing Node, Expo & Vue Native

Expo

- open-source platform for making universal native apps for Android, iOS, and the web with JavaScript

- Used for running our Vue Apps on our phones (android / ios)
- Install with the terminal
  - *npm i -g expo-cli*

Red Capes

Starting a new project

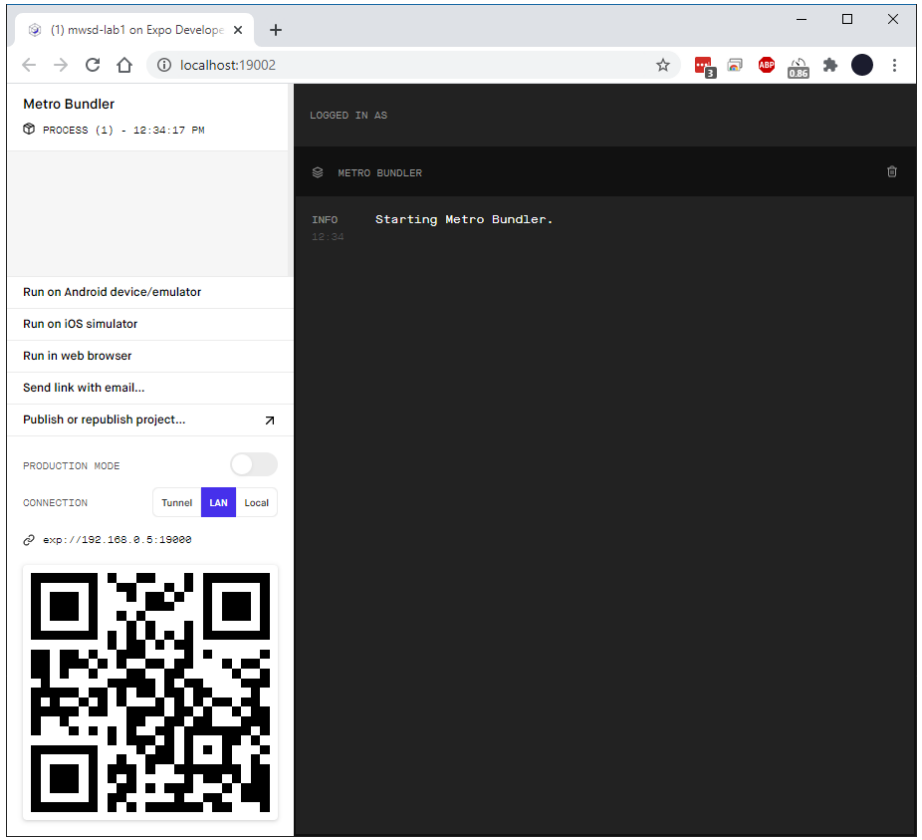
Vue Native is a framework to build cross platform native mobile apps using JavaScript

Create project

- ✓ Open a terminal window inside a folder where you want to create your project
- ✓ Run *vue-native init <projectName>* in the terminal
  - ✓ Replace <projectName> with the actual name of your project

Red Capes

Running the project on your phone

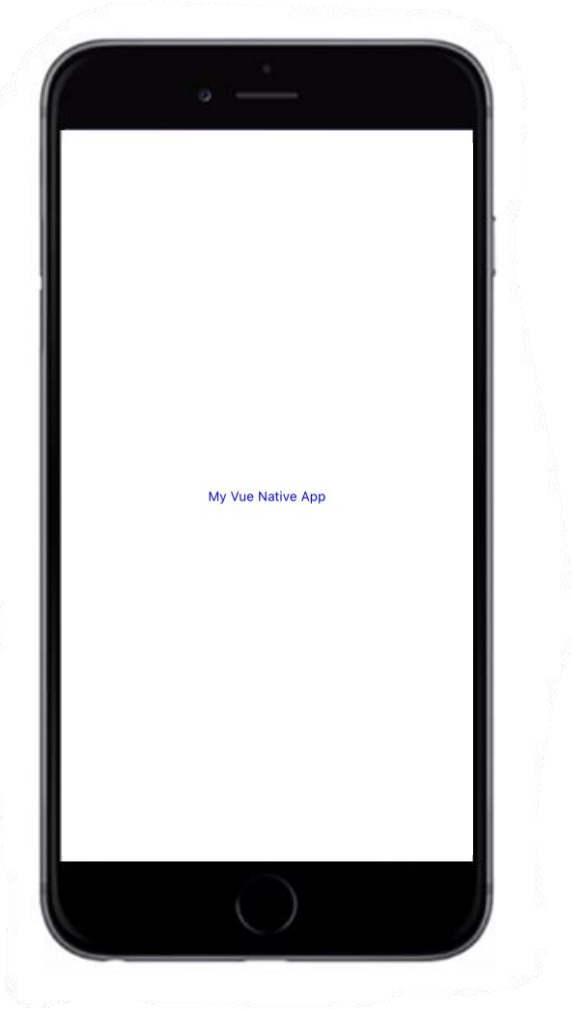


Run project

- ✓ Run *cd <projectName>* where <projectName> is replaced by your actual project name
- ✓ Run npm run start to start expo in a browser
- ✓ Install the expo app on your phone
- ✓ Connect to the same network as your computer
- ✓ Scan the QR code presented by Expo
- ✓ The app should start shortly

Red Capes

Running the project on your phone



Run project

- ✓ After Expo has completed building the app, you should see it in your phone

- ✓ Some yellow warnings might be visible when you start the app, ignore these

Red Capes

Basic Vue Native

App.vue

```
App.vue > template
1   <template>
2     <view class="container">
3       <text class="text-color-primary">My Vue Native App</text>
4     </view>
5   </template>
6
7   <style>
8   .container {
9     background-color: white;
10    align-items: center;
11    justify-content: center;
12    flex: 1;
13  }
14  .text-color-primary {
15    color: blue;
16  }
17  </style>
18
```

App.vue

- ✓ Here's your apps base code
- ✓ This is where your app is started
- ✓ Change the text "My Vue Native App" to something else and save the file to see the app update in your phone

Red Capes

Basic Vue Native

```HTML
<template>
  <view class="container">
    <text class="text-color-primary">{{ message }}</text>
    <button title="Press me!" @press="exclaim" />
  </view>
</template>

<script>
export default {
  data() {
    return {
      message: "Hello World"
    };
  },
  methods: {
    exclaim() {
      this.message += "!";
    }
  },
};
</script>

<style>
.container {
  flex: 1;
  background-color: white;
  align-items: center;
  justify-content: center;
}
.text-color-primary {
  color: blue;
  font-size: 30;
}
</style>
```

Methods & Data

- ✓ Add methods that can run from interractions with your app
- ✓ Add data that can change and update live in your app
- ✓ Add styling and classes to change the looks of your app

Red Capes

Basic Vue Native

```HTML
<template>
  <view class="container">
    <text class="text-color-primary">{{ message }}</text>
    <button title="Press me!" @press="exclaim" />
  </view>
</template>

<script>
export default {
  data() {
    return {
      message: "Hello World"
    };
  },
  methods: {
    exclaim() {
      this.message += "!";
    }
  },
};
</script>

<style>
.container {
  flex: 1;
  background-color: white;
  align-items: center;
  justify-content: center;
}
.text-color-primary {
  color: blue;
  font-size: 30;
}
</style>
```

Methods & Data

- ✓ Add methods that can run from interractions with your app
- ✓ Add data that can change and update live in your app
- ✓ Add styling and classes to change the looks of your app
- ✓ This code will add an exclamation mark on each press of the button

Red Capes

Basic Vue Native

```JS
<script>
export default {
  data: function() {
    return {
      message: "Hello World"
    };
  },
  methods: {
    handleBtnPress: function() {
      alert('Btn Press');
    }
  }
};
</script>
```

Methods & Data

- ✓ Type *alert('Some message')* to show an alert from a method

- ✓ Alert “Btn press” on the press of a button

Red Capes

Basic Vue Native

```HTML
<template>
  <view class="container">
    <text class="text-color-primary">{{ message }}</text>
    <button title="Press me!" @press="exclaim" />
  </view>
</template>
```

Components: view

- ✓ Used to separate content or style different parts of the app in different ways
- ✓ Kind of equivalent to <*div*> in HTML

Red Capes

Basic Vue Native

Text

A Vue Native component for displaying text. The Text component supports nesting, styling, and touch handling.

HTML

```
<template>
    <view class="text-container">
        <text>Hello World</text>
    </view>
</template>
```

```
<style>
.text-container {
  flex: 1;
  margin-bottom: 30;
}
</style>
```

Components: text

- ✓ Simple component almost exclusively used for containing text blocks

Red Capes

Basic Vue Native

Image

A Vue Native component for displaying different types of images, including network images, static resources, temporary local images, and images from local disk, such as the camera roll.

Here we can see that we are dynamically setting the `source` attribute of each image using the `v-bind` shorthand syntax " `:` ". We also `v-bind` to the `style` property and pass in an object with a couple key-value pairs of styles we want to apply.

HTML

```
<template>
    <view>
        <image
          :source="require('/vue-native/img/favicon.png')"
        />
        <image
          :style="{width: 50, height: 50}"
          :source="{uri: 'https://facebook.github.io/react-native/docs/assets/fav
        />
        <image
          :style="{width: 66, height: 58}"
          :source="{uri: '
        />
    </view>
</template>
```

Components: image

- ✓ Simple component used for showing images
- ✓ Image can be an online resource or a locally stored image from the app
  - ✓ Online resources will, of course, require internet access for the user

Red Capes

Basic Vue Native

TextInput

A foundational component for inputting text into the app via a keyboard. Props provide configurability for several features, such as auto-correction, auto-capitalization, placeholder text, and different keyboard types, such as a numeric keypad.

In this example, we again take advantage of the `v-bind` directive, as well as a new one. The `v-model` directive. We will come back to this directive in just a bit, but for now just know that `v-model` directive allows us to set up an awesome concept known as `Two-Way Data Binding`.

HTML

```
<template>
    <text-input
        :style="{height: 40, width: 100, borderColor: 'gray', borderWidth: 1}"
        v-model="text"
      />
</template>
```

JS

```
<script>
    export default {
        data: function() {
            return {
                text: ''
            };
        }
    }
</script>
```

Components: text input

- ✓ Simple component used for getting input from the user
- ✓ Use v-model to connect the input to a data variable

Red Capes

Basic Vue Native

Button

A basic button component that should render nicely on any platform. Supports a minimal level of customization.
We can do more than just bind to data and style views with Vue Directives.

This button has an event handler `on-press` that we are binding to. When that event gets fired, we run a method on our Vue Instance.

HTML

```
<template>
    <button
        :on-press="onPressLearnMore"
        title="Learn More"
        color="#841584"
        accessibility-label="Learn more about this purple button"
    />
</template>
```

JS

```
<script>
export default {
    methods: {
        onPressLearnMore: function() {
            alert('Hello')
        }
    }
}
</script>
```

Components: button

- ✓ Simple component used for getting touch input from the user
- ✓ Use *<touchable-opacity>* for a more style:able component

Red Capes

Basic Vue Native

```
<template>
<touchable-opacity class="menu-button">
    <text class="menu-button-text" @press="onPress">{{text}}</text>
  </touchable-opacity>
</template>

<script>
export default {
  props: {
    text: {type: String},
    onPress: {type: Function}
  }
};
</script>

<style>
.menu-button {
  font-size: 25;
  border-color: rgb(0, 47, 255);
  background-color: rgb(94, 123, 255);
  border-style: solid;
  border-radius: 5;
  border-width: 2;
  padding: 10px;
  width: 95%;
  text-align: center;
  color: white;
  margin-bottom: 10;
}
.menu-button-text {
  font-size: 25;
  width: 100%;
  text-align: center;
  color: white;
}
</style>
```

Components: touchable opacity

- ✓ Simple component used for getting touch input from the user
- ✓ Use instead of <*button*> for a more style:able component

Basic Vue Native

Components

Find more components at: vue-native.io/docs/basic-components.html

Red Capes

Basic Vue Native

```HTML
<view>
  <text v-if="seen">Now you see the first one.</text>
  <text v-else>Now you see the second one.</text>
  <button :on-press="toggleSeen">Click to Toggle</button>
</view>
```

```HTML
<script>
export default {
  data() {
    return {
      seen: false
    };
  },
  toggleSeen() {
    this.seen = !this.seen;
  }
};
</script>
```

Conditionals (v-if & v-else)

- ✓ Hide or show a component based on a true or false statement

Red Capes

Basic Vue Native

```HTML
<view class="container">
  <text class="text-container" v-for="todo in todos" :key="todo.text">
    {{ todo.text }}
  </text>
</view>
```

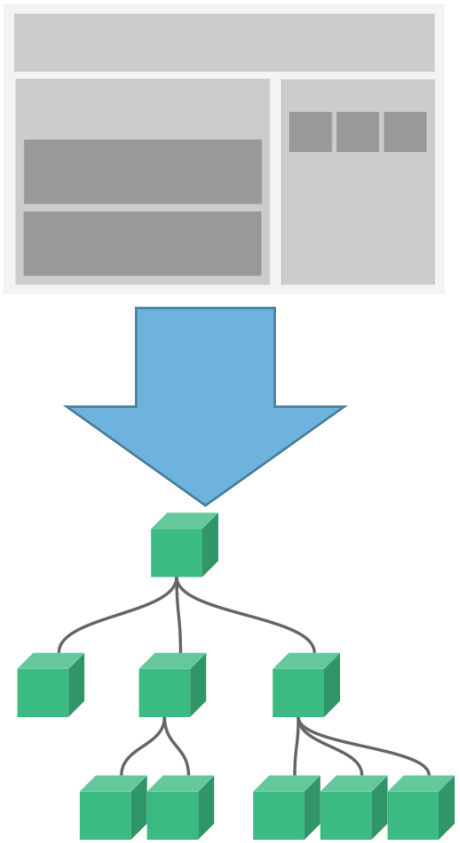
```JS
<script>
export default {
  data: function() {
    return {
      todos: [
        { text: "Learn JavaScript" },
        { text: "Learn Vue" },
        { text: "Build something awesome" }
      ]
    };
  }
};
</script>
```

Loops (v-for)

- ✓ Show the same component multiple times.
- ✓ Once for each element in an array

Red Capes

Basic Vue Native



Custom components

- ✓ Components can be nested to create advanced / complicated app structures

Red Capes

Basic Vue Native

Custom components

App.vue

```vue
<template>
  <view>

    <main-button
      v-for="menuButton in menuButtons"
      :key="menuButton.text"
      :text="menuButton.text"
      :onPress="menuButton.onClick"
    />

  </view>
</template>

<script>
import MainButton from "./../components/MainButton";

export default {
  components: { MainButton },

  data() {
    return {
      isLoading: true,
      textContent: "",
      growth: 0,

      menuButtons: [
        {
          text: "Button 1",
          onClick: () => {
            console.log("Button 1 clicked");
          },
        },
        {
          text: "Button 2",
          onClick: () => {
            this.navigation.navigate("Button 2 clicked");
          },
        }
      ],
    };
  }
};
</script>
```

MainButton.vue

```vue
<template>
  <touchable-opacity class="menu-button">
    <text class="menu-button-text" @press="onPress">{{text}}</text>
  </touchable-opacity>
</template>

<script>
export default {
  props: {
    text: { type: String },
    onPress: { type: Function },
  },
};
</script>

<style>
.menu-button {
  font-size: 25;
  border-color: rgb(0, 47, 255);
  background-color: rgb(94, 123, 255);
  border-style: solid;
  border-radius: 5;
  border-width: 2;
  padding: 10px;
  width: 95%;
  text-align: center;
  color: white;
  margin-bottom: 10;
}
.menu-button-text {
  font-size: 25;
  width: 100%;
  text-align: center;
  color: white;
}
</style>
```