Pixelmatters

pixelmatters_ wearepixelmatters pixelmatters

Tiago Coelho
CTO
@tiagofscoelho

SINF
October 30, 2019
FEUP

🔍 Who we are

Pixelmatters is a ~6 year old **digital product Design and Development** company based in Porto 🇵🇹 and working in-house, with clients worldwide.

🚀 Our mission

To create top-notch digital products that make make people's life easier, everywhere, everyday.

Pixelmatters

1st office

17 / 10 / 2013

—

Team-size: **1**

Pixelmatters

5th office

Today

Current team-size: **40**

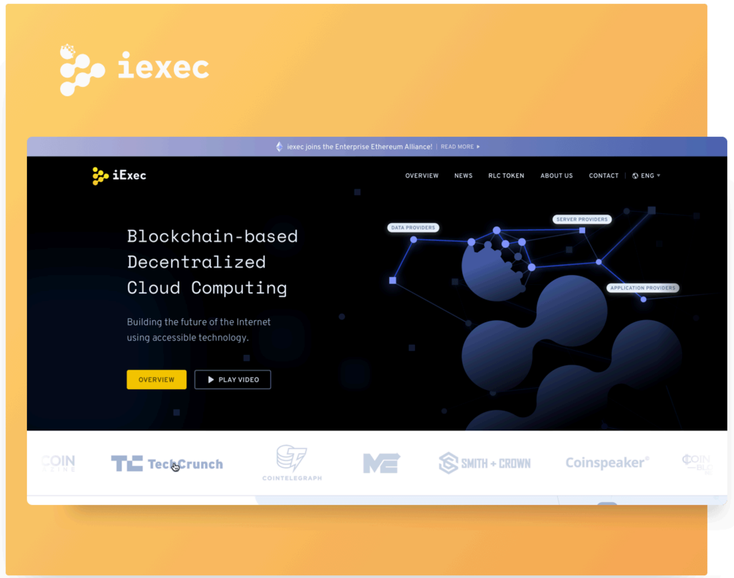
Pixelmatters

About our projects

A team of ~30, working mostly on three types of projects:

- **Marketing / Corporate websites;**
- **WebApps;**
- **Mobile Apps.**

Pixelmatters

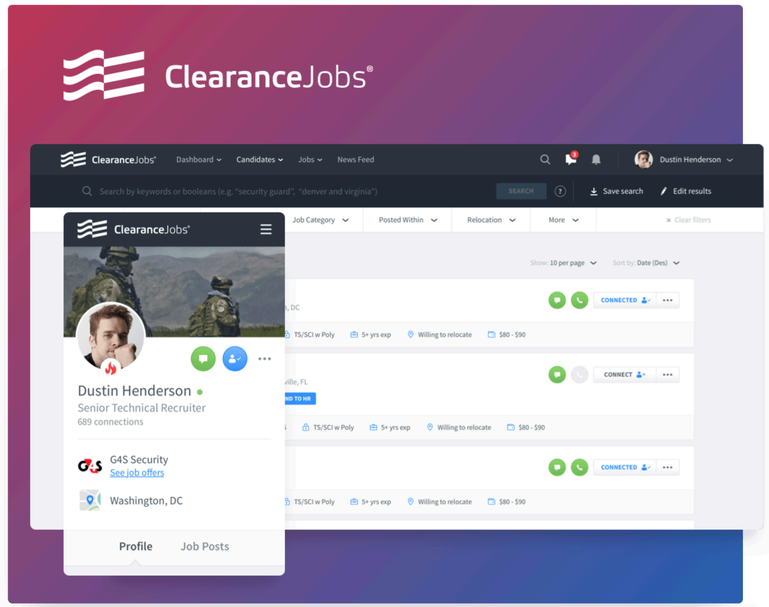


Blockchain Decentralized Cloud Computing →

Corporate Website

Competitive Gaming Platform →

iOS App | Android App



Security Jobs Network →

WebApp

Lots of **diversity** within the projects due to the nature of the business.

Pixelmatters

About our clients

Majority of our clients are **US-based** 🇺🇸 technology companies that, normally, have their own design, development, management and marketing teams internally. **We're their partners.**

Some of these companies are backed by well-known VC firms such as:

ANDREESSEN HOROWITZ SEQUOIA greylockpartners. G/

Pixelmatters

We’re an extension of our client’s team

Our goal is always to create **long-lasting relationships**.

To be the Client’s **partner**, and not only a services provider, for all things related with Design, Development and Management.

Pixelmatters

About our

mindset

Communication

Communication is the **key and central part of everything.** Our team has direct contact with the client - no “middle-man”.

Tools and **technologies** we work with.

Basecamp
slack
Pixelmatters
Trello
GitHub
Bitbucket
Jira
Confluence

Sketch
InVision
Zeplin

Vue.js
React
Swift
Kotlin
React Native
node.js
python
django CMS
WordPress
Gatsby

Pixelmatters

About our culture

Quality driven, in both design and development.

No-bullshit communication.

Honest feedback is key to evolution.

- **Work from Home** days available
- Friendly **student-worker** policy
- **Performance Reviews** regularly
- Always listening to your **needs and ambitions**, adjusting projects/clients/technologies

Pixelmatters

Inspiring environment and great location **@ Aliados**

- **Teamatters:** our yearly retreat.
- **Designmatters, Codematters:** internal event, once every 2-weeks to foster the team to share knowledge internally.

Pixelmatters

To finish

Careers

- **UX/UI** Designers
- **Front-End** Developers
- **Mobile** Developers
- **Full-Stack** Developers
- **Technical** Project Manager(s)
- **Delivery** Manager(s)

Pixelmatters means: do things well.

Codematters

React Hooks

About

Hooks

Hooks were **released in React 16.8**

~February 2019

Pixelmatters

Stateful components without writing a class

Backwards-compatible which allow a

Gradual Adoption Strategy

About the

Motivation

It's hard to **reuse** stateful logic between components without hooks. To do so, there are some techniques as:

👉 Render Props

👉 Higher-Order Components

Complex components become hard to understand

Classes confuse both people and machines

👉 People must understand how **this** works in Javascript.

👉 Classes don’t minify very well, they make hot unreliable, ahead-of-time compilation problems, etc.

Pixelmatters

About the Rules

Only Call Hooks from **React Functions**

👉 Call Hooks from React function components

👉 Call Hooks from custom Hooks

Only call Hooks at the **Top Level** of your function

👉 Don’t call Hooks inside **loops**, **conditions**, or **nested functions**.

Let’s see some

Code

Pixelmatters

```jsx
import React from 'react';

export default class TodoList extends React.Component {
    constructor() {
      super()
      this.state = {
        inputValue: ''
      }
    }

    render() {
      return (
        <>
          <input
            type="text"
            value={this.state.inputValue}
            onChange={e => {
              this.setState({
                inputValue: e.target.value
              })
            }
          />
        </>
      )
    }
}
```

“useState” Hook

***useState** Hook allow us to **declare a state variable** and provides us a **method to update** the it.*

```
const [inputValue, setInputValue] = useState('');
```

“useState” Hook

***useState** Hook allow us to **declare a state variable** and provides us a **method to update** the it.*

```
const [inputValue, setInputValue] = useState('');
```

```
this.state = {
  inputValue: ''
}
```

Pixelmatters

“useState” Hook

__useState__ Hook allow us to __declare a state variable__ and provides us a __method to update__ the it.

```
const [inputValue, setInputValue] = useState('');
```

```
this.state = {
  inputValue: ''
}
```

“useState” Hook

***useState** Hook allow us to **declare a state variable** and provides us a **method to update** the it.*

```
const [inputValue, setInputValue] = useState('');
```

```
this.state = {
  inputValue: ''
}
```

“useState” Hook

__useState__ Hook allow us to __declare a state variable__ and provides us a __method to update__ the it.

```
const [inputValue, setInputValue] = useState('');
```

```
this.setState({
  inputValue: 'new value'
})
```

“useState” Hook

__useState__ Hook allow us to __declare a state variable__ and provides us a __method to update__ the it.

```
const [inputValue, setInputValue] = useState('');
```

```
setInputValue('new value')
```

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

export default () =>  {
    const [inputValue, setInputValue] = useState('');

    return (
      <>
        <input
          type="text"
          value={inputValue}
          onChange={
            event => setInputValue(event.target.value)
          }
        />
      </>
    )
}
```

Pixelmatters

```jsx
import React from 'react';

export default class Input extends React.Component {
    constructor() {
      super()
      this.state = {
        inputValue: ''
      }
    }

    render() {
      return (
        <>
          <input
            type="text"
            value={this.state.inputValue}
            onChange={event =>
              this.setState({
                inputValue: event.target.value
              })
            }
          />
        </>
      )
    }
}
```

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

export default () =>  {
    const [inputValue, setInputValue] = useState('');

    return (
      <>
        <input
          type="text"
          value={inputValue}
          onChange={
            event => setInputValue(event.target.value)
          }
        />
      </>
    )
}
```

Pixelmatters

```jsx
import React from 'react';

export default class Component extends React.Component {
    constructor() {
      super()
      this.state = {
        inputValue: ''
      }
    }

    componentDidMount() {
      document.title = this.state.inputValue
    }

    componentDidUpdate () {
      document.title = this.state.inputValue
    }

    render () {
       return (
         <>
           <input
             type="text"
             value={this.state.inputValue}
             onChange={event ⇒
               this.setState({
                 inputValue: event.target.value
               })
             }
           />
         </>
       )
    }
}
```

Pixelmatters

“useEffect” Hook

useEffect *is a lifecycle Hook for that combines* ***componentDidMount***, ***componentDidUpdate*** *and* ***componentWillUnmount*** *lifecycle methods.*

```
const [inputValue, setInputValue] = useState('')
useEffect(() => {
  // componentDidMount and componentDidUpdate

  return () => {
    // componentWillUnmount
  }
})
```

“useEffect” Hook

useEffect *is a lifecycle Hook for that combines* ***componentDidMount***, ***componentDidUpdate*** *and* ***componentWillUnmount*** *lifecycle methods.*

```
const [inputValue, setInputValue] = useState('')
useEffect(() => {
  // componentDidMount
  return () => {
    // componentWillUnmount
  }
}), [inputValue]) // componentDidUpdate called only when “inputValue" changes
```

“useEffect” Hook

__useEffect__ is a lifecycle Hook for that combines __componentDidMount__, __componentDidUpdate__ and __componentWillUnmount__ lifecycle methods.

```
const [inputValue, setInputValue] = useState('')
useEffect(() => {
  // componentDidMount

  return () => {
    // componentWillUnmount
  }
}, []) // componentDidUpdate is not called
```

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

export default () =>  {
    const [inputValue, setInputValue] = useState('')

    useEffect(() => {
      document.title = inputValue
    }, [inputValue])

    return (
      <>
        <input
          type="text"
          value={inputValue}
          onChange={event => setInputValue(event.target.value)}
        />
      </>
    )
}
```

Pixelmatters

```jsx
import React from 'react';

export default class Component extends React.Component {
    constructor() {
      super()
      this.state = {
        inputValue: ''
      }
    }

    componentDidMount() {
      document.title = this.state.inputValue
    }

    componentDidUpdate () {
      document.title = this.state.inputValue
    }

    render () {
       return (
        <>
          <input
            type="text"
            value={this.state.inputValue}
            onChange={event =>
              this.setState({
                inputValue: event.target.value
              })
            }
          />
        </>
      )
    }
}
```

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

export default () =>  {
    const [inputValue, setInputValue] = useState('')

    useEffect(() => {
      document.title = inputValue
    }, [inputValue])

    return (
      <>
        <input
          type="text"
          value={inputValue}
          onChange={event => setInputValue(event.target.value)}
        />
      </>
    )
}
```

Pixelmatters

```jsx
import React from 'react';

export default class Component extends React.Component {
    constructor() {
      super()
      this.state = {
        inputValue: ''
      }
    }

    componentDidMount() {
      document.title = this.state.inputValue
    }

    componentDidUpdate () {
      document.title = this.state.inputValue
    }

    render () {
       return (
        <>
          <input
            type="text"
            value={this.state.inputValue}
            onChange={event =>
              this.setState({
                inputValue: event.target.value
              })
            }
          />
        </>
      )
    }
}
```

34 lines

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

export default () =>  {
    const [inputValue, setInputValue] = useState('')

    useEffect(() => {
      document.title = inputValue
    }, [inputValue])

    return (
      <>
        <input
          type="text"
          value={inputValue}
          onChange={event => setInputValue(event.target.value)}
        />
      </>
    )
}
```

19 lines

Pixelmatters

“useReducer” Hook

useReducer *allow us to manage complex states. Everyone with Redux experience will notice similarities with the syntax of this Hook.*

```
const todosReducer = (todos, action) => {
  let newTodos = todos
  switch (action.type) {
    case 'add':
      newTodos = [...todos, { text: action.text, isCompleted: false }];
      break
    case 'done':
      newTodos = [...todos];
      newTodos[action.index].isCompleted = true;
      break
    case 'undone':
      newTodos = [...todos];
      newTodos[action.index].isCompleted = false;
      break
    case 'remove':
      newTodos = [...todos];
      newTodos.splice(action.index, 1);
      break
    default:
      return newTodos
  }

  return newTodos
}
```

Pixelmatters

“useReducer” Hook

useReducer *allow us to manage complex states. Everyone with Redux experience will notice similarities with the syntax of this Hook.*

```
const todosReducer = (todos, action) => {
  let newTodos = todos
  switch (action.type) {
    case 'add':
      newTodos = [...todos, { text: action.text, isCompleted: false }];
      break
    case 'done':
      newTodos = [...todos];
      newTodos[action.index].isCompleted = true;
      break
    case 'undone':
      newTodos = [...todos];
      newTodos[action.index].isCompleted = false;
      break
    case 'remove':
      newTodos = [...todos];
      newTodos.splice(action.index, 1);
      break
    default:
      return newTodos
  }

  return newTodos
}
```

Usage

```
const [inputValue, setInputValue] = useState('');
const [todos, dispatchTodos] = useReducer(todosReducer, []);
dispatchTodos({type: 'add', text: inputValue});

dispatchTodos({type: todo.isCompleted ? 'undone' : 'done', index})

dispatchTodos({type: 'remove', index})
```

Built-in Hooks

*This slide lists the APIs for the **built-in Hooks** in React.*

Basic Hooks

- **useState**
- **useEffect**
- useContext

Additional Hooks

- **useReducer**
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

Pixelmatters

Custom Hooks

*Building **your own** Hooks let you extract component's logic into reusable functions.*

https://github.com/rehooks/awesome-react-hooks

Demo

Todo Application with Hooks

Thanks.

pixelmatters_ wearepixelmatters pixelmatters

Tiago Coelho
CTO
@tiagofscoelho

SINF
October 30, 2019
FEUP