Increase the Performance of your Site with Lazy-Loading and Code-Splitting

About me

👋 I'm Jose and I work as Engineering Manager in Spotify

I like sites (building/using) with good performance

@jmperezperez on Twitter

The era of components

What we'll talk about

Compositional Patterns

High Order Components

```
const MyComponent = props => (
  <div>
    {props.id} - {props.name}
  </div>
);

// ...

const ConnectedComponent = connect(
  mapStateToProps,
  mapDispatchToProps
)(MyComponent);
```

Function as Child Component aka Render Callback

```
const MyComponent = () => (
  <Media query="(max-width: 599px)">
    {matches =>
      matches ? (
        <p>The document is less than 600px wide.</p>
      ) : (
        <p>The document is at least 600px wide.</p>
      )
    }
  </Media>
);
```

Improving performance of our sites by loading only what is needed

Most times you would include all the scripts and CSS needed to render all sections as soon as the user visits the page. Until recently it was difficult to define a module’s dependencies, and load what was needed.

- How likely is it for the user to see the header?
- What about the map?

- Yahoo's YUI Loader
- Facebook's Haste, Bootloader and Primer

Lazy-Loading has trade-offs too

Invisible content in some scenarios

- printing the page
- RSS readers
- SEO

A small component to detect when an area is visible

Function as Child Component aka Render Callback

```
class Observer extends Component {
  constructor() {
    super();
    this.state = { isVisible: false };
    this.io = null;
    this.container = null;
  }
  componentDidMount() {
    this.io = new IntersectionObserver([entry] => {
      this.setState({ isVisible: entry.isIntersecting });
    }, {});
    this.io.observe(this.container);
  }
  componentWillUnmount() {
    if (this.io) {
      this.io.disconnect();
    }
  }
  render() {
    return (
      // we create a div to get a reference.
      // It's possible to use findDOMNode() to avoid
```

Using it

```
const Page = () => {
  <div>
    <Header />
    <Observer>{isVisible => <Gallery isVisible />}</Observer>
    <Observer>{isVisible => <Map isVisible />}</Observer>
    <Footer />
  </div>;
};
```

make sure that you reserve the area for the lazy-loaded component

```
class Map extends Component {
  constructor() {
    super();
    this.state = { initialized: false };
    this.map = null;
  }

  initializeMap() {
    this.setState({ initialized: true });
    // loadScript loads an external script, its definition is not included here.
    loadScript('https://maps.google.com/maps/api/js?key=<your_key>', () => {
      const latlng = new google.maps.LatLng(38.34, -0.48);
      const myOptions = { zoom: 15, center: latlng };
      const map = new google.maps.Map(this.map, myOptions);
    });
  }

  componentDidMount() {
    if (this.props.isVisible) {
      this.initializeMap();
    }
  }

  componentWillReceiveProps(nextProps) {
    if (!this.state.initialized && nextProps.isVisible) {
```

```
class Gallery extends Component {
  constructor() {
    super();
    this.state = { hasBeenVisible: false };
  }
  componentDidMount() {
    if (this.props.isVisible) {
      this.setState({ hasBeenVisible: true });
    }
  }
  componentWillReceiveProps(nextProps) {
    if (!this.state.hasBeenVisible && nextProps.isVisible) {
      this.setState({ hasBeenVisible: true });
    }
  }
  render() {
    return (
      <div>
        <h1>Some pictures</h1>
        Picture 1
        {this.state.hasBeenVisible ? (
          <img src="http://example.com/image01.jpg" width="300" height="300" />
        ) : (
```

Stateless Child Components

```
const Gallery = ({ isVisible }) => (
  <div>
    <h1>Some pictures</h1>
    Picture 1
    {isVisible ? (
      <img src="http://example.com/image01.jpg" width="300" height="300" />
    ) : (
      <div className="placeholder" />
    )}
    Picture 2
    {isVisible ? (
      <img src="http://example.com/image02.jpg" width="300" height="300" />
    ) : (
      <div className="placeholder" />
    )}
  </div>
```

```
const Page = () => {
  ...
  <Observer>
    {(isVisible, hasBeenVisible) =>
      <Gallery hasBeenVisible /> // Gallery can be now stateless
    }
  </Observer>
  ...
}
```

```
constructor() {
  super();
  this.state = { hasBeenVisible: false };
  this.io = null;
  this.container = null;
}
componentDidMount() {
  this.io = new IntersectionObserver(entries => {
    entries.forEach(entry => {
      if (entry.isIntersecting) {
        this.setState({ hasBeenVisible: true });
        this.io.disconnect();
      }
    });
  }, {});
  this.io.observe(this.container);
}
componentWillUnmount() {
  if (this.io) {
    this.io.disconnect();
  }
}
render() {
  return (
    <div
      ref={div => {
        this.container = div;
      }}
    >
      {Array.isArray(this.props.children)
        ? this.props.children.map(child => child(this.state.hasBeenVisible))
        : this.props.children(this.state.hasBeenVisible)}
    </div>
  );
}
```

More use cases

```
  super();
  this.state = { progress: 0 };
  this.interval = null;
  this.animationDuration = 2000;
  this.startAnimation = null;
}
componentWillReceiveProps(nextProps) {
  if (
    !this.props.isVisible &&
    nextProps.isVisible &&
    this.state.progress !== 1
  ) {
    this.startAnimation = Date.now();
    const tick = () => {
      const progress = Math.min(
        1,
        (Date.now() - this.startAnimation) / this.animationDuration
      );
      this.setState({ progress: progress });
      if (progress < 1) {
        requestAnimationFrame(tick);
      }
    };
    tick();
  }
}
render() {
  return (
    <div>
      {Math.floor(this.state.progress * 3)} days ·
      {Math.floor(this.state.progress * 21)} talks ·
      {Math.floor(this.state.progress * 4)} workshops ·
      {Math.floor(this.state.progress * 350)} attendees
    </div>
  );
```

Polyfilling IntersectionObserver on-demand

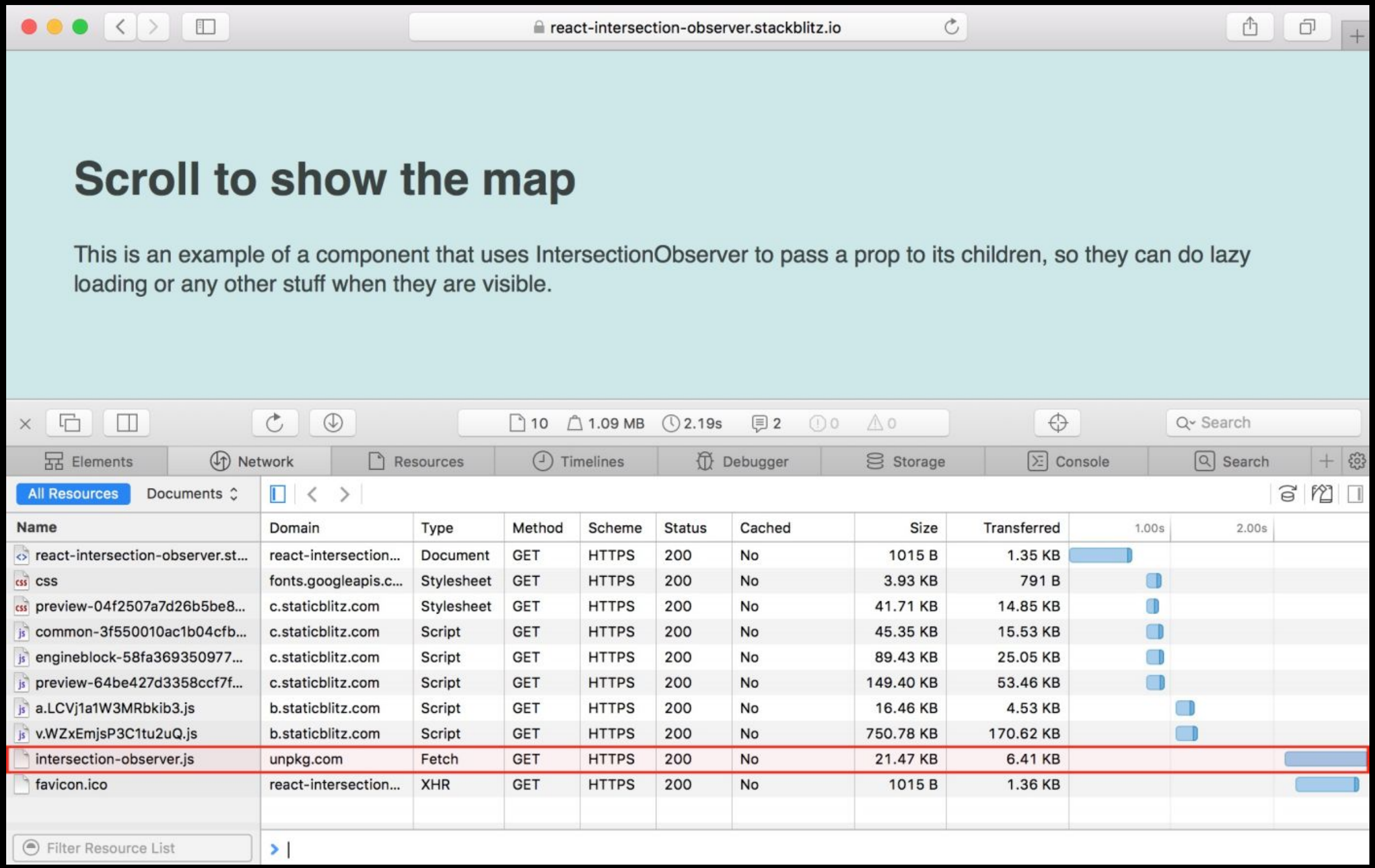
Disabling lazy-loading if IntersectionObserver is not supported

```
class Observer extends Component {
  constructor() {
    super();
    // isVisible is initialized to true if the browser
    // does not support IntersectionObserver API
    this.state = { isVisible: !(window.IntersectionObserver) };
    this.io = null;
    this.container = null;
  }
  componentDidMount() {
    // only initialize the IntersectionObserver if supported
    if (window.IntersectionObserver) {
      this.io = new IntersectionObserver(entries => {
        ...
      }
    }
  }
}
```

Requesting a polyfill on demand

```
class Observer extends Component {
  ...
  componentDidMount() {
    (window.IntersectionObserver
      ? Promise.resolve()
      : import('intersection-observer')
    ).then(() => {
      this.io = new window.IntersectionObserver(entries => {
        entries.forEach(entry => {
          this.setState({ isVisible: entry.isIntersecting });
        });
      }, {});
      this.io.observe(this.container);
    });
  }
  ...
}
```

Safari requests the polyfill for intersection-observer on demand.



No need to ship it to browsers that support it natively.

Code Splitting and CSS-in-JS

- react-router and Next.js have made code-splitting easy to implement
- lazy-loading can be applied to other resources (SVGs, CSS)
- With CSS-in-JS we take code splitting further, loading CSS on demand.

Useful implementations

- thebuilder/react-intersection-observer
- researchgate/react-intersection-observer

Conclusion

Componentization makes code-splitting and loading assets on-demand easier than ever!