Introduction to ReactJS, MongoDB

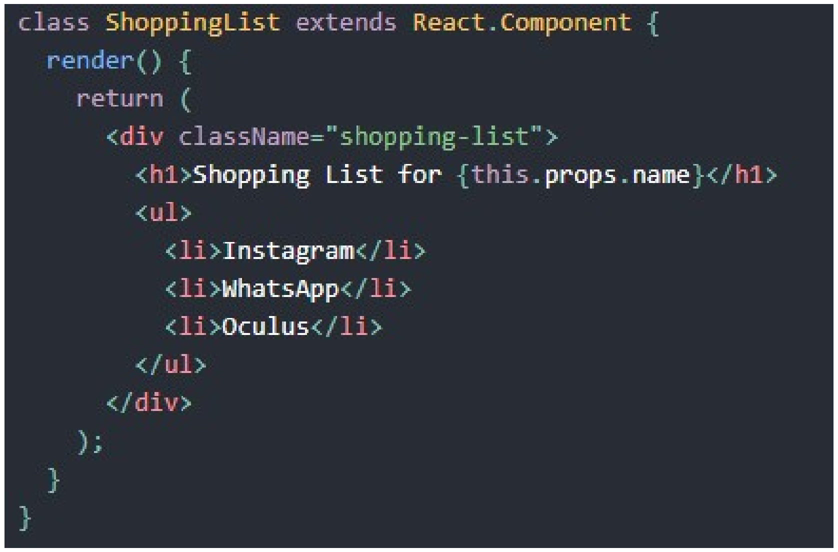
ReactJS setup

- Follow instructions for local installation at:
    - https://reactjs.org/tutorial/tutorial.html
- Create React App locally:
    - npm install -g create-react-app
    - npx create-react-app
- Follow instructions for creating a TicTacToe game frame

- Taken from: https://reactjs.org/tutorial/tutorial.html

What is React?

- React is a declarative, efficient, and flexible JavaScript library for building user interfaces.
- It lets you compose complex UIs from small and isolated pieces of code called “components”.



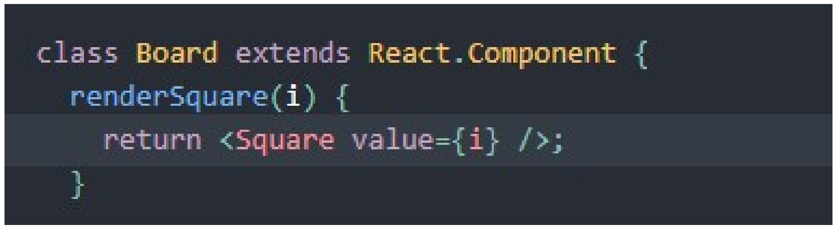
```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}
```

React Component Class

- Component tell React what we want to see on the screen
- A component takes in parameters called props
- Render method returns a description of what you want to see on the screen
- Render returns a React element that describes what to render
- React developers use JSX for writing structures to be rendered
- JSX is an XML/HTML-like syntax that allows us to put HTML into JavaScript
- React components can be composed and rendered as required
- Each React component is encapsulated and can operate independently
- This allows building complex UI4s from simple components

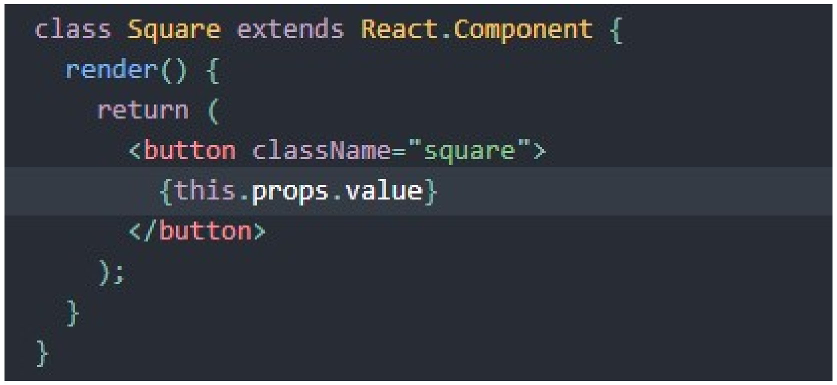
Passing Data Through Props

- Change the renderSquare method to pass a prop called value to the Square:



```
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={i} />;
  }
```

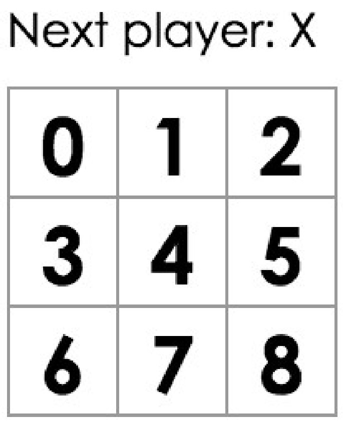
- Change Square's render method to show the square value:



```
class Square extends React.Component {
  render() {
    return (
      <button className="square">
        {this.props.value}
      </button>
    );
  }
}
```

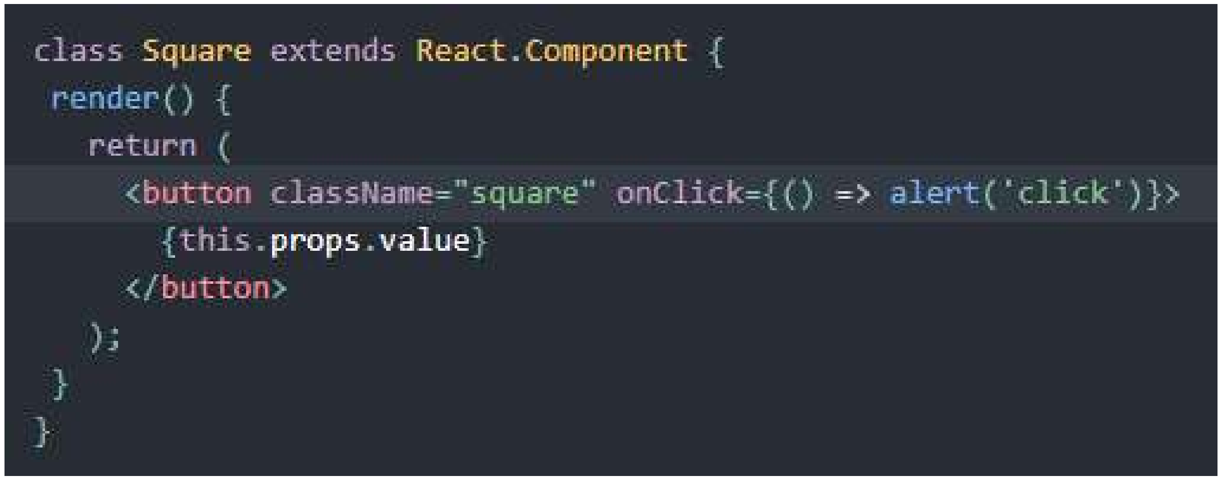
Result of passing a prop

- Refresh the browser to see the number in each square
- We passed a prop from a parent Board component to a Child square component
- Information flows in React apps by passing props from parents to children



Making an interactive Component

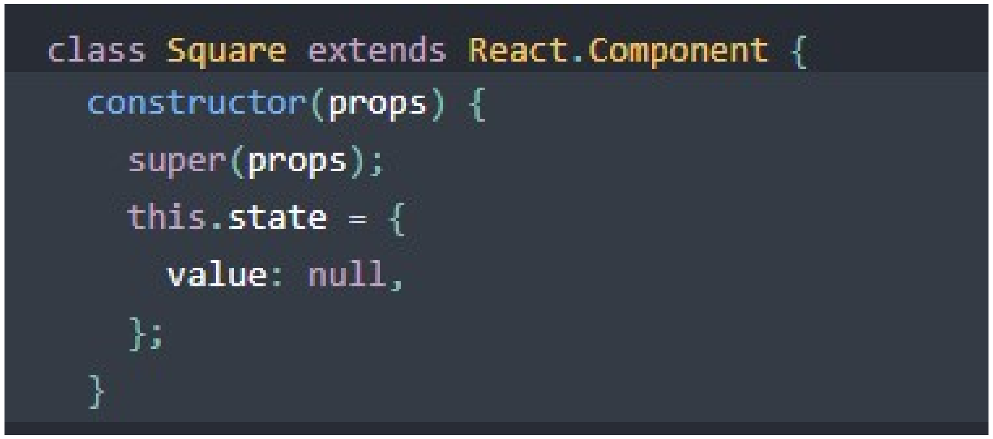
- Getting an alert
- Note the use of arrow function



```
class Square extends React.Component {
  render() {
    return (
      <button className="square" onClick={() => alert('click')}>
        {this.props.value}
      </button>
    );
  }
}
```

Using state to remember actions

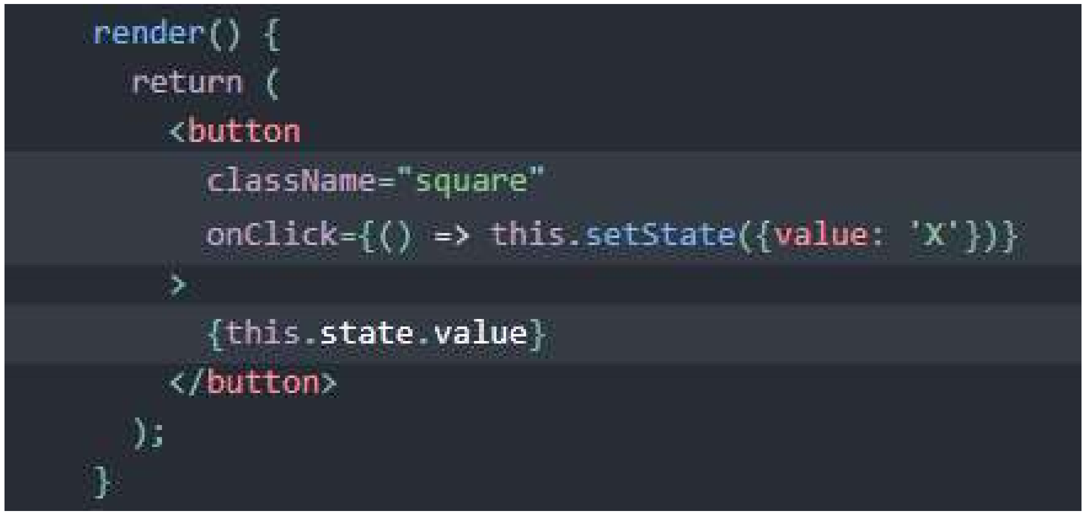
- State of a component is to be initialized in a constructor of a component
- State should be considered as private to a React component
- Add a constructor to the Square class to initialize state
- Following JavaScript guidelines, all React component classes with a constructor should have a super(props) call



```
class Square extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: null,
    };
  }
```

Changing the states

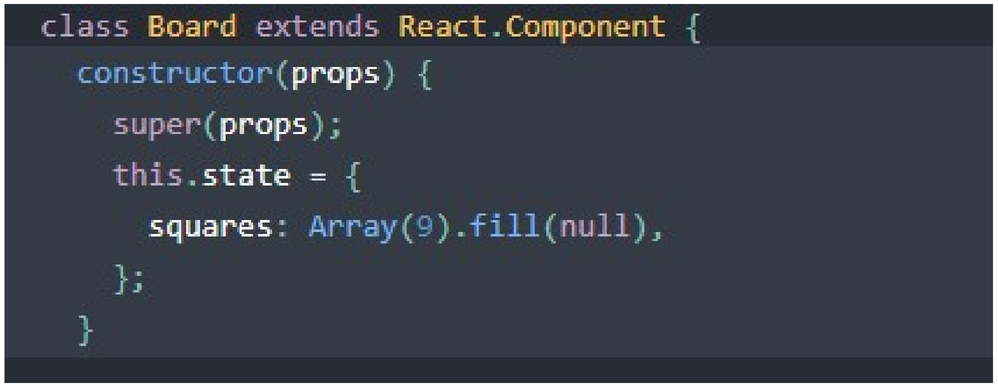
- Use this.setState from an onClickhandler in the render method for changing state of a square
- Calling a setState in a component will automatically update the child components inside it



```
render() {
  return (
    <button
      className="square"
      onClick={() => this.setState({value: 'X'})}
    >
      {this.state.value}
    </button>
  );
}
```

Lifting State Up

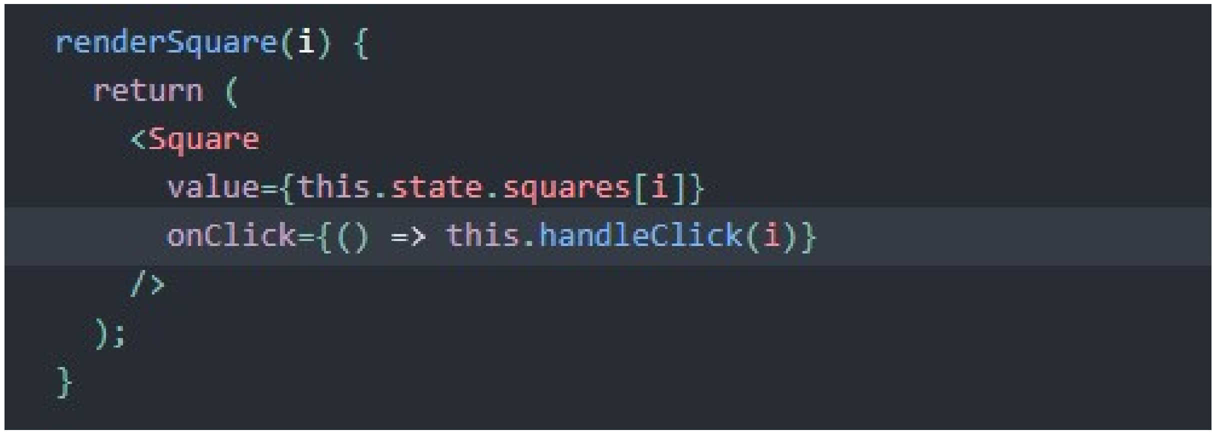
- To determine winner, the value of each of the 9 squares need to be in one location
- Best approach is to store game's state in the parent Board component
- Board tells each square what to display by passing a prop
- Ass a constructor to Board and set initial values with 9 nulls



```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
    };
  }
```

Passing States down to the Squares

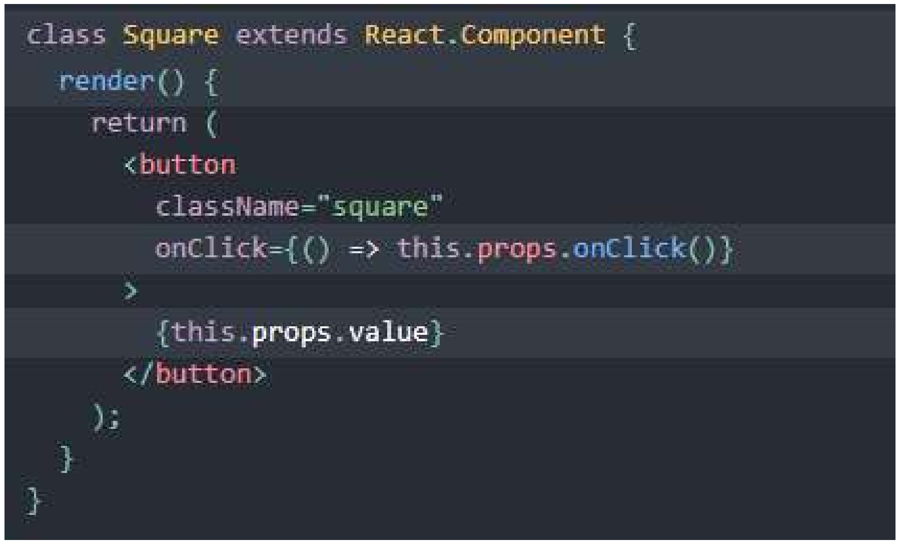
- Modify renderSquare method to read value from the board’s state
- Pass down a function which will get called when a Square is clicked



```
renderSquare(i) {
  return (
    <Square
      value={this.state.squares[i]}
      onClick={() => this.handleClick(i)}
    />
  );
}
```

Modify Square Class

- Board class passes down two props to Square: value and onClick
- Square doesn’t need to keep track of state now, so we can delete square state
- Also, delete square constructor and change render to the following:

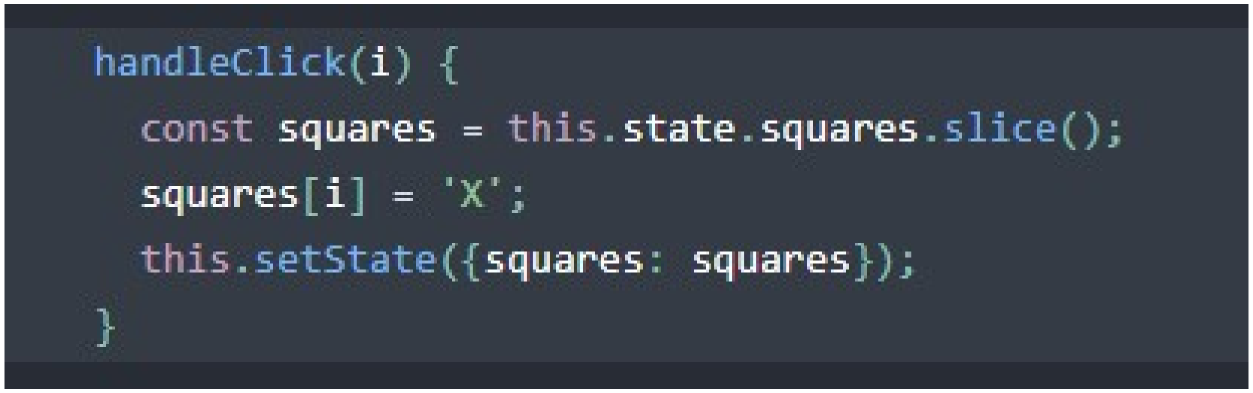


```
class Square extends React.Component {
  render() {
    return (
      <button
        className="square"
        onClick={() => this.props.onClick()}
      >
        {this.props.value}
      </button>
    );
  }
}
```

What will happen when a Square is clicked?

What will happen when a Square is clicked?

- We have not defined the handleClick() method yet, so our code crashes
- Add handleClick to the Board class



```
handleClick(i) {
  const squares = this.state.squares.slice();
  squares[i] = 'X';
  this.setState({squares: squares});
}
```

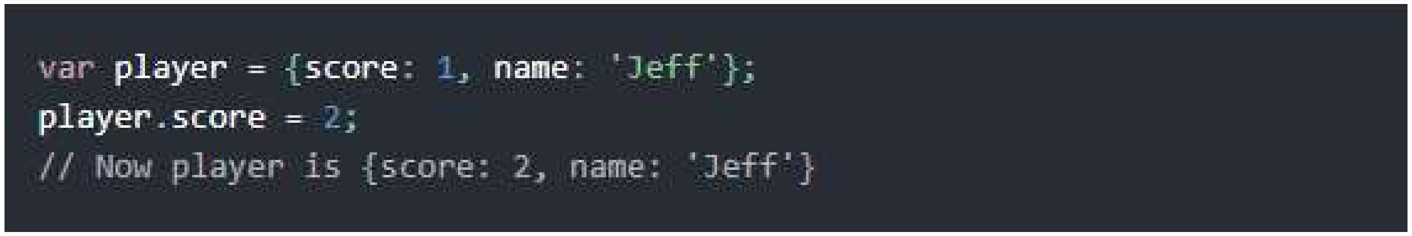
- The Square component receive values from Board component
- Square components are now controlled components

Mutability vs. Immutability

- We used the .slice() operator to create a copy of the squares array instead of modifying the existing array
- Mutation refers to changing data directly, other approach is replacing the data with a new copy
- Immutability allows us to implement ‘time travel’ – useful for undo and redo operations
- Detecting changes in the immutable objects is easier

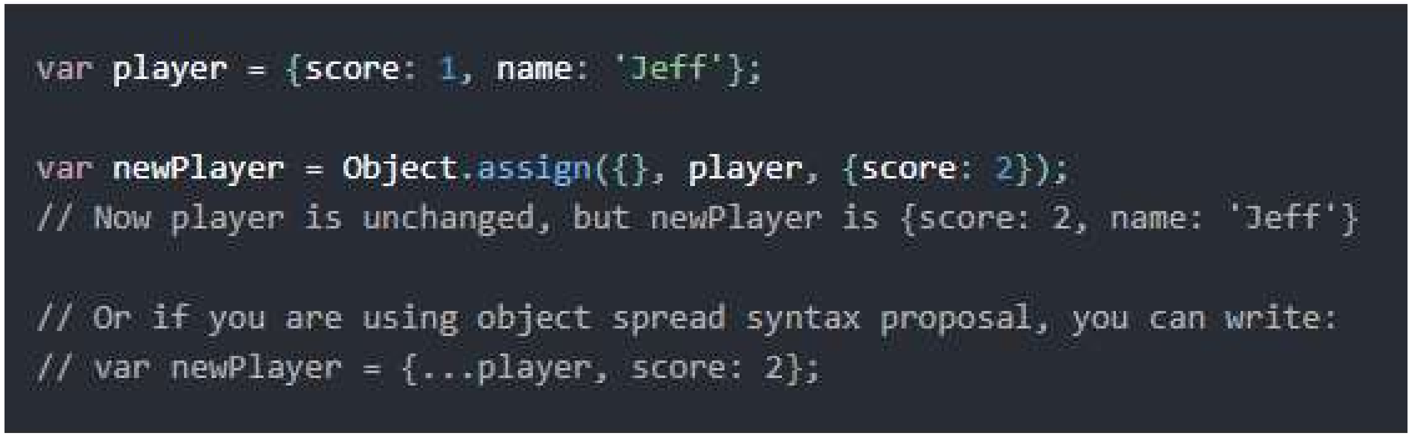
Mutability vs. Immutability

Data Change with Mutation



```
var player = {score: 1, name: 'Jeff'};
player.score = 2;
// Now player is {score: 2, name: 'Jeff'}
```

Data Change without Mutation



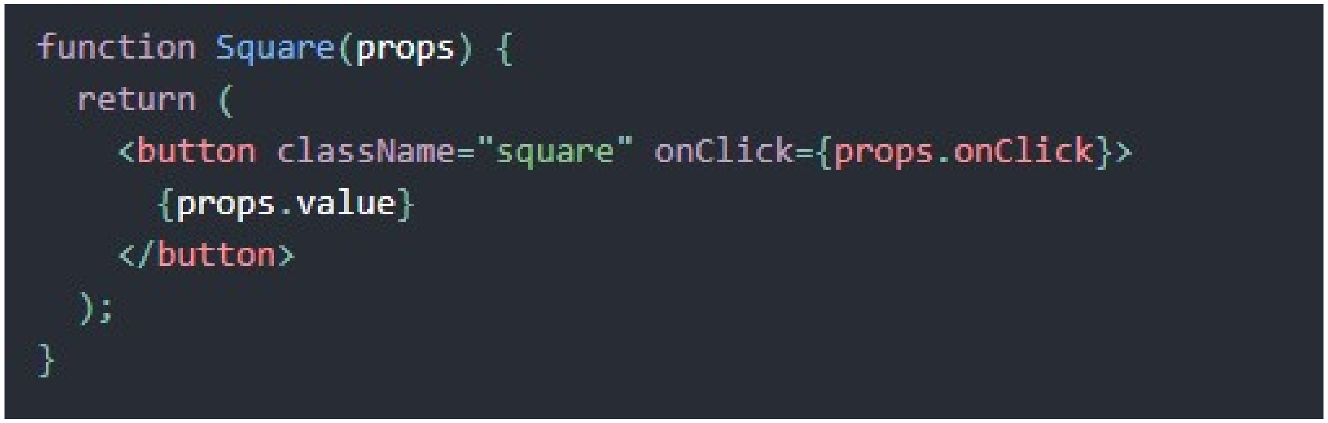
```
var player = {score: 1, name: 'Jeff'};

var newPlayer = Object.assign({}, player, {score: 2});
// Now player is unchanged, but newPlayer is {score: 2, name: 'Jeff'}

// Or if you are using object spread syntax proposal, you can write:
// var newPlayer = {...player, score: 2};
```

Function Components

- React classes which contain only render method and don’t have own state could be converted to function components
- Function takes props as input and returns what should be rendered
- Replace the Square class with a function

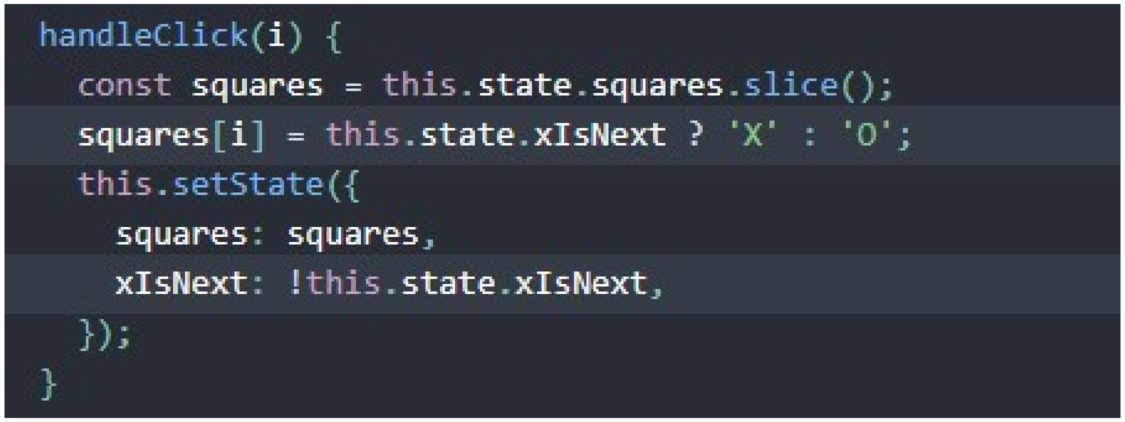


```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

Adding Logic to Take Turns

- First move is always X
  - Add xIsNext: true, to the Board state in it’s constructor
  - We will flip xIsNext to True or False depending on who is next: X or O

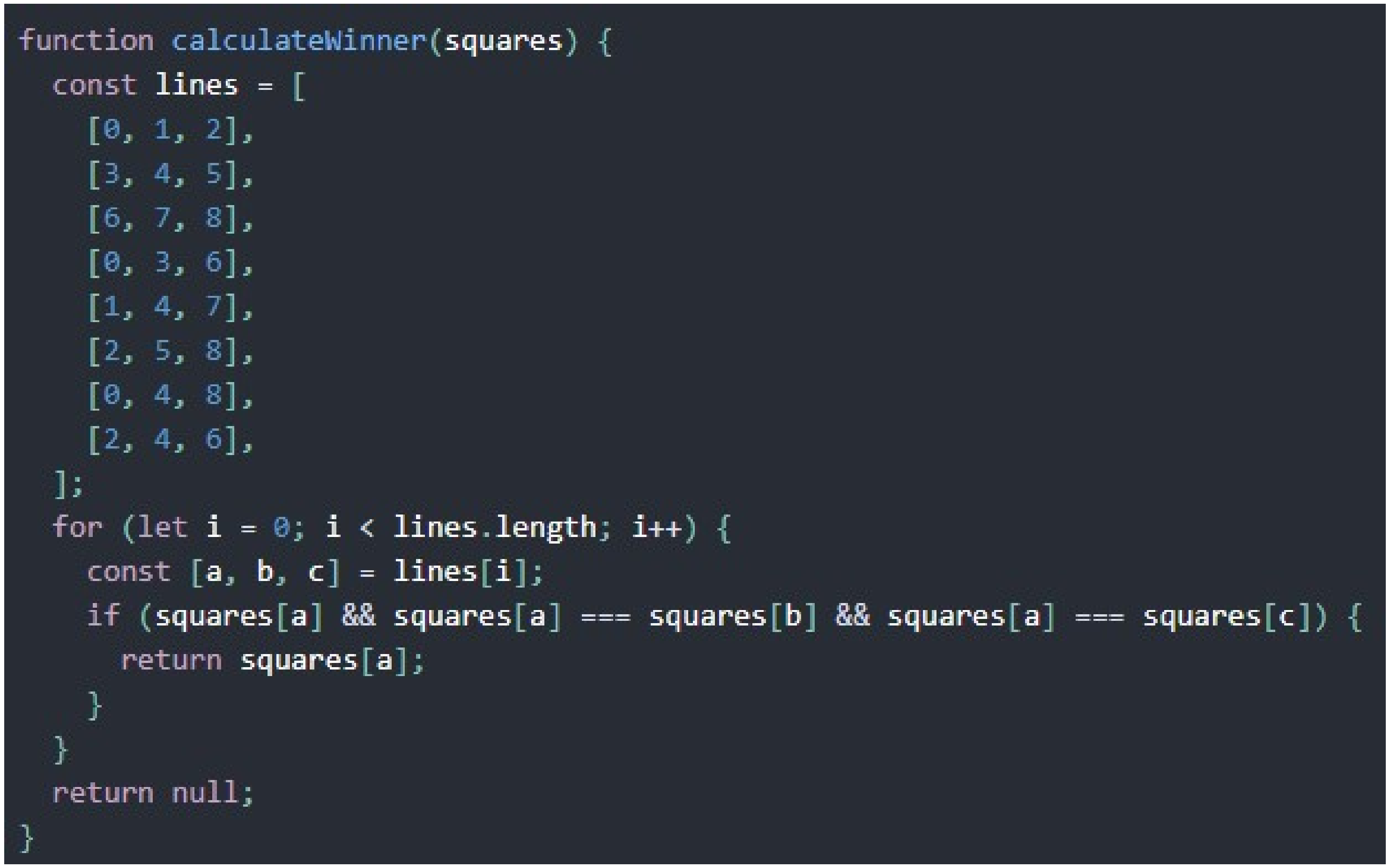
- Change handleClick function to change the value of squares and xIsNext depending on the turn



```
handleClick(i) {
  const squares = this.state.squares.slice();
  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}
```

- Change status text in Board’s render to display player with next turn
  - const status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O’);

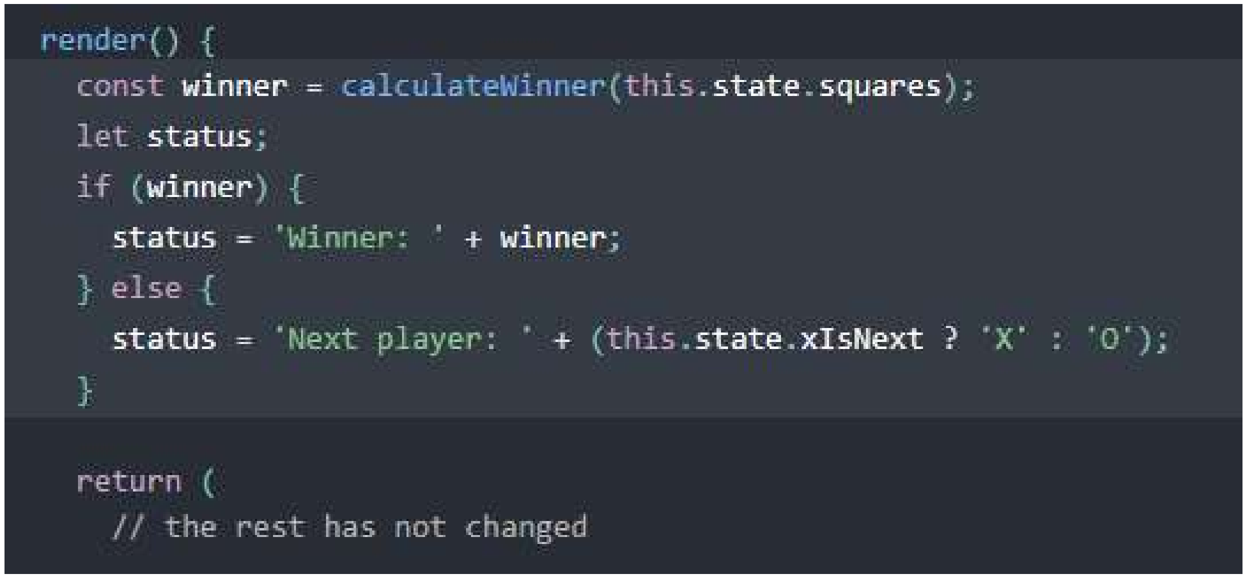
Calculating a Winner – Helper Function



```
function calculateWinner(squares) {
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
      return squares[a];
    }
  }
  return null;
}
```

Announcing a Winner

- Call calculateWinner(squares) in the Board’s render function
- If a player has won, display text such as “Winner: X” or “Winner: O”

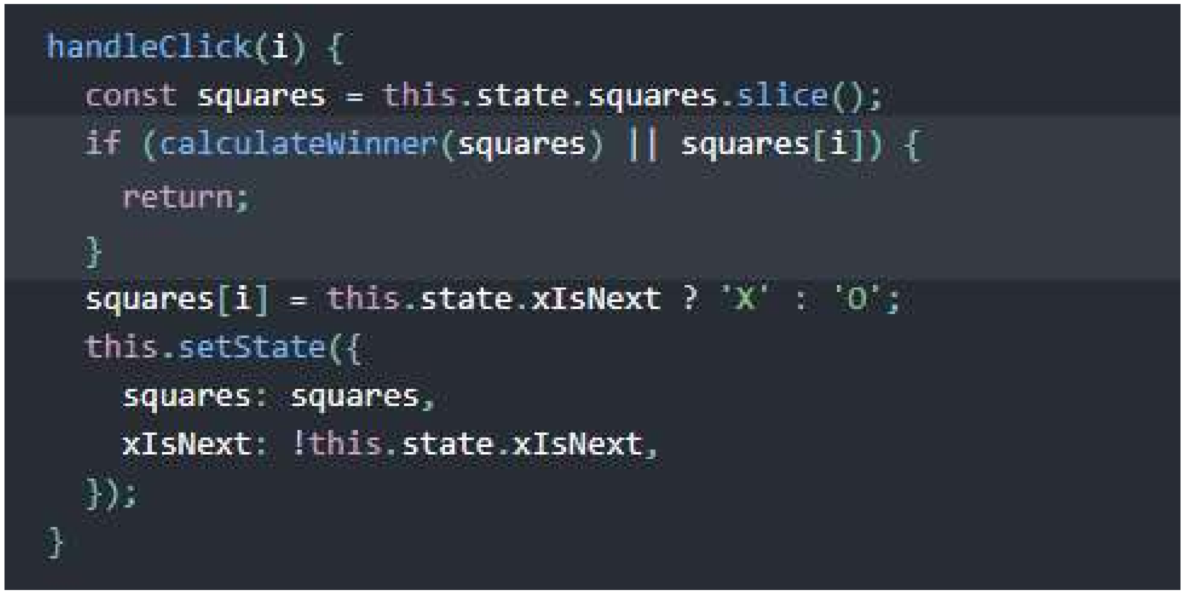


```
render() {
  const winner = calculateWinner(this.state.squares);
  let status;
  if (winner) {
    status = 'Winner: ' + winner;
  } else {
    status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O');
  }

  return (
    // the rest has not changed
```

Ignore Clicks if Game is Finished

- Change the Board’s handleClick function to return early by ignoring a click if someone has won the game or if a Square is already filled



```
handleClick(i) {
  const squares = this.state.squares.slice();
  if (calculateWinner(squares) || squares[i]) {
    return;
  }
  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}
```

Adding Time Travel

- Homework

Relational vs. Non Relational Databases (RDBMS vs NoSQL)

- Relational Databases
  - Example: Microsoft SQL Server, Oracle DB, DB2
  - Used in large enterprise scenarios

- Non-relational databases
  - Example: MongoDB, Cassandra
  - Four categories: Key-value stores, wide-column stores, Document stores and graph stores

Some of these slides taken from: https://www.pass.org/DownloadFile.aspx?File=0141eebe

What is NoSQL?

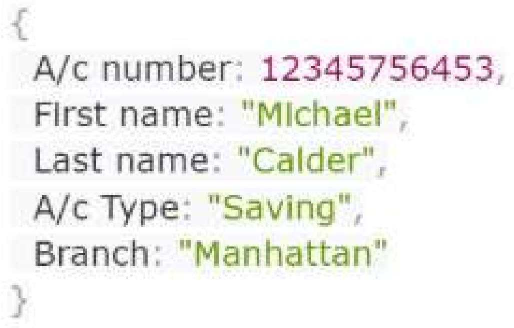
- NoSQL is a class of database management system identified by its non-adherence to the widely used relational database management system (RDBMS) model with its structured query language (SQL).
- NOSQL has evolved to mean “Not Only” SQL
- NOSQL has become prominent with the advent of web scale data and systems created by Google, Facebook, Amazon, Twitter and others to manage data for which SQL was not the best fit.

Relational Stores

- Data Stored in tables
- Tables contain some number of columns, each of a type
- A schema describes the columns each table can have
- Every table’s data is stored in one or more rows
- Each row contains a value for every column in that table
- Rows aren’t kept in any particular order

Key-Value Stores

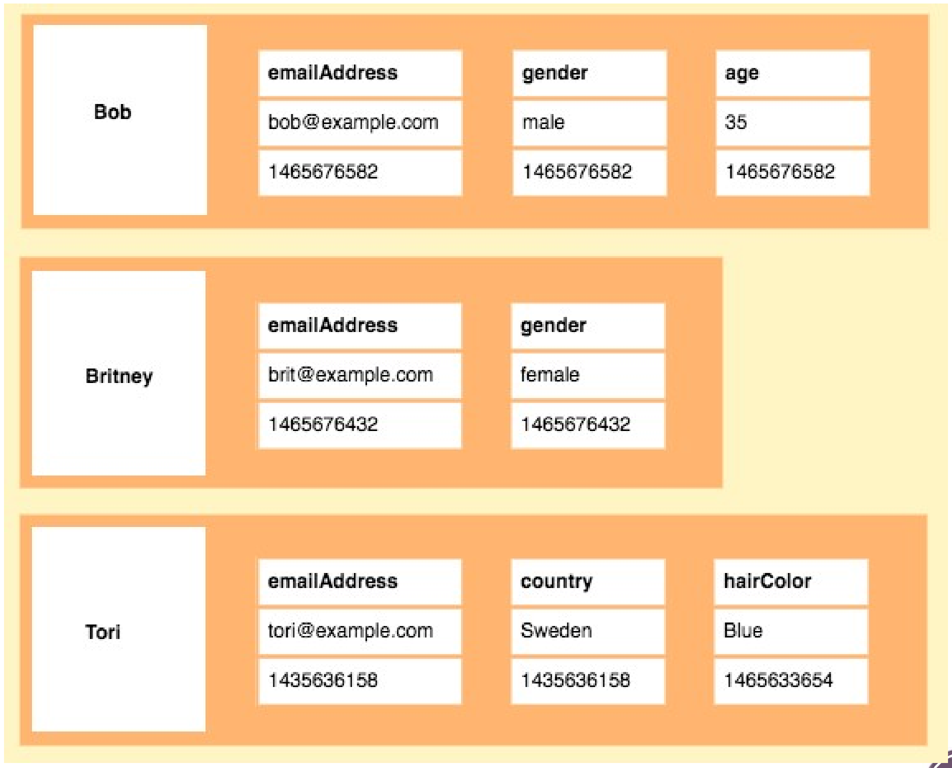
- Anything can be stored as a value, as long as each value is associated with a key or a name
- Key-value stores offer very high speed via the least complicated data model
- The basic data structure is a dictionary or map. You can store a value, such as an integer, string, a JSON structure, or an array, along with a key used to reference that value.



```
{
 A/c number: 12345756453,
 First name: "MIchael",
 Last name: "Calder",
 A/c Type: "Saving",
 Branch: "Manhattan"
}
```

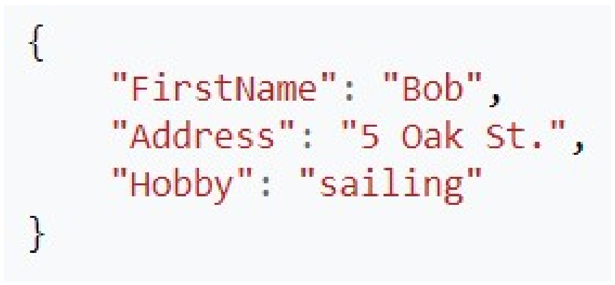
Wide-Column Stores

- Also called column stores
- Stores data using a column oriented model.
- The store contains the column families (like relational tables) which contain rows, which contain columns.
- Each column is contained to its row. It doesn't span all rows like in a relational database. Each column contains a name/value pair, along with a timestamp.

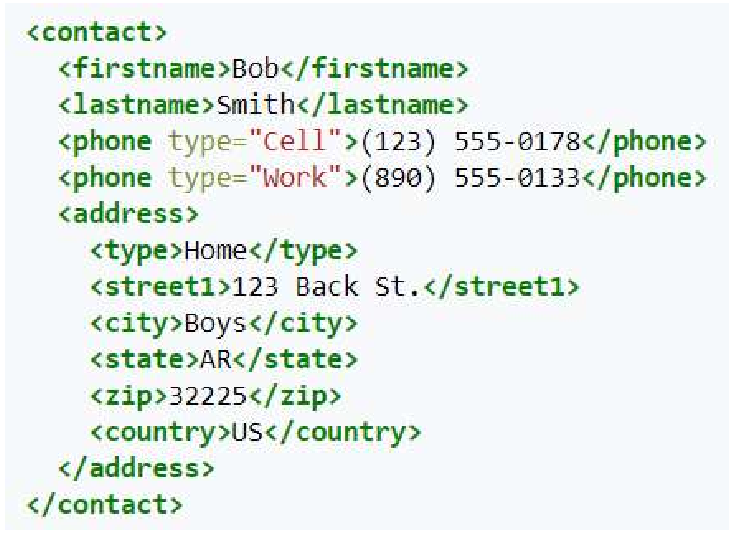


Document Stores

- Document stores contain data objects that are inherently hierarchical, tree-like structures (JSON or XML).
- Documents in a document store are not required to adhere to a standard schema.



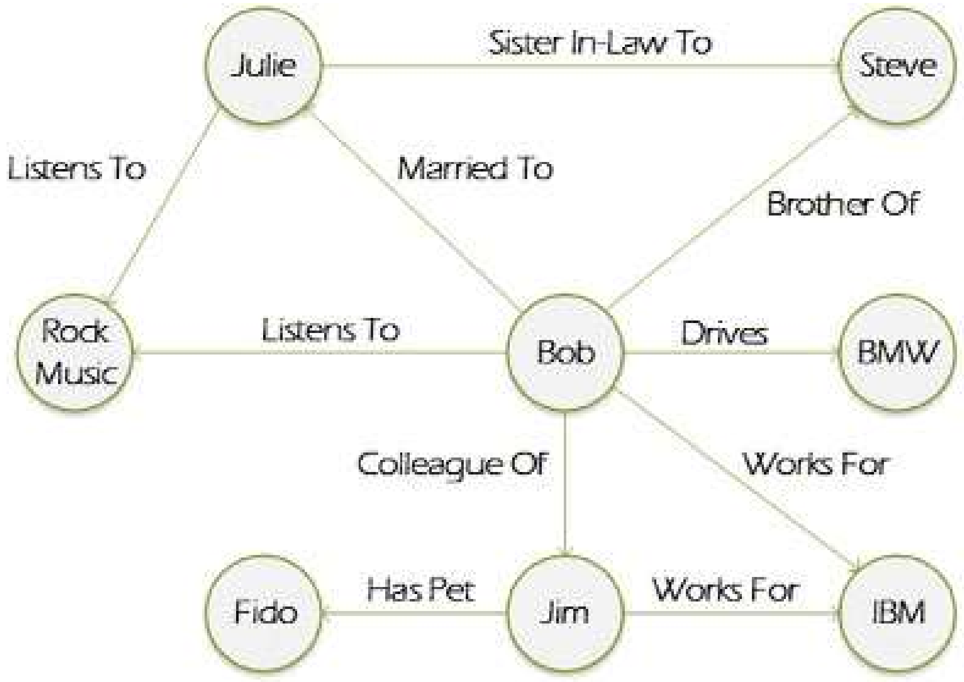
```
{
    "FirstName": "Bob",
    "Address": "5 Oak St.",
    "Hobby": "sailing"
}
```



```
<contact>
  <firstname>Bob</firstname>
  <lastname>Smith</lastname>
  <phone type="Cell">(123) 555-0178</phone>
  <phone type="Work">(890) 555-0133</phone>
  <address>
    <type>Home</type>
    <street1>123 Back St.</street1>
    <city>Boys</city>
    <state>AR</state>
    <zip>32225</zip>
    <country>US</country>
  </address>
</contact>
```

Graph Stores

- Uses graphic structures for semantic queries with nodes, edges and properties to represent and store data.
- A graph database is essentially a collection of nodes and edges. Each node represents an entity (such as a person or business) and each edge represents a connection or relationship between two nodes.
- Graph databases are well-suited for analyzing interconnections.
- There has been a lot of interest in using graph databases to mine data from social media.



Use Cases for NoSQL Databases

- Key-value stores: [Redis] For cache, queues, fit-in memory, rapidly changing data, store blob data. (Shopping cart, Session Data, Stock Prices). Fastest performance
- Wide-column stores: [Cassandra] Real-time querying of random (non-sequential) data, huge number of writes, sensors. (Web analytics, real-time data analytics, time series analytics): Internet scale
- Document stores: [MongoDB] Flexible schemas, dynamics queries, defined indexes, good performance on big DB: (Order data, customer data, log data, chat sessions, tweets, ratings, comments). Fastest development
- Graph database: [Neo4j] Graph-style data, social network, master data management (fraud detection, graph search, gene sequencing)

RDBMS vs. MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
SQL query language	Document-based query language
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

MongoDB Download and Install

- https://www.mongodb.com/download-center/community
- Download and install
- Create data folder which will be used as default database storage path for MongoDB:
  - C:\pravinp\SUNYK\Spring2019\Courses\CSE308\project\mongodb\data
- Start Mongodb using following command:
  - "C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" –dbpath "C:\pravinp\SUNYK\Spring2019\Courses\CSE308\project\mongodb\data"

Taken from https://geekflare.com/getting-started-mongodb/

Creating the First Collection

- MongoDB stores the data in the form of JSON documents.
- A group of such documentation is collectively known as a collection in MongoDB.
- A collection is analogous to a table in a relational database.
- A document is analogous to a record in a relational database.
- Execute following command from MongoDB home directory to create a database:
  - C:\Program Files\MongoDB\Server\4.0\bin> mongo tutorial
- Create a collection with following command:
  - > db.createCollection('firstCollection');

Inserting a Document in a Collection

- Insertion of the first JSON document into the **firstCollection**
  - > db.firstCollection.insertOne({name:'Abhishek',skill:'MongoDB'});
- Find above document from the collection
  - db.firstCollection.find(); //Fetches all available documents
- Insert another document in the collection
  - > db.firstCollection.insertOne({name:'GeekFlare',skill:'Java,MongoDB,NodeJS'});
- Try following commands:
  - db.firstCollection.find();
  - db.firstCollection.find({name:'Abhishek'}); //using filter
  - db.firstCollection.find({skill:/.*MongoDB.*/}); //filter using regex
  - db.firstCollection.find({skill:/.*Java.*/}); //filter using RegEx

Complex Queries in MongoDB

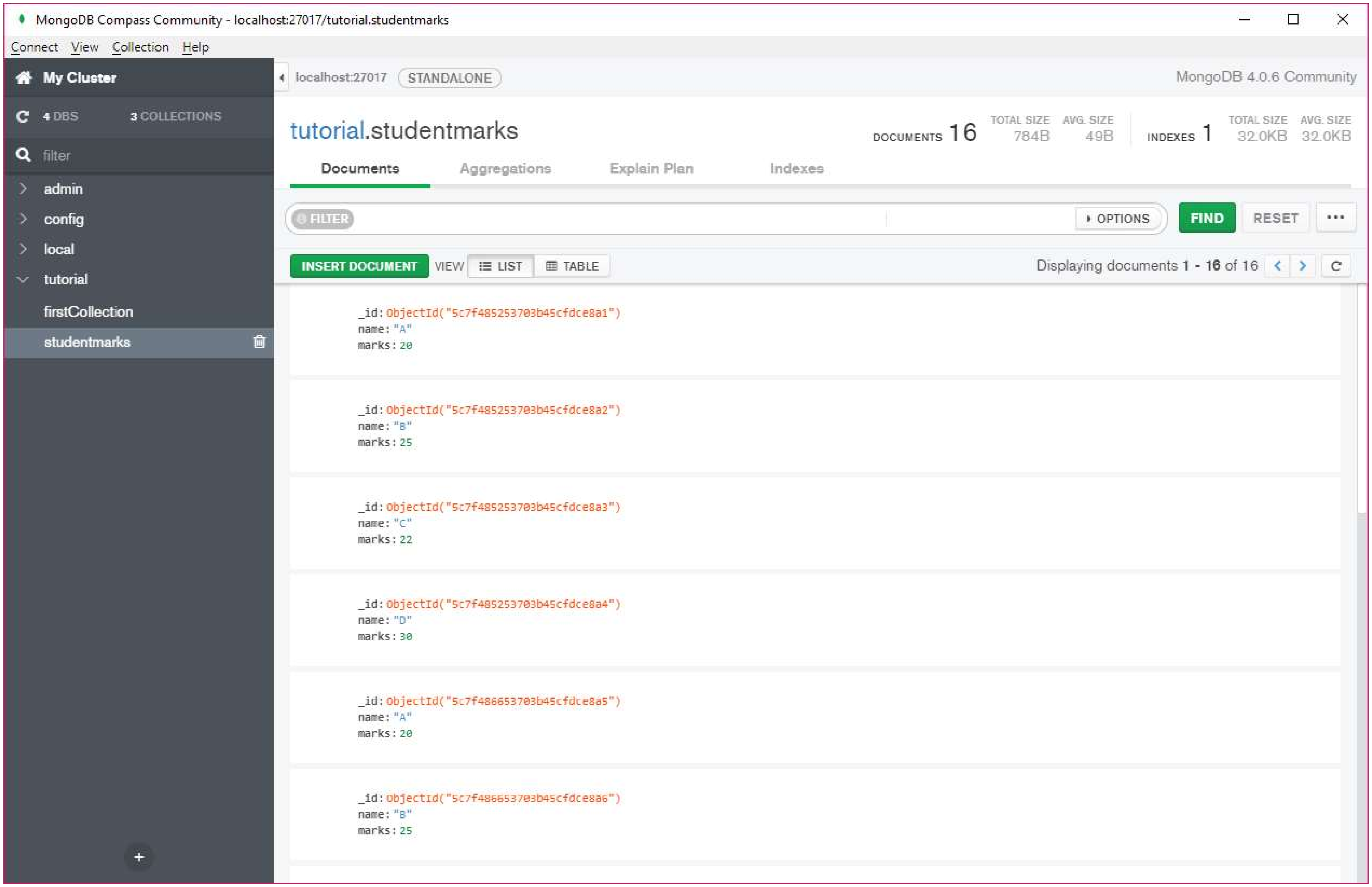
- Operators such $or, $and as well as $not could be used for query operations
- E.g. get the list of documents where name attribute contains Abhishek or skill contains Java.
  - db.firstCollection.find({$or: [{name:'Abhishek'},{skill:/.*Java.*/}]});

Complex Queries in MongoDB

- Create another collection
  - db.createCollection('studentmarks');
- Insert a document in studentmarks collection
  - db.studentmarks.insertMany([{name:'A',marks:20},{name:'B',marks:25},{name:'C',marks:22},{name:'D',marks:30}]);
- Use operators such as greater than ($gt), less than ($lt) or not equal to ($ne)
  - db.studentmarks.find({marks:{$gt:22}});

More operators

Operator	Use	Example
$eq	Check if the value is equal	{marks:{$eq:20}}
$lt	Check if value is less than	{marks: {$lt:20}}
$gte	Check if value is greater than or equal to	{marks:{$gte:22}}
$lte	Check if value is less thank or equal to	{marks:{$lte:22}}
$ne	Check if value is not equal to	{marks:{$ne:22}}
$in	Check if value is equal to either of the values from the array	{marks:{$in:[20,22]}}
$nin	Check if value is not equal to any value from the array	{marks:{$nin:[22,25]}}


MongoDB Compass Community - localhost:27017/tutorial.studentmarks
Connect View Collection Help
My Cluster
4 DBS 3 COLLECTIONS
filter
admin
config
local
tutorial
firstCollection
studentmarks
localhost:27017 STANDALONE
MongoDB 4.0.6 Community
tutorial.studentmarks
DOCUMENTS 16 TOTAL SIZE 784B AVG. SIZE 49B
INDEXES 1 TOTAL SIZE 32.0KB AVG. SIZE 32.0KB
Documents
Aggregations
Explain Plan
Indexes
FILTER
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 16 of 16

```
_id: ObjectId("5c7f485253703b45cfdce8a1")
name: "A"
marks: 20

_id: ObjectId("5c7f485253703b45cfdce8a2")
name: "B"
marks: 25

_id: ObjectId("5c7f485253703b45cfdce8a3")
name: "C"
marks: 22

_id: ObjectId("5c7f485253703b45cfdce8a4")
name: "D"
marks: 30

_id: ObjectId("5c7f486653703b45cfdce8a5")
name: "A"
marks: 20

_id: ObjectId("5c7f486653703b45cfdce8a6")
name: "B"
marks: 25
```

Individual Homework Assignment

- A user logins using Facebook
- A map pops up showing user's current location
- User is able to create/edit/delete a basic contact profile by clicking user account button
- React Native location tracking:
  - https://medium.com/quick-code/react-native-location-tracking-14ab2c9e2db8
- Facebook OAuth in React Native
  - https://alexanderpaterson.com/posts/add-social-authentication-to-a-react-native-application
- Building Spring Boot, MongoDB and React.js CRUD Web Application
  - https://www.djamware.com/post/5ab6397c80aca714d19d5b9c/building-spring-boot-mongodb-and-reactjs-crud-web-application