Auto-Garçon Style Guide

Contents

Chapter 1

General Style

1.1 File Naming

The file naming convention requires PascalCasing where every word in the file name is capitalized. The file name should not include any spaces, hyphens, underscores, or other special characters. Avoid creating generic filenames such as main.txt or abbreviations whenever possible.
GOOD:

- RestaurantSite.html
- DatabaseAPI.api

BAD:

- Two Words.txt
- Using_Underscores.java
- Super-hyphenated.js

Test files should be setup in their own separate

figure out how testing should happen

1.2 Syntax

Although Javascript is a flexbile language. It is important for the readability of the code to include uniform syntax. This includes

- Including semicolons after every line of coding
- Including curly braces whenever defining new scope. Specifically in the following way:
  function(){ ← the curly bracket is on the same line as the function header
  ...
  Code
  ...
  }
- Inline comments are to be written above the line of code that it is referring to. An example would be:
  \\Here is a comment for x = 5*x
  x = 5*x

1.3 Indentation

Indentation is required for the readability of code. Indents should be done through tabs equivalent to 4 spaces.

Include instructions on how to do the tab settings for some popular editing tools to help people out.

1.4 README

Every group is responsible to include a README.txt within their working directory. A README.md is also an acceptable format which provides some additional functionality. The format for the README.txt file is as follows:

```
Date Created:
Last Updated:
Group: [Web UI/Database/Android/Alexa]

Execution:

./script arguments
Description: This script does the following

Files:
-Examplefile.txt: Describe what the file does
-AnotherExample.txt
-Example3.txt: Another description

Folders:
+exampleDirectory
    -FileWithinDirectory
    +anotherDirectory
        -FileHere
+SecondDirectory
    -SecondFile
```

Chapter 2

Java

2.1 Documentation

The code documentation should be set up in the standard Javadocs format so that it is able to be compiled. The format is shown below:

2.1.1 Functions

Each function should have a JavaDoc comment with input parameter descriptions, return value descriptions, and a brief description of what the function does. The comment should be in the following format:

```
/**
* Description of the function
* @param input
* @return value
**/
```

2.2 Declarations

These are the guidelines for declaring functions and variables.

2.2.1 Variables

Variables should be declared in a common place. This means they should be declared at the start of a function and all together. There shouldn't be a declaration of a variable in the middle of the code. The exception being declaring an iterator in a loop.
Example:

```
main() {
    int someInteger;
    int[] someIntegerArray;
    int[] anotherInt = 0;
    \\now code is written
}
```

When naming variables, Camel Casing should be used. Variables should be named in such a way that indicates what the variable is for. Some abbreviation is acceptable but not so much that the variable cannot be recognized for their purpose. The exception for this is using "i", "j", or the likes for an iterator. Example:

Acceptable: int returnInt
Not acceptable: int ri, int ReturnInt

2.2.2 Functions

Functions will all be declared inside the main function. However, the code for the functions should take place after the main function. When naming a function, Pascal Casing is to be used. The names of functions should not be abbreviated, instead they should be stated in full. Example:

Acceptable: MaxOfArray()
Not acceptable: MOA(), maxOfArray()

2.3 Brackets

There are different types of brackets used. The following brackets are to be discussed: { } , [ ].

2.3.1 Egyptian Brackets { }

Also known as curly braces. With a loop, function, or conditional statement, the open bracket is to be declared in line with the statement. Example as follows:

```
while(true) {
```

The closing bracket is to be declared on a new line from any code with proper comments following the closing bracket identify what the bracket is closing. Example as follows:

```
    \\some code here
} \\while loop
```

For list declarations, the style should follow the block brackets section below as they act similarly to arrays.

2.3.2 Block Brackets [ ]

These brackets are used for arrays. If arrays are declared in one line, the should be declared as follows:

```
int[] arr = [1, 2, 4, 1, 9];
```

Chapter 3

JavaScript

3.1 Variables

- Each variable must have its own type declaration.
- String literals must be denoted with single quotes.
- Using multi-line string literals is advised for readability. When creating a multi-line string, it should be done using concatenation instead of escaping the new line. We will not be using the ES6 template literal to preserve backward compatibility.

3.2 Naming Conventions

- Variables and functions must use camelCase.
- CONSTANTS must be all capitalized.

3.3 Conditionals

- All conditionals should be as simple as possible.
- All equality logical comparisons must use ===/!==.
- NO TERNARY OPERATORS.

3.4 Functions

- Avoid nested conditionals.

3.5 Comments

- Always comment unclear code.

3.6 Other

- ALWAYS USE SEMICOLONS.
- MUST USE ON LINE CURLY BRACES TO DEFINE SCOPE.

Chapter 4

ReactJS/JSX Style Guide

4.1 Before creating Components

Research the available components, layout spacing, etc. in our Bootstrap library here:
https://getbootstrap.com/docs/4.4/components/alerts/

Often times there is no need to create new components, and if so we can build a component consisting of other components. Look at all the available options the component offers as well.

Finally, use their offered spacing classNames instead of creating style variables if possible. They are shown here: **https://getbootstrap.com/docs/4.0/utilities/spacing/**

Example:

```
// Good - this gives the Column component a padding of 3
<Col className="p-3">
    <div>
        {this.renderHours()}
    </div>
</Col>
```

.

CSS box model is very important for UI development. A refresher is here:
https://www.geeksforgeeks.org/css-box-model/

4.2 Components

Creating components should follow this class declaration format.

```
// bad
const Listing = React.createClass({
  // ...
  render() {
    return <div>{this.state.hello}</div>;
  }
});

// good
class Listing extends React.Component {
  // ...
  render() {
    return <div>{this.state.hello}</div>;
  }
```

```
}
.
```

4.3 Naming

Variables should always have meaning and never be single characters or vague. Use .jsx for components and PascalCase for the component and filename

```
// bad
import reservationCard from './ReservationCard';

// good
import ReservationCard from './ReservationCard';
```

4.4 Comments and Documentation

Components should be documented at the top of the file with their purpose and how they function.

Before a function that is not within the render of a component, the purpose of the function should be commented. Within the function any logic that is not straightforward should have a brief description.

```
//Good
/*
    This Prop is used to render the cards of the Manager Menu page.
    The menu is a 2d array with the first array containing only
    the title of the food item. The second array contains the category,
    price, calories, picture and whether the item is in stock.

    getStockState is a helper function which takes in_stock property
    which is either 0 or 1 and creates the appropriate text to display.

*/
function MenuProp(props) {
    ...
}



// Good - although uncomplicated logic just for example purposes

/*
*   Used to loop through a number
*/
function myFunction(number) {
    // Loops through every number up to the parameter and prints it
    for (var j = 0; j < number; j++) {
        console.log(j);
    }
}
```

4.5 Alignment

Use a single indent for each block of scope. First curly brace should be on first line and second should be at the same indentation as the first line. Indentation should be **2 spaces**

```
// if props fit in one line then keep it on the same line
<Foo bar="bar" />

// Good
for (let i = 0; i < 3; i++) {
  let children = []
  //Inner loop to create children
  for (let j = 0; j < 5; j++) {
    children.push(<td>{`Column ${j + 1}`}</td>)
  }
  //Create the parent and add the children
  table.push(<tr>{children}</tr>)
}
```

4.6 Styling

Styling for each variable should be within the component and be outside of the component declaration. The variable names should be declared as constants and be meaningful. The Javascript object should have the key and values as strings.

```
//Bad
var itemStyle = {
    display: "flex",
    borderBottom: "grey solid 1px",
};

//Good
const cardHeaderStyle = {
  'background-color': '#0b658a',
  'color': '#ffffff'
}
```

4.7 Props

Always use camelCase for prop names

```
// bad
<Foo UserName="hello" phone_number={12345678} />

// good
<Foo userName="hello" phoneNumber={12345678} />
```

4.8 Parenthesis

Wrap JSX tags in parentheses when they span more than one line

```
// bad
render() {
```

```
  return <MyComponent variant="long body" foo="bar">
           <MyChild />
         </MyComponent>;
}

// good
render() {
  return (
    <MyComponent variant="long body" foo="bar">
      <MyChild />
    </MyComponent>
  );
}

// good, when single line
render() {
  const body = <div>hello</div>;
  return <MyComponent>{body}</MyComponent>;
}
```

4.9 Full example file

The export of the component should be at the bottom of the file.

```
import React from "react";
import Order from "./Order";
import Container from 'react-bootstrap/Container';

class Orders extends React.Component{
    /*
    This Prop is used to render the orders for the Cook page.
    The orders are an array of object containing order details.  Look at the Order compone

    renderOrders is a helper function which takes all the orders,
    converts them to Order objects and returns a single JSX object that
    will be exported as an Orders
    component.
*/
  constructor(props) {
    super(props);
    this.state = {
      orders: [
        {table: 1, items: [{quantity: 1, title: "Hamburger"}]},
        {table: 2, items: [{quantity: 1, title: "Hamburger"}]}
      ]
    };
  }

  renderOrders() {
    // Returns every order stored in the components state as an individual
    Order component
    return this.state.orders.map((item, key) =>
      <Order key={key} id={key} order={item}/>
    );
```

```
  }

  render() {
    return (
      <Container fluid>
        <div style={ordersStyle}>
          {this.renderOrders()}
        </div>
      </Container>
    )
  };
}

const ordersStyle = {
  'display': 'flex',
  'margin': '30px',
};

export default Orders;
```

Chapter 5

SQL

5.1 Naming Conventions

In general, every name should be unique and should not match with any of the reserved key words in SQL. Names should be limited to 30 characters. Names must begin with a letter and can only contain letters and underscores. Names should only have a single underscore. Names should avoid using abbreviations and should be commonly understood.

5.1.1 Tables

When creating tables for our database, use a collective name such as staff or people, or the plural form of a noun.

5.1.2 Attributes

The attributes of each table should be a singular noun that does not have the same name as a table. There are a collection of suffixes that have universal meaning that should be used if they are applicable.

- _id—a unique identifier such as a column that is a primary key.
- _status—flag value or some other status of any type such as publication_status.
- _total—the total or sum of a collection of values.
- _num—denotes the field contains any kind of number.
- _name—signifies a name such as first_name.
- _seq—contains a contiguous sequence of values.
- _date—denotes a column that contains the date of something.
- _tally—a count.
- _size—the size of something such as a file size or clothing.
- _addr—an address for the record could be physical or intangible such as ip_addr.

5.1.3 Stored procedures

Stored procedures must contain a verb.

5.2 Querying the Database

Querying the database must use uppercase for all reserved keywords. The query should be aligned to the right of a key word. Every keyword should appear on a separate line except for the AS keyword.

5.2.1 Space

Spaces should be included between keywords, before and after the equals sign =, after commas and surround apostrophes where not within parentheses.

5.2.2 Line spacing

Newlines should be applied before an AND or OR, after a semicolon to separate queries, after each keyword definition, after a comma when separating multiple columns into logical groups.

5.3 Creating a table

The data types that could be used should be.

5.4 Documentation

5.4.1 Queries

Each query should have a comment briefly describing the query in the C style comment block /**/

5.4.2 Tables

Tables should have the following documentation

5.5 Reference

This style guide is modelled after the style guide provided by Simon Holywell and can be referenced by going to https://www.sqlstyle.guide

Chapter 6

XML

6.1 Documentation

The code documentation should be set up in the standard XML format so that it is able to be compiled. The format is shown below:

6.1.1 Functions

Each XML element should provide a brief comment with what the element is for, The comment should be in the following format:

```
<!--
*
* Description of the Element
*
 -->
```

6.1.2 Header

Each xml file should include the following filled in with the appropriate information.

```
<!--
*
* Description of the File
*
 -->
```

6.2 Declarations

These are the guidelines for declaring functions and variables.

6.2.1 Strings

Strings should be declared in a common place and NOT hard coded. This means they should be declared in the strings.xml file. There shouldn't be a declaration of a variable in the middle of the code.

When naming strings, camelCasing should be used. Strings should be named in such a way that indicates what the String is for. Some abbreviation is acceptable but not so much that the variable cannot be recognized for their purpose.