CBT Documentation

About CBT Project: CBT is a computer-based recruitment system for BJIT. BJIT recruits a huge number of employees every year, mostly engineers. It's quite tedious and resource intensive for HR to evaluate the candidates especially if they require conducting examinations. Smart CBT is designed to automate these processes and produce faster results.

The key functionalities include generating custom question sets and automated evaluation wherever possible. The system will be kept up to date by peer review. As a huge number of users will be engineering, the system will include functionality for compiling exam codes and similar essential tools.

Meet the team member:

SL	Name	Role	Start date	End date
01	Zulkar Nayin	FE		
02	Ashikujjaman Shaikat	FE		
03	Sumiya Alam Akhi	FE		
04	Toufikur Rahman Toufik	FE		
05	Shehzan Haider Chowdhury	FE		
06	Saadman Sayeed	FE		
07	Humayra Eva	FE		
08	Israr Munim Chowdhury	FE		
09	Rafia Zahan Tamanna	FE		
10	Abid Hasan	FE		

Table of contents:

Tools we used For this project: Figma, Redmine, VS code, Git Gerrit

1. Installation and Usage

Use the package manager yarn to install the project

1.1. **Install dependencies:** yarn install

1.2. **Run project:** yarn run dev

1.3. **Build project:** yarn run build

1.4. **Eslint run:** yarn run lint

1.5. **Prettier run:** yarn run prettier

1.6. **Storybook run:** yarn run storybook

2. The technology used in this project:

2.1 **Webpack** (using custom web pack) link

2.2 **Storybook** Storybook is a tool for UI development. It makes development faster and easier by isolating components. link

2.3 **Hygen** is the simple, fast, and scalable code generator that lives *in* your project. link

2.4 **Husky** is a git pre-commit hook link

2.5 **Eslint** ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code. link

2.6 **Prettier** is an opinionated code formatter. It enforces a consistent style by parsing your code and reprinting it with its own rules that take the maximum line length into account, wrapping code when necessary. link

2.7 **SCSS** support

2.8 **Stylelint** link

2.9 **Commitlint** link

2.10 **Docker** is an open platform for developing, shipping, and running applications link

**Note: most of the above-mentioned technologies are used for development only*

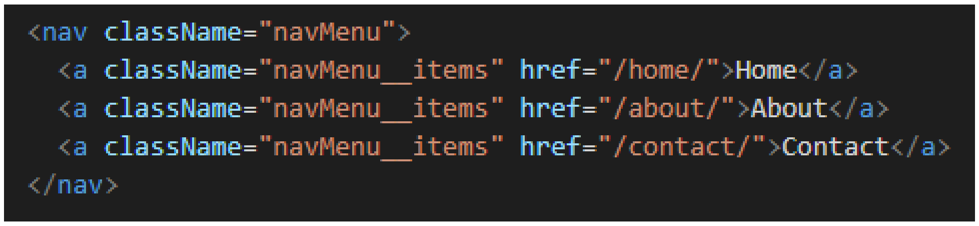
3. The conventions we need to follow

3.1 HTML:

- Naming convention: **BEM** link
- Semantic HTML
- Ensure web accessibility.

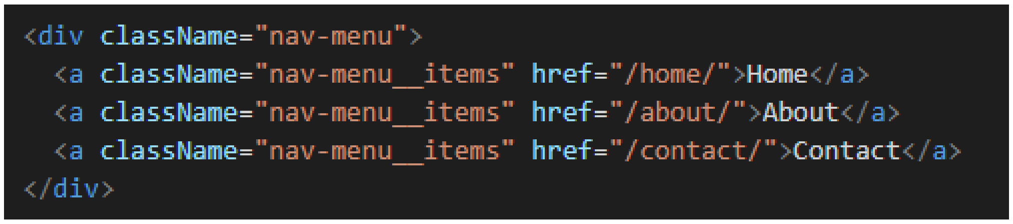
Example: *Use camel case instead of kabab case*

DO:



```
<nav className="navMenu">
  <a className="navMenu__items" href="/home/">Home</a>
  <a className="navMenu__items" href="/about/">About</a>
  <a className="navMenu__items" href="/contact/">Contact</a>
</nav>
```

DON'T:



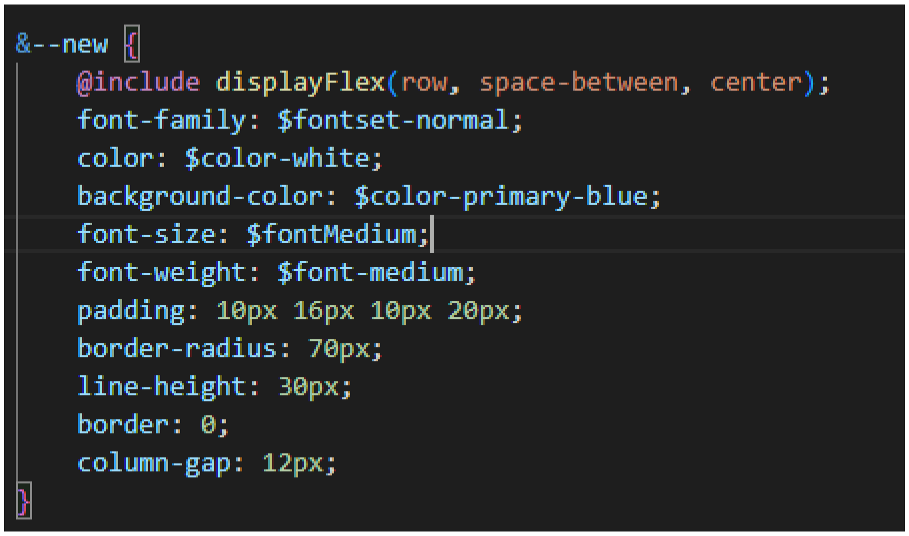
```
<div className="nav-menu">
  <a className="nav-menu__items" href="/home/">Home</a>
  <a className="nav-menu__items" href="/about/">About</a>
  <a className="nav-menu__items" href="/contact/">Contact</a>
</div>
```

3.2 CSS:

- Use SCSS instead of pure CSS
- Avoid embedding CSS in <style> tags or using inline CSS
- All color, font weight, font family and relevant information must be used as variables.
- Add required vendor prefixes (Example: -webkit- , -moz-)
- Expected design with minimal CSS.
- Pixel perfect at any cost.

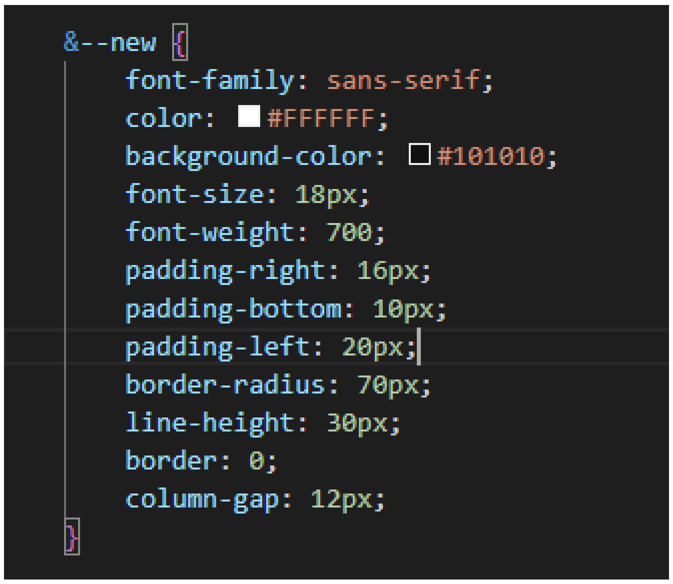
***Example**: Using variables, mixin and avoiding unnecessary CSS   properties.*

DO:



```scss
&--new {
    @include displayFlex(row, space-between, center);
    font-family: $fontset-normal;
    color: $color-white;
    background-color: $color-primary-blue;
    font-size: $fontMedium;
    font-weight: $font-medium;
    padding: 10px 16px 10px 20px;
    border-radius: 70px;
    line-height: 30px;
    border: 0;
    column-gap: 12px;
}
```

DON'T:

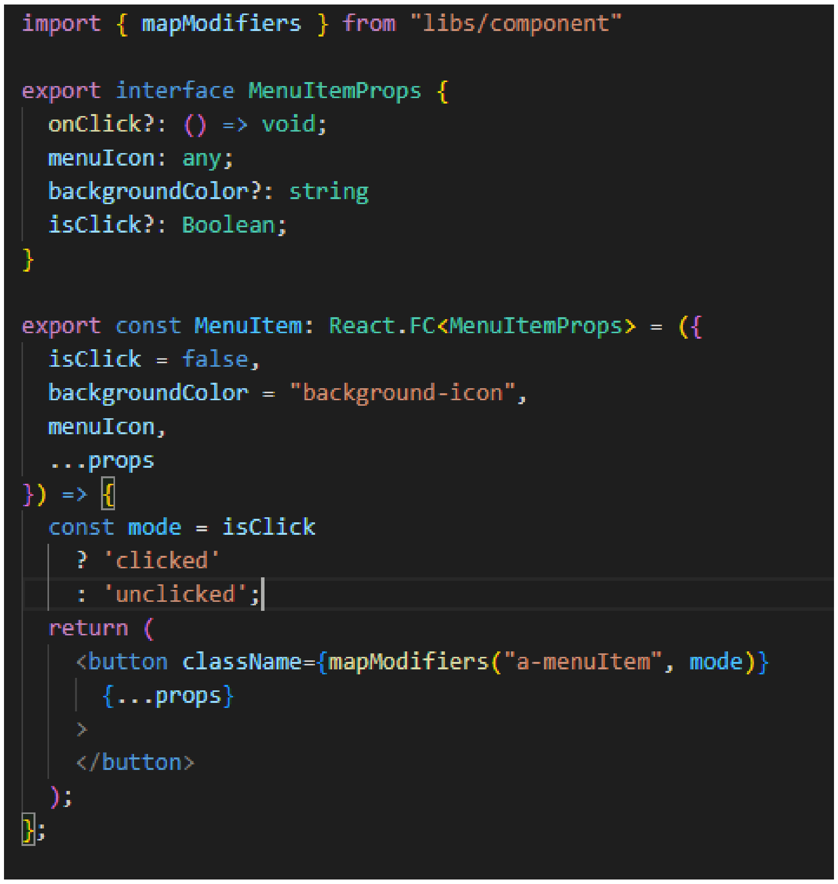


```scss
&--new {
    font-family: sans-serif;
    color: ⬜#FFFFFF;
    background-color: ⬛#101010;
    font-size: 18px;
    font-weight: 700;
    padding-right: 16px;
    padding-bottom: 10px;
    padding-left: 20px;
    border-radius: 70px;
    line-height: 30px;
    border: 0;
    column-gap: 12px;
}
```

3.3. **JavaScript**:

- Clean and simple code as much as possible.
- Ensure no runtime error and console warning.
- Class name: Follow the pascal case convention **(E.g: NavItem)**
- Function name: Follow camel case convention (**E.g: mapDataAttrs**)
- Variable name: Follow camel case convention (**E.g: loadedData**)
- Boolean name: boolean variables should have the prefix 'is'. (**E.g: isLoaded**)
- Ensure consistent use of arrow function.

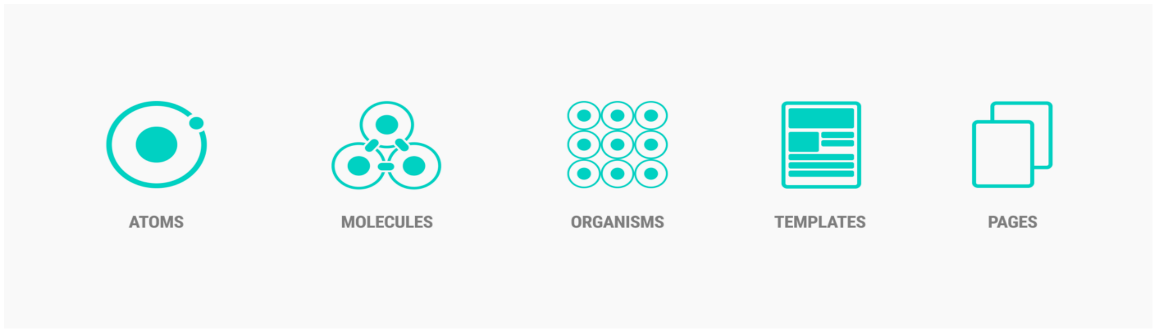
Example:



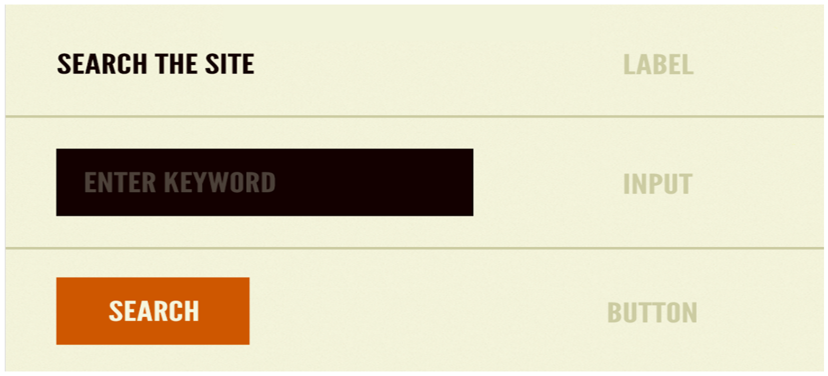
```
import { mapModifiers } from "libs/component"

export interface MenuItemProps {
  onClick?: () => void;
  menuIcon: any;
  backgroundColor?: string
  isClick?: Boolean;
}

export const MenuItem: React.FC<MenuItemProps> = ({
  isClick = false,
  backgroundColor = "background-icon",
  menuIcon,
  ...props
}) => {
  const mode = isClick
    ? 'clicked'
    : 'unclicked';
  return (
    <button className={mapModifiers("a-menuItem", mode)}
      {...props}
    >
    </button>
  );
};
```

4. <u>Atomic Design Details:</u>

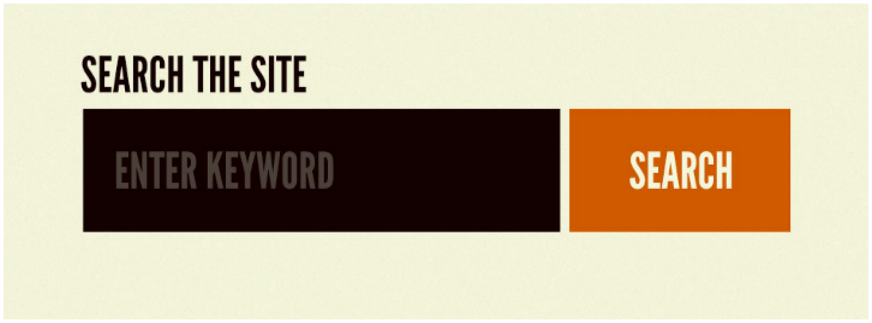
Atomic design provides a clear methodology for crafting design systems. There are five distinct levels in atomic design: link



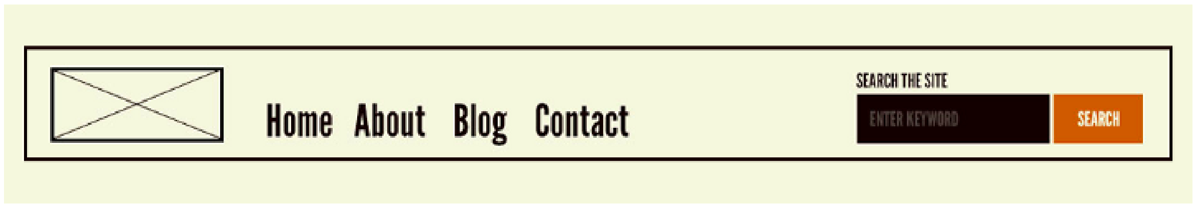
4.1 ***Atoms:*** represent the smallest entity in UI elements and they can't be broken down any further. Some examples are colors, fonts, animations and single images.



4.2 ***Molecules:*** are groups of atoms bonded together that take on distinct new properties. They form relatively simple UI elements functioning together as a unit. Some examples are a form label and search input.

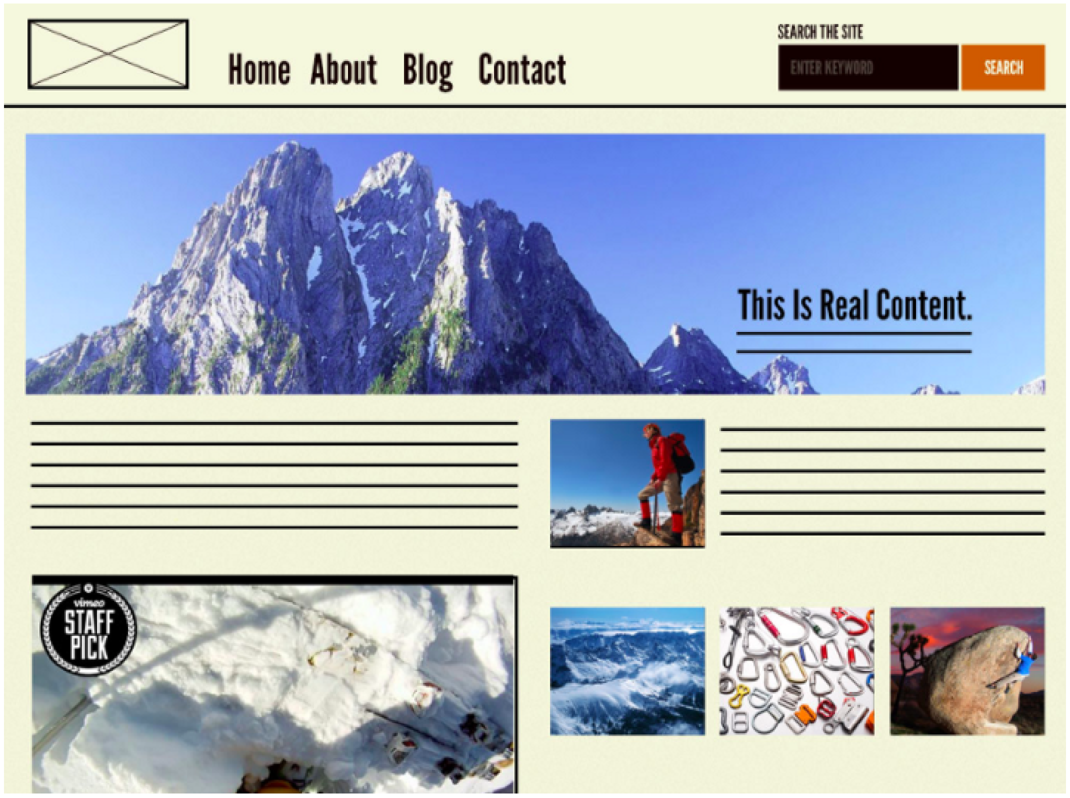


4.3 ***Organisms*** are relatively complex UI components composed of groups of molecules and/or atoms. Most organisms can function on their own, without relying on other elements on the page. Some examples are navigations, sidebars, forms, card design and popups



4.4 ***Templates*** are pages without real content and articulate the design's underlying content structure. Essentially, they combine organisms into a proper website layout.

4.5 ***Pages:*** are specific instances of templates that demonstrate the final UI looks like and with real representative content in place.

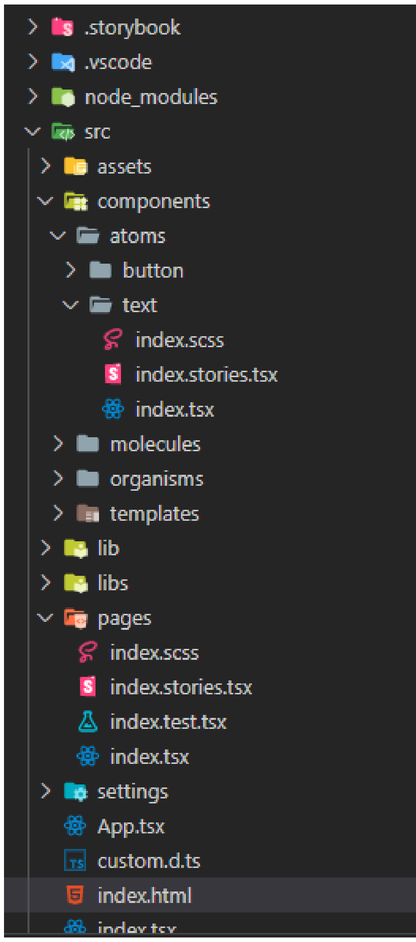


5. Folder Structure

5.1 **Atomic architecture: Atomic** is a methodology composed of five distinct stages working together to create interface design systems in a more deliberate and hierarchical manner. Link

5.2 **Atomic folder structure:** Inside atoms, molecules, organism or page folder include 4 files like as:
Example:

- ➔ **atoms**
  - ◆ Index.tsx
  - ◆ Index.scss
  - ◆ Index.storybook.tsx



```
> .storybook
> .vscode
> node_modules
v src
  > assets
  v components
    v atoms
      > button
      v text
          index.scss
          index.stories.tsx
          index.tsx
    > molecules
    > organisms
    > templates
  > lib
  > libs
  v pages
      index.scss
      index.stories.tsx
      index.test.tsx
      index.tsx
  > settings
    App.tsx
    custom.d.ts
    index.html
    index.tsx
```

6. <u>Contribution Guide(PR Process):</u>

6.1. Create topics branch:

- Atom: atom/button
- Molecule: molecule/profileCard
- Organism: organism/comment
- Template: template/questionList
- Page: page/admin

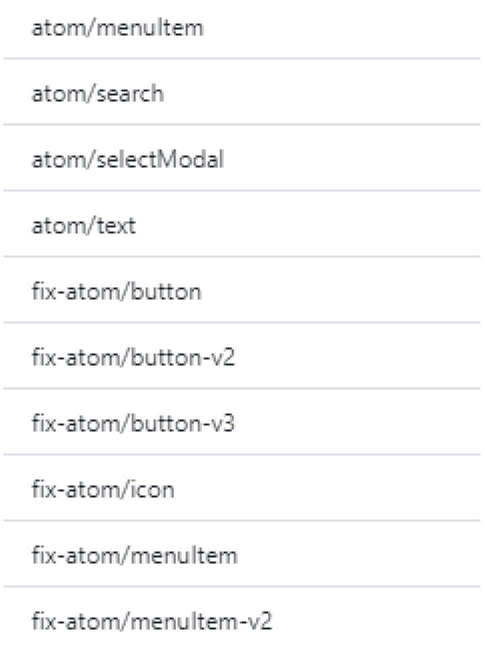
6.2 Bug fix branch Name:

Use proper versioning numbers for naming branches. A simple guide of adding version numbers are as follows. Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible changes
2. MINOR version when you add functionality in a backward compatible manner
3. PATCH version when you make backward compatible bug fixes

- Single time fix: fix-atom/button-v1.0.0
- Multiple time fix: fix-atom/button-v1.0.1., fix-atom/button-v1.1.0

Example:



atom/menuItem

atom/search

atom/selectModal

atom/text

fix-atom/button

fix-atom/button-v2

fix-atom/button-v3

fix-atom/icon

fix-atom/menuItem

fix-atom/menuItem-v2

6.3 Before commit: One of our topmost priority is ensuring the quality of code. Use Husky to help ensure that no mistakes are committed. Check the code according to the guideline then make sure you have run stylelint , ESLint and prettier properly. any redundancy caught by husky will stop you from committing.
Also make sure your logic is working properly.
Enabling “format on save” in the IDE(VS code) will make sure your code is properly formatted.

<u>6.4: Commit Guide:</u>

PR Message: The commit message for a PR should be structured as follows:

<type>: <Subject>

[optional body]

[optional footer]

List of Commit types:

- **docs**: Documentation only changes
- **feat**: A new feature

- **fix**: A bug fix
- **perf**: A code change that improves performance
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **style**: Changes that do not affect the meaning of the code (white-space, formatting, missing semicolons, etc)
- **test**: Adding missing tests or correcting existing tests

An ***Example*** of the commit message is given in the following:

```
fix: prevent racing of requests

Introduce a request id and a reference to the latest request.
Dismiss
incoming responses other than from the latest request.

Remove timeouts which were used to mitigate the racing issue but are
obsolete now.

Reviewed-by: Shehzan Chowdhury
Refs: #123
```

7.0 Project Management:

We are using redmine for this project management purpose. Redmine is a free and open-source, web-based project management and issue tracking tool. It allows users to manage multiple projects and associated subprojects. It features per-project wikis and forums, time tracking, and flexible, role-based access control.

8.0 Reference:

Acronyme	**Full form**
CBT	Computer Base Test
FE	Front end
BE	Backend
PR	pull request
KT	Knowledge transfer
BEM	Block , element, modifier
ABEM	Atomic, Block, element , modifier