Side Effects Script

Working of the script

Run command

```
node script-sideEffects-analyzer.mjs <ENTRY DIR>
```

The script takes in entry directory of from where it builds the dependency graph of each entry module. All the modules in this graph are parsed by the script independently.

Workflow

The script uses `babel parser` to parse the AST of the modules. The following is the general workflow after running the script:

1.) `getFiles` function is used to get the entry directory modules. These modules are used to build the dependency graph

2.) `getImportedModules` function is used to build the dependency graph starting from a source module. This function collects the imports/re-exports and recursively calls itself. It makes use of a BFS traversal over the static import export declarations

3.) After the dependency graph is constructed and all target modules collected, `hasSideEffects` method is called to mark the module as side effect free or not. It parses the AST and analyse all the nodes case by case in a recursive manner.

4.) The final list of side-effects modules is written to the "sideEffects" property of package.json.

5.) After the script, the build process of NextJS is called.

Achievables

- The script can handle many cases of side effects and provides a tool to greatly reduce the number of modules to analyse for side effects
- Script works in many primitive cases and can always be coupled with other solutions without breaking the code.
- It can also be used for tree shaking barrel-files and hence provides an alternative solution to the problem of eliminating barrel exports.

Drawbacks

1.) Requires Developers to mark `/* #__ PURE__* /` before Call expressions to tell that the call is side Effect free

Eg.

```
var AvatarGroup = React.forwardRef(function (_a, ref) {
    ...
});
var StyledComponent = styled('div', 'flex');
AvatarGroup.displayName = 'AvatarGroup';
export { AvatarGroup };

/*
    The script will consider This module as impure since there i
    Call expression React.forwardRef(function () {...}) which o
    can tell that it is side Effect free
*/
```

2.) Currently only considers assignments to `window` and `document` objects as side effects. But there may be assignments to Other global properties like Array.prototype which the existing script fails to parse.

3.) The **dynamic imports are considered as side effect free calls by default** which works in most cases but their may be cases when we would like to import a side effectful module like CSS modules.

4.) Even setting sideEffects may not work in cases like library imports, importing client component module in server components in app routers which seem to include the module anyways.

5.) The script has been tested on packages like space-web and doesnt take much time to run. However it may take considerable time if our project is very large and the side effects are deeply nested in the modules.

6.) The ROI of running the script is very low given the unused modules represent a very minor issue causing bundle bloating in the main-repo. The major issue of **tree shaking failing in case of shared modules used in different pages** is not solved by this script.