Proposed Coding Standards for WebApp - ShebaBondhu

Ruponti Muin Nova (2254902038)
Jawad Anzum Fahim (2254902045)

September 27, 2025

1 Airbnb Style Guide

Imports

Airbnb standard:

- Imports always at the top.
- Grouping order:
  1. Node.js built-ins
  2. External libraries
  3. Internal modules (`@/lib`, `@/components`)
- Use absolute imports (Next.js supports with `tsconfig.json` paths).
- Sort alphabetically within group.
- No unused imports.

```
//     Good
import fs from 'fs';
import express from 'express';

import { dbConnect } from '@/lib/dbConnect';
import Navbar from '@/components/Navbar';
import type { User } from '@/types/user';

//     Bad
import Navbar from '../components/Navbar';
import fs from 'fs'; import express from 'express';
```

Functions

- Function names → camelCase
- Components → PascalCase
- Prefer arrow functions for anonymous functions
- Keep functions pure (avoid side-effects)
- Async functions → always use try/catch

```
// Utility function
const formatDate = (date: Date): string => {
  return date.toISOString().split('T')[0];
};

// Async function
const fetchUser = async (id: string): Promise<User | null> =>
    {
  try {
    const res = await fetch(`/api/users/${id}`);
    if (!res.ok) throw new Error('Failed to fetch user');
    return res.json();
  } catch (err) {
    console.error(err);
    return null;
  }
};
```

Variables

- Use `const` by default
- Use `let` only when reassignment needed
- Never use `var`
- Naming:
  - camelCase → variables & functions
  - UPPER_CASE → constants
  - PascalCase → components, classes, types

```
//     Good
const userName = 'Ruponti';
let counter = 0;
const API_URL = process.env.API_URL;

//     Bad
var username = 'ruponti';
const Counter = 0;
```

Page Component (`page.tsx`)

- Components → PascalCase
- Return statement → JSX at bottom (logic first, then JSX)
- Self-closing tags when no children
- Use semantic HTML
- Keep JSX clean, avoid inline styles

```
// app/page.tsx
import Navbar from '@/components/Navbar';

export default function HomePage() {
  const message = 'Welcome to Next.js with Airbnb style!';

  return (
    <main>
      <Navbar />
      <h1>{message}</h1>
    </main>
  );
}
```

API Routes (`route.ts`)

- Use async functions
- Import order same as before
- Always return structured response (`Response.json`)
- Handle errors properly

```
// app/api/users/route.ts
import { dbConnect } from '@/lib/dbConnect';
import User from '@/models/User';

export async function GET() {
  try {
    await dbConnect();
    const users = await User.find();
    return Response.json(users, { status: 200 });
  } catch (error) {
    console.error(error);
    return Response.json({ error: 'Failed to fetch users' },
       { status: 500 });
  }
}

export async function POST(req: Request) {
  try {
    await dbConnect();
    const body = await req.json();
    const newUser = new User(body);
    await newUser.save();
    return Response.json(newUser, { status: 201 });
  } catch (error) {
    console.error(error);
    return Response.json({ error: 'Failed to create user' },
       { status: 500 });
  }
}
```

Database Connection (`lib/dbConnect.ts`)

- Common db connection → utility function in `lib/`
- Export default or named function
- Use singleton pattern to avoid multiple connections

```
// lib/dbConnect.ts
import mongoose from 'mongoose';

const MONGO_URI = process.env.MONGO_URI as string;

if (!MONGO_URI) {
  throw new Error('Please define MONGO_URI in .env');
}
```

```
let cached = global.mongoose as {
  conn: typeof mongoose | null;
  promise: Promise<typeof mongoose> | null
};

if (!cached) {
  cached = global.mongoose = { conn: null, promise: null };
}

export async function dbConnect() {
  if (cached.conn) return cached.conn;

  if (!cached.promise) {
    cached.promise = mongoose.connect(MONGO_URI).then((m) =>
       m);
  }
  cached.conn = await cached.promise;
  return cached.conn;
}
```

Schema (`models/User.ts`)

- One model per file
- PascalCase model name
- Define schema first, then export

```
// models/User.ts
import mongoose, { Schema, Document } from 'mongoose';

export interface IUser extends Document {
  name: string;
  email: string;
}

const UserSchema: Schema = new Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
  },
  { timestamps: true }
);

export default mongoose.models.User ||
  mongoose.model<IUser>('User', UserSchema);
```

Components (`components/`)

- PascalCase file + function name
- Keep component pure functional
- Props → always typed with `interface`
- Destructure props in function parameter

```
// components/Button.tsx
import React from 'react';

interface ButtonProps {
  label: string;
  onClick: () => void;
}

export default function Button({ label, onClick }:
   ButtonProps) {
  return (
    <button
      type="button"
      onClick={onClick}
      className="rounded bg-blue-500 px-4 py-2 text-white"
    >
      {label}
    </button>
  );
}
```

2 Google Style Guide

Imports

Google Style Rules:

- Always at the top of the file.
- Use ES6 `import`, never require().
- Group order: Node built-ins, external libs, internal modules.
- One import per line, sorted alphabetically within groups.
- No unused imports.

```
//     Good
import fs from 'fs';
import express from 'express';

import { dbConnect } from '@/lib/dbConnect';
import Navbar from '@/components/Navbar';
import type { User } from '@/types/user';

//     Bad
import Navbar, { dbConnect } from '../components/Navbar';
   import fs from 'fs';
```

Functions

Google Style Rules:

- Function names: `camelCase`.
- Components: `PascalCase`.
- Use arrow functions for inline/anonymous functions.
- Prefer pure functions, avoid side-effects.
- Async functions must use `try/catch`.

```
// Utility function
function formatDate(date: Date): string {
  return date.toISOString().split('T')[0];
}

// Async function
async function fetchUser(id: string): Promise<User|null> {
  try {
    const res = await fetch(`/api/users/${id}`);
    if (!res.ok) throw new Error('Failed to fetch user');
    return res.json();
  } catch (err) {
    console.error(err);
    return null;
  }
}
```

Variables

Google Style Rules:

- Use `const` by default.
- Use `let` only when reassignment is required.
- Never use `var`.
- Naming:
  - `camelCase` for variables and functions.
  - `UPPER_CASE` for constants.
  - `PascalCase` for components, classes, types.

```
//     Good
const userName = 'Ruponti';
let counter = 0;
const API_URL = process.env.API_URL;

//     Bad
var username = 'ruponti';
const Counter = 0;
```

Next.js Pages (page.tsx)

Google Style Rules:

- Component functions → PascalCase.
- Logic at the top, JSX return at the bottom.
- Use semantic HTML.
- Avoid inline styles, prefer classNames.

```
// app/page.tsx
import Navbar from '@/components/Navbar';

export default function HomePage() {
  const message = 'Welcome to Next.js with Google style!';

  return (
    <main>
      <Navbar />
```

```
      <h1>{message}</h1>
    </main>
  );
}
```

API Routes (route.ts)

Google Style Rules:

- Use async functions.
- Import order as before.
- Always return structured JSON.
- Handle errors explicitly.

```
// app/api/users/route.ts
import { dbConnect } from '@/lib/dbConnect';
import User from '@/models/User';

export async function GET() {
  try {
    await dbConnect();
    const users = await User.find();
    return Response.json(users, { status: 200 });
  } catch (error) {
    console.error(error);
    return Response.json({ error: 'Failed to fetch users' },
       { status: 500 });
  }
}
```

Database Connection (lib/dbConnect.ts)

Google Style Rules:

- Common code → utility functions in `lib/`.
- Export named function.
- Use singleton pattern to prevent multiple connections.

```
// lib/dbConnect.ts
import mongoose from 'mongoose';

const MONGO_URI = process.env.MONGO_URI as string;

if (!MONGO_URI) {
  throw new Error('Please define MONGO_URI in .env');
}

let cached = global.mongoose as {
  conn: typeof mongoose | null;
  promise: Promise<typeof mongoose> | null
};

if (!cached) {
  cached = global.mongoose = { conn: null, promise: null };
}

export async function dbConnect() {
  if (cached.conn) return cached.conn;

  if (!cached.promise) {
    cached.promise = mongoose.connect(MONGO_URI).then((m) =>
       m);
  }
  cached.conn = await cached.promise;
  return cached.conn;
}
```

Schema (models/User.ts)

Google Style Rules:

- One model per file.
- PascalCase for model names.
- Define schema first, then export.

```
// models/User.ts
import mongoose, { Schema, Document } from 'mongoose';

export interface IUser extends Document {
  name: string;
  email: string;
}
```

```
const UserSchema: Schema = new Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
  },
  { timestamps: true }
);

export default mongoose.models.User ||
  mongoose.model<IUser>('User', UserSchema);
```

Components (components/)

Google Style Rules:

- File name and component name → PascalCase.
- Pure functional components.
- Props must be typed.
- Destructure props in parameters.

```
// components/Button.tsx
import React from 'react';

interface ButtonProps {
  label: string;
  onClick: () => void;
}

export default function Button({ label, onClick }:
   ButtonProps) {
  return (
    <button
      type="button"
      onClick={onClick}
      className="rounded bg-blue-500 px-4 py-2 text-white"
    >
      {label}
    </button>
  );
}
```

3 StandardJS Style Guide

Key StandardJS Rules

- No semicolons (`;` avoided unless necessary).
- 2-space indentation.
- Single quotes for strings.
- No unused variables or imports.
- One variable per declaration.
- Always use `const` or `let`, never `var`.

Imports

StandardJS Rules:

- Imports always at the top.
- Grouped logically: built-ins, external, internal.
- No semicolons, single quotes.

```
//     Good
import fs from 'fs'
import express from 'express'

import { dbConnect } from '@/lib/dbConnect'
import Navbar from '@/components/Navbar'
import type { User } from '@/types/user'

//     Bad
import fs from "fs"; import express from 'express';
import Navbar from '../components/Navbar';
```

Functions

StandardJS Rules:

- Function names: `camelCase`.
- Components: `PascalCase`.

- No semicolons at end.
- Async functions with `try/catch`.

```
// Utility function
function formatDate (date: Date): string {
  return date.toISOString().split('T')[0]
}

// Async function
async function fetchUser (id: string): Promise<User|null> {
  try {
    const res = await fetch(`/api/users/${id}`)
    if (!res.ok) throw new Error('Failed to fetch user')
    return res.json()
  } catch (err) {
    console.error(err)
    return null
  }
}
```

Variables

StandardJS Rules:

- Use `const` by default.
- Use `let` if reassignment required.
- One declaration per variable.

```
//     Good
const userName = 'Ruponti'
let counter = 0
const API_URL = process.env.API_URL

//     Bad
var username = 'ruponti';
const x = 1, y = 2;
```

Next.js Pages (page.tsx)

StandardJS Rules:

- Components → PascalCase.

- Logic above, JSX return at bottom.
- Use semantic HTML, no inline styles.

```
// app/page.tsx
import Navbar from '@/components/Navbar'

export default function HomePage () {
  const message = 'Welcome to Next.js with StandardJS!'

  return (
    <main>
      <Navbar />
      <h1>{message}</h1>
    </main>
  )
}
```

API Routes (route.ts)

StandardJS Rules:

- Async functions with try/catch.
- JSON responses only.
- No semicolons.

```
// app/api/users/route.ts
import { dbConnect } from '@/lib/dbConnect'
import User from '@/models/User'

export async function GET () {
  try {
    await dbConnect()
    const users = await User.find()
    return Response.json(users, { status: 200 })
  } catch (error) {
    console.error(error)
    return Response.json({ error: 'Failed to fetch users' },
       { status: 500 })
  }
}
```

Database Connection (lib/dbConnect.ts)

StandardJS Rules:

- Utility function in `lib/`.
- Named export.
- Singleton pattern for connection.

```
// lib/dbConnect.ts
import mongoose from 'mongoose'

const MONGO_URI = process.env.MONGO_URI as string

if (!MONGO_URI) {
  throw new Error('Please define MONGO_URI in .env')
}

let cached = global.mongoose as {
  conn: typeof mongoose | null
  promise: Promise<typeof mongoose> | null
}

if (!cached) {
  cached = global.mongoose = { conn: null, promise: null }
}

export async function dbConnect () {
  if (cached.conn) return cached.conn

  if (!cached.promise) {
    cached.promise = mongoose.connect(MONGO_URI).then(m => m)
  }
  cached.conn = await cached.promise
  return cached.conn
}
```

Schema (models/User.ts)

StandardJS Rules:

- One model per file.
- PascalCase names.

```
// models/User.ts
import mongoose, { Schema, Document } from 'mongoose'

export interface IUser extends Document {
  name: string
  email: string
}

const UserSchema: Schema = new Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true }
  },
  { timestamps: true }
)

export default mongoose.models.User ||
  mongoose.model<IUser>('User', UserSchema)
```

Components (components/)

StandardJS Rules:

- PascalCase file and function names.
- Pure functional components.
- Typed props.

```
// components/Button.tsx
import React from 'react'

interface ButtonProps {
  label: string
  onClick: () => void
}

export default function Button ({ label, onClick }:
   ButtonProps) {
  return (
    <button
      type='button'
      onClick={onClick}
      className='rounded bg-blue-500 px-4 py-2 text-white'
    >
      {label}
```

```
    </button>
  )
}
```

4 Comparison of Coding Standards

Feature	Airbnb	Google	StandardJS
Semicolons	Required	Required	Forbidden
String Quotes	Single quotes	Single quotes	Single quotes
Trailing Commas	Always for multi-line arrays and objects	Recommended	Never
Function Declaration	Prefers function expressions for anonymous functions	Prefers function declarations	Function declarations are fine
Naming Conventions	`camelCase` for variables/functions, `PascalCase` for classes/components	Similar to Airbnb, but stricter on JSDoc for exports	Similar to Airbnb
Unused Variables	Error (strict)	Error (strict)	Error (strict)
Configuration	Highly configurable via `.eslintrc`	Configurable	Zero-configuration (enforced)

Table 1: Key Differences Between Style Guides

5 Recommendation for ShebaBondhu

After a thorough review of the Airbnb, Google, and StandardJS coding standards, we recommend the adoption of the **Airbnb Style Guide** for the ShebaBondhu project. This decision is based on the following key factors that align with our project's goals and technology stack:

1. Alignment with the React Ecosystem

The Airbnb style guide is overwhelmingly popular within the React and Next.js communities. Its conventions are considered a de-facto standard for

modern React development. By adopting it, we align ShebaBondhu with the broader ecosystem, making it easier to onboard new developers familiar with React and leverage a vast amount of community-created tooling, documentation, and solutions.

2. Clarity and Reduced Ambiguity

One of the most significant advantages of the Airbnb guide is its strictness, which minimizes ambiguity. For instance, its requirement for semicolons and trailing commas prevents common JavaScript pitfalls and version control conflicts. While StandardJS aims for simplicity by omitting semicolons, this can occasionally lead to unexpected behavior. For a critical healthcare application like ShebaBondhu, prioritizing code safety and predictability is paramount.

3. Excellent Tooling and Automation

Airbnb provides a comprehensive ESLint configuration package (`eslint-config-airbnb`) that makes automated enforcement of the style guide seamless. This allows our development team to focus on building features while the linter automatically flags and often fixes stylistic inconsistencies, ensuring a clean and uniform codebase with minimal manual effort.

Conclusion

The Airbnb Style Guide offers the best balance of strictness, community support, and alignment with our Next.js and TypeScript stack. Its widespread adoption in the React world ensures that our codebase remains modern, maintainable, and easy for new developers to adopt. We believe it provides the ideal foundation for building a robust and scalable application like ShebaBondhu.

References

- Airbnb JavaScript Style Guide: https://github.com/airbnb/javascript
- Google JavaScript Style Guide: https://google.github.io/styleguide/jsguide.html
- StandardJS Guide: https://standardjs.com/