NABU Code Standard (Version 1.0)

Clear, simple, enforceable coding rules for the whole team.

---

1. General Style Rules

1.1 Formatting

- 2 spaces indentation
- Semicolons **required**
- Max line length: **100 characters**
- Prettier + ESLint enforced
- No tabs, consistent spacing everywhere

1.2 Naming Conventions

Item	Convention	Example
Variables	camelCase	userId, accessToken
Functions	camelCase	createUser()
Classes	PascalCase	UserController
Backend files	kebab-case.js	auth-controller.js
Frontend files	PascalCase.vue	LoginForm.vue
Environment variables	UPPER_CASE	DB_PASSWORD

---

2. Backend (Node.js + Express)

2.1 Project Structure

```
server/
│— controllers/
│— routes/
│— services/
│— middleware/
│— db/
│— utils/
│— tests/
```

2.2 Controllers

- Controllers **must not contain business logic**

- Every controller wrapped in try/catch
- Standard API response format:

```
{
  "success": true,
  "data": {},
  "error": null
}
On error:
{
  "success": false,
  "error": { "message": "string", "code": 400 }
}
```

2.3 Services

- Contain the actual logic (validation, DB actions)
- Never use req or res
- Prefer pure functions

---

3. Error & Response Handling

3.1 Throw errors — don't send responses inside services

Bad

return res.status(400).json({ error: 'Missing email' });

Good

throw new ApiError(400, 'Missing email');

3.2 Global Error Middleware

```
app.use((err, req, res, next) => {
  res.status(err.code || 500).json({
    success: false,
    error: { message: err.message, code: err.code }
  });
});
```

---

4. Testing (Jest + Supertest)

4.1 Test Structure

server/tests/

```
├── auth.test.js
├── user.test.js
├── db.test.js
```

4.2 Required Tests

Each endpoint must have at least:

- success case
- validation error case
- database failure case

4.3 Test Example

```
describe("POST /api/signup", () => {
 it("should return 400 if email is missing", async () => {
  const res = await request(app).post("/api/signup").send({});
  expect(res.status).toBe(400);
  expect(res.body.error).toBeDefined();
 });
});
```

---

5. Git Workflow

5.1 Branch Names

```
feature/signup-endpoint
bugfix/login-nullpointer
refactor/user-service
tests/auth-tests
```

5.2 Commit Messages

- English
- Short and meaningful

Examples:

```
feat(auth): implement JWT login
fix(db): handle null user object
test(user): add missing validation tests
```

---

6. Security Rules

- Never log passwords or sensitive data
- Use bcrypt for password hashing
- Always validate inputs (Joi/Zod preferred)
- Never use raw SQL strings → use prepared statements
- JWT tokens: HS256, expiry max **1 hour**
- .env must never be committed

---

7. Frontend (React or Vue)

7.1 Component Rules

- Use Composition API (Vue) or functional components (React)
- Props & emits must be typed (TypeScript recommended)
- Components should stay below **250 lines**
- No inline CSS → use Tailwind or scoped CSS

7.2 API Calls

- All API requests must be placed in /services/api.js or /lib/api.ts
- Standard API return shape:

```
return { success, data, error };
```

---

8. Documentation

8.1 JSDoc Required for Services

```
/**
 * Creates a new user
 * @param {Object} userData
 * @returns {Promise<Object>}
 */
```

8.2 README Must Include

- Setup
- Environment variables
- Start scripts
- Testing instructions