Coding Standards: Drop — Fintech Payment App

Coding Standards: Drop — Fintech Payment App

> **Project:** Drop — Remittance + QR Payments **Version:** 1.0 **Date:** 2026-02-23
> **Author:** John (AI Director) **Status:** Approved **Reviewers:** Alem Bašić (CEO)

Document History

Version	Date	Author	Changes
0.1	2026-02-23	John	Initial standards — TypeScript strict, Next.js App Router, fintech security rules

---

Purpose

These standards apply to all code in **Drop**. When automated tools enforce a rule, the tool wins. When in doubt, optimize for readability — the next AI agent reading your code is trying to understand intent, not guess it.

Non-negotiable: All new code must pass TypeScript strict check, ESLint, and Prettier formatting before merge. No exceptions.

Fintech non-negotiable: Parameterized SQL everywhere. No stored balances. No stored card numbers or CVV. These are encoded as automated tests — failing these means P0.

---

1. Language-Specific Conventions

1.1 Naming Conventions

TypeScript (Drop's primary language):

Type	Convention	Example
Variables	`camelCase`	`userId`, `kycStatus`
Functions	`camelCase`	`hashPassword()`, `calculateFee()`
Classes	`PascalCase`	`TransactionService`
Interfaces	`PascalCase` (no `I` prefix)	`User`, `TransactionResult`
Types	`PascalCase`	`KycStatus`, `CurrencyCode`
Enums	`PascalCase`	`TransactionType`, `KycStatus`
Constants	`UPPER_SNAKE_CASE`	`REMITTANCE_FEE_RATE`, `MIN_AMOUNT_NOK`
Files — components	`PascalCase.tsx`	`SendMoneyForm.tsx`
Files — utilities	`camelCase.ts`	`calculateFee.ts`, `verifyJWT.ts`
Files — API routes	`route.ts` (Next.js convention)	`app/api/auth/login/route.ts`
Test files	`{name}.test.ts`	`auth.test.ts`, `db.test.ts`
E2E test files	`{name}.spec.ts`	`user-flows.spec.ts`

Drop-specific constants:

```
// Fee rates (business rules — change requires CEO approval + ADR)
const REMITTANCE_FEE_RATE = 0.005;  // 0.5%
const QR_MERCHANT_FEE_RATE = 0.01;  // 1%
const MIN_REMITTANCE_NOK = 100;
const MAX_REMITTANCE_NOK = 50_000;

// Supported corridors
const SUPPORTED_CURRENCIES = ['RSD', 'BAM', 'PKR', 'TRY', 'PLN', 'EUR'] as const;
type CurrencyCode = typeof SUPPORTED_CURRENCIES[number];
```

1.2 File Organization

```
src/drop-app/src/
├── app/
```

```
│   ├── api/                 # Next.js API Routes
│   │   ├── auth/
│   │   │   ├── login/route.ts
│   │   │   ├── register/route.ts
│   │   │   └── me/route.ts
│   │   ├── transactions/
│   │   │   ├── remittance/route.ts
│   │   │   └── qr-payment/route.ts
│   │   └── rates/
│   │       ├── route.ts           # GET /api/rates
│   │       └── [currency]/route.ts # GET /api/rates/RSD
│   └── (app)/               # Protected pages (dashboard, send-money, etc.)
├── lib/
│   ├── auth.ts              # JWT, bcrypt, session helpers
│   ├── db.ts                # SQLite/PostgreSQL client
│   ├── validate.ts          # Zod schemas for input validation
│   └── fee.ts               # Fee calculation functions
└── components/
    ├── drop-logo.tsx        # Brand logo component
    └── ui/                  # Shared UI components
```

Rules:

- One route handler per file (`route.ts` in Next.js App Router)
- Business logic extracted to `lib/` — never inline in route handlers
- Test files co-located with source files in `__tests__/`

1.3 Import Ordering

```
// 1. Node built-ins
import { cookies } from 'next/headers';

// 2. External dependencies (node_modules)
import { z } from 'zod';
import * as bcrypt from 'bcrypt';
import { SignJWT, jwtVerify } from 'jose';

// 3. Internal — absolute paths (@/ alias)
import { db } from '@/lib/db';
import { verifyJWT } from '@/lib/auth';
```

```
import type { User } from '@/lib/types';

// 4. Internal — relative paths
import { validateRemittanceBody } from './validate';
```

Enforced by: ESLint `import/order` rule

1.4 Error Handling Patterns

```
// PREFERRED — consistent HTTP error responses in Next.js API routes
export async function POST(request: Request): Promise<Response> {
  let body: unknown;
  try {
    body = await request.json();
  } catch {
    return Response.json({ error: 'Invalid JSON body' }, { status: 400 });
  }

  const validation = remittanceSchema.safeParse(body);
  if (!validation.success) {
    return Response.json(
      { error: 'Validation failed', details: validation.error.flatten() },
      { status: 422 }
    );
  }

  // Business logic in lib/ — throws typed errors
  try {
    const result = await processRemittance(validation.data);
    return Response.json(result, { status: 201 });
  } catch (error) {
    if (error instanceof InsufficientBalanceError) {
      return Response.json({ error: 'Insufficient balance' }, { status: 402 });
    }
    if (error instanceof KycRequiredError) {
      return Response.json({ error: 'KYC verification required' }, { status: 403 });
    }
    // Never expose internal errors
    console.error('Unhandled error in remittance:', error);
    return Response.json({ error: 'Internal server error' }, { status: 500 });
```

```
  }
}
```

Rules:

- NEVER return stack traces or DB error messages to the client
- NEVER use `any` type — TypeScript strict mode enforces this
- NEVER swallow errors silently
- Use 402 for Insufficient Balance (fintech convention), not 400
- Error messages in Norwegian for user-facing validation

---

2. Code Formatting

2.1 Formatter

Language	Tool	Config File	Enforced In
TypeScript / JavaScript	Prettier	`.prettierrc`	CI + pre-commit hook
JSON / YAML	Prettier	`.prettierrc`	CI + pre-commit hook

Key formatting rules:

- Indentation: 2 spaces
- Max line length: 100 characters
- Semicolons: required
- Trailing commas: `all` (ES2020)
- Quotes: single quotes

Auto-format on save: Enabled via `.vscode/settings.json` (see local-development-setup.md)

2.2 Linter

Language	Tool	Config	Rules Severity
TypeScript	ESLint + TypeScript-ESLint	`.eslintrc.json`	Error = blocks CI

Linting rules that are errors (block CI):

- `@typescript-eslint/no-explicit-any` — no `any` types
- `no-console` (except `console.error` in server code)
- `@typescript-eslint/no-unused-vars`
- SQL string concatenation (custom Drop rule)

Disable linting inline (sparingly):

```
// eslint-disable-next-line @typescript-eslint/no-explicit-any -- External library returns
untyped response
const rawResponse: any = await externalLibrary.call();
```

Every inline disable must have a comment explaining why.

---

3. Git Conventions

3.1 Commit Message Format

Standard: Conventional Commits

Format:

```
<type>(<scope>): <description>

[optional body]

[optional footer(s)]
```

Types:

Type	When to Use
`feat`	New feature
`fix`	Bug fix
`docs`	Documentation only
`style`	Formatting changes (no logic change)
`refactor`	Code change that neither fixes a bug nor adds a feature
`test`	Adding or updating tests
`chore`	Build system, dependency updates, CI changes
`perf`	Performance improvements
`security`	Security fix or hardening

Drop examples:

```
feat(auth): add bcrypt rounds configuration via env var
fix(transactions): prevent double-spend with per-user lock
test(db): add assertion that users table has no balance column
security(middleware): add CSRF protection to all mutating endpoints
chore: upgrade Next.js to 15.x patch
```

Breaking changes:

```
feat(api)!: rename /api/transactions/send to /api/transactions/remittance

BREAKING CHANGE: All clients must update API calls to new path
```

3.2 Branch Naming Convention

Type	Pattern	Example
Feature	`feature/MC-{task-id}-description`	`feature/MC-042-rate-limiting`
Bug fix	`fix/MC-{task-id}-description`	`fix/MC-051-double-spend`
Hotfix	`hotfix/MC-{task-id}-description`	`hotfix/MC-060-jwt-bypass`
Security	`security/MC-{task-id}-description`	`security/MC-070-csrf-fix`
Chore	`chore/MC-{task-id}-description`	`chore/MC-080-upgrade-deps`

Rules:

- Lowercase only; hyphens, not underscores
- Always include Mission Control task ID (`MC-{id}`)

3.3 PR Title & Description Format

Title format: Same as commit message: `type(scope): description`

Required PR description sections:

1. **What**: What does this PR do? (bullet points)
2. **Why**: Which Mission Control task / acceptance criterion?
3. **Pass-through model check**: Does this change touch DB schema? If yes, confirm no balance/CVV columns added
4. **Tests**: What tests were added/modified?
5. **Security**: Does this touch auth, payments, or input validation?

3.4 PR Size Guidelines

Size	Lines Changed	Status
Small	< 200	Ideal — review same session
Medium	200-400	Acceptable
Large	400-800	Needs justification — split if possible
Extra Large	> 800	Exceptional only — must be pre-approved by John

---

4. Code Review Guidelines

4.1 What to Look For (Validator Agent Checklist)

Reviewers must check:

- ☐ Pass-through model invariant: no `balance` column; no `card_number` or `cvv`
- ☐ Parameterized SQL — no string interpolation in any DB query
- ☐ bcrypt used for passwords — SHA-256 explicitly forbidden
- ☐ JWT secret not hardcoded; not in response body
- ☐ Rate limiting applies to new auth/transaction endpoints
- ☐ Input validation (Zod schema) on all new request bodies
- ☐ Error messages don't leak internal state
- ☐ Fee calculations correct (0.5% remittance; 1% QR)
- ☐ Tests cover the change and edge cases

Reviewers should NOT block on:

- Personal style preferences covered by Prettier/ESLint (the linter decides)
- Minor refactors not related to the PR's scope

4.2 Review Turnaround

PR Type	First review
Security / Critical fix	Same session
Standard feature	Within 1 session

4.3 Approval Requirements

Branch	Required Approvals
`main`	Validator agent approval + John (AI Director) sign-off
Feature branches	Validator agent approval
Hotfix	John (AI Director) or Alem Bašić (CEO) approval — async OK

4.4 Constructive Review Feedback

- `nit:` — Minor issue, not a blocker
- `question:` — Need clarification
- `suggestion:` — Improvement idea (not required)
- `blocker:` — Must be fixed before merge (e.g., SQL injection risk, bcrypt bypass)

---

5. Testing Standards

5.1 Test Naming Conventions

```
// Vitest format: describe → it('should [behavior] when [condition]')
describe('hashPassword', () => {
  it('should return a bcrypt hash starting with $2', async () => { ... });
  it('should reject SHA-256 hashes at verify time', async () => { ... });
  it('should reject passwords longer than 1000 characters', async () => { ... });
});

describe('calculateRemittanceFee', () => {
  it('should return 5 NOK fee for 1000 NOK amount', () => { ... });
  it('should reject amounts below 100 NOK', () => { ... });
});
```

5.2 Test Organization

```
src/drop-app/__tests__/
├── auth.test.ts              # Unit: bcrypt, JWT, password validation
├── validation.test.ts        # Unit: input validation, XSS, injection
├── transactions.test.ts      # Unit: fee calculation, transaction logic
├── rates.test.ts             # Unit: exchange rates
```

```
├── api-endpoints.test.ts     # Integration: all 26 API routes
├── db.test.ts                # Integration: schema compliance, FK constraints
├── middleware.test.ts        # Integration: rate limiting, auth, CSRF
├── api-benchmarks.test.ts    # Performance: bcrypt timing, DB queries
├── feature-flags.test.ts     # Unit: feature flag behavior
├── regression-suite.test.ts  # Regression: critical path smoke
├── sumsub-integration.test.ts # Integration: KYC webhook mock
├── cards-integration.test.ts  # Integration: cards (feature-flagged)
└── e2e/
    ├── user-flows.spec.ts    # E2E: registration, login
    ├── full-flows.spec.ts    # E2E: remittance, QR payment
    └── input-chaos.spec.ts   # E2E: 20+ validation edge cases
```

5.3 Mocking Guidelines

```
// PREFERRED: Mock BaaS at the module level (NEXT_PUBLIC_SERVICE_MODE=mock)
// The mock mode is configured via env var — no vi.mock() needed for BaaS

// PREFERRED for unit tests: vi.mock for db queries
vi.mock('@/lib/db', () => ({
  db: {
    query: vi.fn(),
    run: vi.fn(),
  }
}));

// For integration tests: use real SQLite in-memory DB
// (configured in vitest.config.ts — no mock needed)

// NEVER mock the pass-through model assertions in db.test.ts
// NEVER mock the bcrypt rejection tests in auth.test.ts
```

5.4 Coverage Requirements

Layer	Lines	Branches	Notes
Auth module	100%	100%	Fintech security — strictly enforced
Transaction logic	100%	100%	Fee calculation — strictly enforced

Layer	Lines	Branches	Notes
API handlers	≥ 80%	≥ 70%	
Input validation	≥ 90%	≥ 85%	Security-critical
DB layer	≥ 90%	—	Compliance assertions required
Overall minimum	≥ 80%	—	CI gate

---

6. Documentation Standards

6.1 When to Add Comments

```
// GOOD — explains WHY (not obvious from code)
// Bcrypt rounds set to 12 (not 10) per NFR-SEC02 fintech standard.
// Using 10 rounds would be faster but falls below security threshold.
const BCRYPT_ROUNDS = parseInt(process.env.BCRYPT_ROUNDS ?? '12', 10);

// GOOD — documents non-obvious business rule
// Fee is calculated on gross amount, not total debit (gross + fee)
// This is required by Drop's pass-through model (ADR-003)
const fee = amount * REMITTANCE_FEE_RATE;

// BAD — restates what code says
// Multiply amount by fee rate
const fee = amount * REMITTANCE_FEE_RATE;
```

Comment when:

- A fintech business rule is implemented (always cite the rule)
- A security decision was made (e.g., why bcrypt rounds = 12)
- A workaround exists for an external system quirk

6.2 ADR Format

Write an Architecture Decision Record when:

- Changing the database (SQLite → PostgreSQL)
- Changing the authentication mechanism (DOB → BankID)
- Modifying the fee model (new corridor, new fee rate)

- Adding a new BaaS partner

ADR location: `comms/decisions/YYYY-MM-DD-decision-title.md`

6.3 API Documentation

When adding a new API endpoint, update `docs/backend/API-REFERENCE.md`:

- Method + path
- Authentication required (Yes/No)
- Request body schema (Zod type → JSON example)
- Response schema (success + all error codes)

---

7. Security Coding Practices — Drop Fintech Standards

Practice	Rule
SQL queries	NEVER string interpolation. Always parameterized queries (better-sqlite3 `?` placeholders)
Password hashing	bcrypt ONLY with 12 rounds in production. SHA-256 hashes are REJECTED at login
JWT	`jose` library only. Secret via `JWT_SECRET` env var. Fail fast if missing
Input validation	Zod schemas on ALL API route request bodies. Server-side only — never trust client
Pass-through model	NEVER add `balance` column to users. NEVER add `card_number` or `cvv` to cards
Rate limiting	DB-backed rate limiter (not in-memory). Auth: 10/min; General: 60/min
CSRF	CSRF token required on all POST/PATCH/DELETE endpoints
Cookie settings	JWT: httpOnly, SameSite=Strict, Secure in production
Logging	NEVER log passwords, JWT tokens, card numbers, or full phone numbers
Error messages	NEVER expose DB errors, stack traces, or internal state to users
Dependencies	Run `npm audit` before adding any dependency. No HIGH/CRITICAL CVEs

Security review trigger: Any PR touching auth, payments, DB schema, or input validation must be reviewed by Validator agent AND flagged to John (AI Director).

---

8. Performance Coding Practices

Practice	Rule
Database queries	One bounded query per API call where possible. Use JOIN instead of N+1
Pagination	Never load all transactions in history — always paginate (default: 20 per page)
bcrypt	Password max length = 1,000 chars (validation before bcrypt to prevent DoS)
SQLite	Use WAL mode for concurrent reads. Serialize writes (one at a time)
Next.js	Use Server Components for static/cached data; Client Components for interactive UI
Bundle size	Run `npm run build` and check bundle analyzer before UI changes

---

Related Documents

- Developer Onboarding Guide
- Local Development Setup
- Test Strategy
- Definition of Done

Approval

Role	Name	Date	Signature
Author	John (AI Director)	2026-02-23	Approved (AI)
QA Lead	Validator Agent	2026-02-23	Approved (AI)
Tech Lead	John	2026-02-23	Approved
CEO (Alem)	Alem Bašić	TBD	

Revision #6
Created 2026-02-23 12:06:33 UTC by John
Updated 2026-07-05 20:03:12 UTC by John