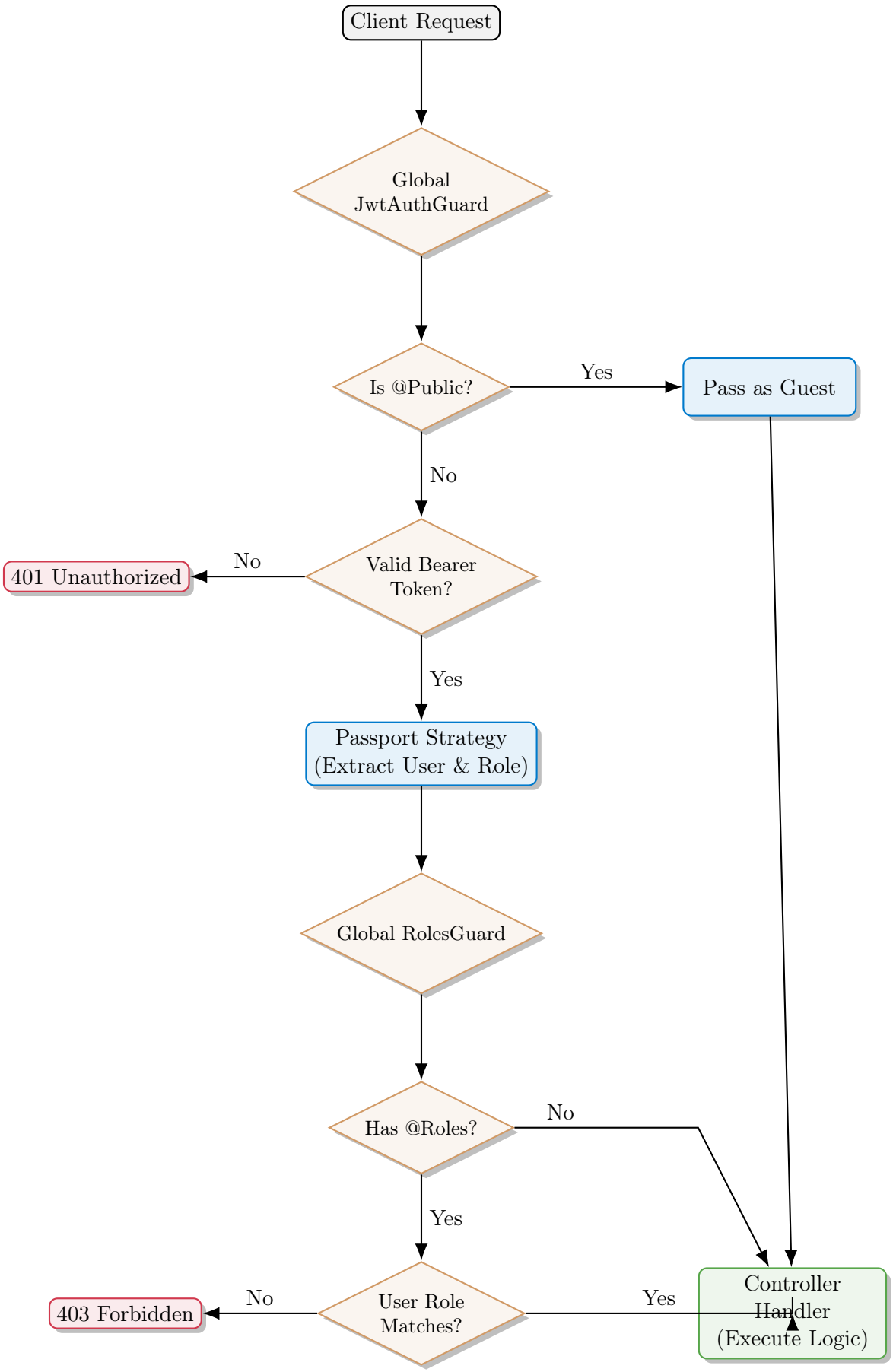
JWT & RBAC Security Walkthrough

Securing the **YouShop Backend**

This document outlines the security architecture using **JSON Web Tokens (JWT)** and **Role-Based Access Control (RBAC)** to secure API endpoints.

🔄 The Security Flow

Here is the decision logic for every incoming request:


Client Request
Global JwtAuthGuard
Is @Public?
Yes
Pass as Guest
No
Valid Bearer Token?
No
401 Unauthorized
Yes
Passport Strategy (Extract User & Role)
Global RolesGuard
Has @Roles?
No
Yes
User Role Matches?
No
403 Forbidden
Yes
Controller Handler (Execute Logic)

Key Components

1. The ID Card: JwtStrategy

File: `jwt.strategy.ts`
Responsible for identifying the user. It validates the signature and expiration of the token. If valid, it attaches the user (id, email, **role**) to the request object.

Extracting the Role

```
// Inside validate():
return {
  userId: payload.sub,
  email: payload.email,
  role: payload.role // <--- Crucial!
};
```

2. The Gatekeeper: JwtAuthGuard

File: `jwt-auth.guard.ts`
Runs globally on every request.

1. Check if route has `@Public()`.
2. If **Yes**, skip authentication.
3. If **No**, force validation via `JwtStrategy`.

3. The Enforcer: RolesGuard

File: `roles.guard.ts`
Runs globally but only acts if roles are required.

1. Checks for `@Roles(...)` decorator.
2. If found, compares required role with `req.user.role`.
3. **Mismatch?** Throws `403 Forbidden`.

4. The Labels: Decorators

- `@Public()`: Tells the Gatekeeper to stand down.
- `@Roles(Role.ADMIN)`: Tells the Enforcer exactly who is allowed.

Bringing it together: CategoryController

The `CategoryController` demonstrates how we mix and match these guards.

1. Public Route (Read-Only)

Any user (even guests) can list categories. The `@Public` decorator bypasses the check.

TypeScript Snippet

```typescript
@Public() // <--- Bypasses JwtAuthGuard
@Get()
findAll() {
  return this.categoryService.findAll();
}
```

2. Protected Route (Admin Only)

Only an Admin with a valid token can create categories.

TypeScript Snippet

```typescript
@Post()
@Roles(Role.ADMIN) // <--- Enforced by RolesGuard
create(@Body() createCategoryDto: CreateCategoryDto) {
  return this.categoryService.create(createCategoryDto);
}
```

Logic Summary

- **No Token:** `JwtAuthGuard` blocks it → **401 Unauthorized**.
- **Token + Role CUSTOMER:** `RolesGuard` blocks it → **403 Forbidden**.
- **Token + Role ADMIN:** Logic executes → **201 Created**.