Documentation node js fastfiy framework backend  
  
  
File Structured  
  
A screenshot of a computer program

AI-generated content may be incorrect.

Backend

cd backend

npm init -y

npm install fastify @fastify/jwt @fastify/cors @fastify/mysql @fastify/cookie bcryptjs uuid

npm install --save-dev nodemon dotenv

.env

DB\_HOST=127.0.0.1

DB\_USER=root

DB\_PASSWORD="dex123"

DB\_NAME=delete\_practice

DB\_PORT=3306

JWT\_SECRET=mystrongsecret

PORT=4000

src/server.js

// server.js

// Main entry point of Fastify server

require('dotenv').config();                   // Load .env variables (DB, JWT, PORT)

const Fastify = require('fastify');           // Import Fastify

const cors = require("@fastify/cors");

// Plugins

const mysqlPlugin = require('./plugins/mysql'); // MySQL plugin

const jwtPlugin = require('./plugins/jwt');     // JWT plugin

// Routes

const authRoutes = require('./routes/authRoutes'); // Auth routes

const itemRoutes = require('./routes/itemRoutes'); // Item routes

const usersRoutes = require("./routes/usersRoutes"); // User routes

// Create Fastify instance

const fastify = Fastify({ logger: true });

// ✅ Register CORS (must come before routes)

fastify.register(cors, {

  origin: ["http://localhost:5173", "http://127.0.0.1:5173"], // your frontend dev URL

  credentials: true,  // allow Authorization headers / cookies

});

// Register plugins

fastify.register(mysqlPlugin);  // Connect MySQL

fastify.register(jwtPlugin);    // Enable JWT

// Register routes

fastify.register(authRoutes);   // Authentication routes

fastify.register(itemRoutes);   // CRUD item routes

fastify.register(usersRoutes);  // User routes

// Start server

const PORT = process.env.PORT || 4000;

fastify.listen({ port: PORT, host: '0.0.0.0' }, (err, address) => {

  if (err) {

    fastify.log.error(err);     // Log errors

    process.exit(1);            // Exit app if failed

  }

  fastify.log.info(`Server running at ${address}`); // Log success

});

Src/migrate.js

// migrate.js

// Script to auto-drop and recreate database tables (like Laravel migrate:fresh)

require('dotenv').config();                  // Load .env file

const mysql = require('mysql2/promise');     // Import mysql2 with promise support

async function migrate() {

  // Create MySQL connection

  const conn = await mysql.createConnection({

    host: process.env.DB\_HOST,               // Database host (127.0.0.1 by default)

    user: process.env.DB\_USER,               // Database username

    password: process.env.DB\_PASSWORD,       // Database password

    database: process.env.DB\_NAME,           // Database name

    port: process.env.DB\_PORT                // Database port (default 3306)

  });

  console.log('✅ Connected to database');

  // Drop existing tables (order matters: drop items first since it could depend on users in future)

  await conn.query(`DROP TABLE IF EXISTS items`);

  await conn.query(`DROP TABLE IF EXISTS users`);

    await conn.query(`DROP TABLE IF EXISTS blacklisted\_tokens`);

  console.log('🗑️ Old tables dropped (if they existed)');

  // Create users table (fresh)

  await conn.query(`

    CREATE TABLE users (

      id CHAR(36) PRIMARY KEY,               -- UUID v4 (always 36 chars)

      email VARCHAR(255) NOT NULL UNIQUE,    -- Unique email

      password VARCHAR(255) NOT NULL,        -- Hashed password (bcrypt)

      refresh\_token TEXT,                    -- store latest refresh token

      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Timestamp auto generated

    )

  `);

  console.log('✅ Users table created');

  // Create items table (fresh)

  await conn.query(`

    CREATE TABLE items (

      id CHAR(36) PRIMARY KEY,               -- UUID v4

      name VARCHAR(255) NOT NULL,            -- Item name

      description TEXT,                      -- Optional description

      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Auto timestamp

    )

  `);

  console.log('✅ Items table created');

  await conn.end(); // Close DB connection

  console.log('🎉 Migration complete (fresh tables ready)');

}

// Run migration

migrate().catch(err => {

  console.error('❌ Migration failed:', err);

});

src/plugins/mysql.js

// mysql.js

// This plugin connects Fastify to MySQL

const fp = require('fastify-plugin');      // Allows us to create Fastify plugins

const fastifyMysql = require('@fastify/mysql'); // MySQL plugin for Fastify

async function mysqlPlugin(fastify, opts) {

  // Register MySQL connection with environment variables

  fastify.register(fastifyMysql, {

    promise: true,                        // Use async/await

    host: process.env.DB\_HOST,            // DB host from .env

    user: process.env.DB\_USER,            // DB username

    password: process.env.DB\_PASSWORD,    // DB password

    database: process.env.DB\_NAME,        // DB name

    port: process.env.DB\_PORT             // DB port

  });

}

// Export plugin

module.exports = fp(mysqlPlugin);

// This plugin connects Fastify to MySQL

src/plugins/jwt.js

// This plugin handles JWT authentication

const fp = require('fastify-plugin');   // To create a reusable Fastify plugin

const fastifyJwt = require('@fastify/jwt'); // JWT support

async function jwtPlugin(fastify, opts) {

  // Register JWT with secret key from .env

  fastify.register(fastifyJwt, { secret: process.env.JWT\_SECRET });

  // Middleware-like function: verify token before accessing protected routes

  fastify.decorate('authenticate', async function (request, reply) {

    try {

      await request.jwtVerify(); // Verifies JWT token in headers

    } catch (err) {

      reply.code(401).send({ message: 'Unauthorized' }); // Return error if invalid

    }

  });

}

// Export plugin

module.exports = fp(jwtPlugin);

// This plugin handles JWT authentication

src/utils/hash.js

// Helper functions for password hashing & comparison

const bcrypt = require('bcryptjs'); // Library for hashing passwords

// Hash plain password before saving to DB

async function hashPassword(plain) {

  const salt = await bcrypt.genSalt(10);  // Generate salt

  return bcrypt.hash(plain, salt);        // Return hashed password

}

// Compare plain password with hashed one in DB

async function comparePassword(plain, hashed) {

  return bcrypt.compare(plain, hashed);   // Return true/false

}

module.exports = { hashPassword, comparePassword };

// Helper functions for password hashing & comparison

src/routes/authRoutes.js

// authRoutes.js

// Defines authentication routes

const AuthController = require("../controllers/authController"); // Import controller

async function authRoutes(fastify, opts) {

  // Route: Register user

  fastify.post("/api/register", (req, reply) =>

    AuthController.registerUser(fastify, req, reply)

  );

  // Route: Login user

  fastify.post("/api/login", (req, reply) =>

    AuthController.loginUser(fastify, req, reply)

  );

  // Route: Refresh token

  fastify.post("/api/refresh", (req, reply) =>

    AuthController.refreshToken(fastify, req, reply)

  );

  // Route: Logout user

  fastify.post("/api/logout", (req, reply) =>

    AuthController.logoutUser(fastify, req, reply)

  );

}

module.exports = authRoutes; // Export route

Src/model/authModel.js

// authModel.js

// Handles DB queries for "users" table

// Create new user in DB

async function createUser(fastify, { id, email, password }) {

  const conn = await fastify.mysql.getConnection(); // Open DB connection

  try {

    await conn.query(

      "INSERT INTO users (id, email, password) VALUES (?, ?, ?)",

      [id, email, password]

    );

  } finally {

    conn.release(); // Always release DB connection

  }

}

// Find user by email

async function findUserByEmail(fastify, email) {

  const conn = await fastify.mysql.getConnection();

  try {

    const [rows] = await conn.query("SELECT \* FROM users WHERE email = ?", [

      email,

    ]);

    return rows[0]; // Return user record

  } finally {

    conn.release();

  }

}

// Save (or remove) refresh token

async function saveRefreshToken(fastify, userId, token) {

  const conn = await fastify.mysql.getConnection();

  try {

    await conn.query("UPDATE users SET refresh\_token = ? WHERE id = ?", [

      token,

      userId,

    ]);

  } finally {

    conn.release();

  }

}

// Find user by refresh token

async function findUserByRefreshToken(fastify, token) {

  const conn = await fastify.mysql.getConnection();

  try {

    const [rows] = await conn.query(

      "SELECT \* FROM users WHERE refresh\_token = ?",

      [token]

    );

    return rows[0];

  } finally {

    conn.release();

  }

}

module.exports = {

  createUser,

  findUserByEmail,

  saveRefreshToken,

  findUserByRefreshToken,

};

Src/controllers/authController.js

 **Login** → accessToken (15m) + refreshToken (1d).

 **Before 15m** → use accessToken directly.

 **After 15m but before 1d** → refreshToken generates a new 15m accessToken.

 **After 1d** → refreshToken expired → must re-login.

// controllers/authController.js

// Handles authentication logic (cookie-based)

const { v4: uuidv4 } = require("uuid");

const { hashPassword, comparePassword } = require("../utils/hash");

const UserModel = require("../models/authModel");

// REGISTER a new user

async function registerUser(fastify, request, reply) {

  const { email, password } = request.body;

  if (!email || !password) {

    return reply.code(400).send({ message: "Email & password required" });

  }

  const hashed = await hashPassword(password);

  const id = uuidv4();

  await UserModel.createUser(fastify, { id, email, password: hashed });

  return reply.send({ message: "User registered successfully" });

}

// LOGIN existing user

async function loginUser(fastify, request, reply) {

  const { email, password } = request.body;

  if (!email || !password) {

    return reply.code(400).send({ message: "Email & password required" });

  }

  const user = await UserModel.findUserByEmail(fastify, email);

  if (!user) return reply.code(401).send({ message: "Invalid credentials" });

  const valid = await comparePassword(password, user.password);

  if (!valid) return reply.code(401).send({ message: "Invalid credentials" });

  const accessToken = fastify.jwt.sign({ id: user.id, email: user.email }, { expiresIn: "15m"});

  const refreshToken = fastify.jwt.sign({ id: user.id, email: user.email }, { expiresIn: "1day"});

  // Set cookies

  reply

    .setCookie("accessToken", accessToken, {

      httpOnly: true,

      secure: false, // 🔑 true in production HTTPS

      sameSite: "Strict",

      path: "/",

      maxAge: 15 \* 60, // 15 minutes

    })

    .setCookie("refreshToken", refreshToken, {

      httpOnly: true,

      secure: false, // 🔑 true in production HTTPS

      sameSite: "Strict",

      path: "/",

      maxAge: 24 \* 60 \* 60, // 1 day

    })

    .send({ message: "Login successful" });

}

// REFRESH token

async function refreshToken(fastify, request, reply) {

  const { refreshToken } = request.cookies; // ✅ from cookies

  if (!refreshToken) return reply.code(401).send({ error: "No refresh token" });

  try {

    const payload = fastify.jwt.verify(refreshToken);

    const newAccessToken = fastify.jwt.sign({ id: payload.id, email: payload.email }, { expiresIn: "15m" });

    reply

      .setCookie("accessToken", newAccessToken, {

        httpOnly: true,

        secure: false,

        sameSite: "Strict",

        path: "/",

        maxAge: 15 \* 60,

      })

      .send({ message: "Token refreshed" });

  } catch (err) {

    return reply.code(403).send({ error: "Invalid refresh token" });

  }

}

// LOGOUT

async function logoutUser(fastify, request, reply) {

  reply

    .clearCookie("accessToken", { path: "/" })

    .clearCookie("refreshToken", { path: "/" })

    .send({ message: "Logged out successfully" });

}

module.exports = { registerUser, loginUser, refreshToken, logoutUser };

Src/routes/usersRoutes.js

// usersRoutes.js

// Defines routes protected by JWT middleware with cookies)

const UsersController = require("../controllers/usersController");

async function usersRoutes(fastify, opts) {

  // ✅ Protect ALL routes under this file

  fastify.addHook("preHandler", fastify.authenticate);

  // ✅ Route: Get all items

  fastify.get("/api/me", (req, reply) =>

    UsersController.getProfile(fastify, req, reply)

  );

}

module.exports = usersRoutes;

src/models/usersModel.js

// models/usersModel.js

// Handles DB queries for users

async function findUserById(fastify, id) {

  const conn = await fastify.mysql.getConnection();

  try {

    const [rows] = await conn.query("SELECT id, email FROM users WHERE id = ?", [id]);

    return rows[0] || null;

  } finally {

    conn.release();

  }

}

module.exports = {

  findUserById,

};

Src/controller/usersController.js

// controllers/usersController.js

const UsersModel = require("../models/usersModel");

// Get current logged-in user

async function getProfile(fastify, request, reply) {

  try {

    // `fastify.authenticate` already verified token and attached user info

    const userId = request.user.id;

    // Fetch user from DB

    const user = await UsersModel.findUserById(fastify, userId);

    if (!user) return reply.code(404).send({ message: "User not found" });

    return reply.send(user); // { id, email }

  } catch (err) {

    return reply.code(500).send({ message: "Internal server error" });

  }

}

module.exports = { getProfile };

src/routes/itemRoutes.js

// itemRoutes.js

// Defines routes for CRUD items (protected by JWT middleware with cookies)

const ItemController = require("../controllers/itemController");

async function itemRoutes(fastify, opts) {

  // ✅ Protect ALL routes under this file

  fastify.addHook("preHandler", fastify.authenticate);

  // ✅ Route: Get all items

  fastify.get("/api/items", (req, reply) =>

    ItemController.getItemsHandler(fastify, req, reply)

  );

  // ✅ Route: Create new item

  fastify.post("/api/items", (req, reply) =>

    ItemController.createItemHandler(fastify, req, reply)

  );

  // ✅ Route: Get single item by ID

  fastify.get("/api/items/:id", (req, reply) =>

    ItemController.getItemHandler(fastify, req, reply)

  );

  // ✅ Route: Update item by ID

  fastify.put("/api/items/:id", (req, reply) =>

    ItemController.updateItemHandler(fastify, req, reply)

  );

  // ✅ Route: Delete item by ID

  fastify.delete("/api/items/:id", (req, reply) =>

    ItemController.deleteItemHandler(fastify, req, reply)

  );

}

module.exports = itemRoutes;

Src/model/itemModel.js

// itemModel.js

// Handles DB queries for items table

async function createItem(fastify, item) {

  const conn = await fastify.mysql.getConnection(); // Get DB connection

  try {

    // Insert new item

    await conn.query(

      "INSERT INTO items (id, name, description) VALUES (?, ?, ?)",

      [item.id, item.name, item.description]

    );

  } finally {

    conn.release(); // Release connection

  }

}

async function getAllItems(fastify) {

  const conn = await fastify.mysql.getConnection();

  try {

    // Select all items

    const [rows] = await conn.query("SELECT \* FROM items");

    return rows;

  } finally {

    conn.release();

  }

}

async function getItemById(fastify, id) {

  const conn = await fastify.mysql.getConnection();

  try {

    // Find item by ID

    const [rows] = await conn.query("SELECT \* FROM items WHERE id = ?", [id]);

    return rows[0];

  } finally {

    conn.release();

  }

}

async function updateItem(fastify, id, data) {

  const conn = await fastify.mysql.getConnection();

  try {

    // Update item

    await conn.query("UPDATE items SET name=?, description=? WHERE id=?", [

      data.name,

      data.description,

      id,

    ]);

  } finally {

    conn.release();

  }

}

async function deleteItem(fastify, id) {

  const conn = await fastify.mysql.getConnection();

  try {

    // Delete item

    await conn.query("DELETE FROM items WHERE id=?", [id]);

  } finally {

    conn.release();

  }

}

module.exports = {

  createItem,

  getAllItems,

  getItemById,

  updateItem,

  deleteItem,

};

// Handles DB queries for items table

Src/controller/itemController.js

// itemController.js

// Handles CRUD logic for items

const { v4: uuidv4 } = require("uuid"); // Generate unique IDs

const ItemModel = require("../models/itemModel"); // Item model

async function createItemHandler(fastify, request, reply) {

  const { name, description } = request.body; // Get data from request

  if (!name) return reply.code(400).send({ message: "Name required" });

  // Generate unique item ID

  const id = uuidv4();

  // Save item in DB

  await ItemModel.createItem(fastify, { id, name, description });

  // Return new item

  return reply.code(201).send({ id, name, description });

}

async function getItemsHandler(fastify, request, reply) {

  // Get all items

  const rows = await ItemModel.getAllItems(fastify);

  return reply.send(rows);

}

async function getItemHandler(fastify, request, reply) {

  const { id } = request.params; // Get ID from URL

  const item = await ItemModel.getItemById(fastify, id);

  if (!item) return reply.code(404).send({ message: "Not found" });

  return reply.send(item);

}

async function updateItemHandler(fastify, request, reply) {

  const { id } = request.params; // Item ID

  const { name, description } = request.body; // Updated data

  // Update item

  await ItemModel.updateItem(fastify, id, { name, description });

  return reply.send({ message: "Updated" });

}

async function deleteItemHandler(fastify, request, reply) {

  const { id } = request.params; // Item ID

  // Delete item

  await ItemModel.deleteItem(fastify, id);

  return reply.send({ message: "Deleted" });

}

module.exports = {

  createItemHandler,

  getItemsHandler,

  getItemHandler,

  updateItemHandler,

  deleteItemHandler,

};

// Handles CRUD logic for items

Package.json

{

  "name": "node\_backend",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dev": "nodemon --watch src --exec node src/server.js",

    "start": "node src/server.js",

    "migrate": "node src/migrate.js"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "type": "commonjs",

  "dependencies": {

    "@fastify/cookie": "^11.0.2",

    "@fastify/cors": "^11.1.0",

    "@fastify/jwt": "^10.0.0",

    "@fastify/mysql": "^5.0.2",

    "bcryptjs": "^3.0.2",

    "fastify": "^5.6.1",

    "uuid": "^13.0.0"

  },

  "devDependencies": {

    "dotenv": "^17.2.2",

    "nodemon": "^3.1.10"

  }

}

Frontend setup  
  
axios connection from backend

// ✅ Import axios HTTP client

import axios from "axios";

// ✅ Create a reusable axios instance for your API

const axiosClient = axios.create({

  baseURL: "http://127.0.0.1:4000/api", // All API requests will start with this base URL

  withCredentials: true,                // Always send cookies (accessToken + refreshToken) automatically

});

// ✅ Setup a response interceptor → listens for ALL responses

axiosClient.interceptors.response.use(

  // 🔹 If the response is successful (status 2xx), just return it

  (res) => res,

  // 🔹 If an error happens (non-2xx response), handle it here

  async (error) => {

    const originalRequest = error.config; // Save the failed request so we can retry it later

    // ⚠️ Check: was the error due to "Unauthorized" (401)?

    // Also check: did we already retry this request? (avoid infinite loops)

    if (

      error.response &&                     // Make sure server sent a response

      error.response.status === 401 &&      // The error is "Unauthorized"

      !originalRequest.\_retry               // Haven’t retried this request yet

    ) {

      originalRequest.\_retry = true;        // Mark this request so we don’t retry twice

      try {

        // 🔄 Ask backend to refresh the access token (using refreshToken in cookies)

        await axiosClient.post("/refresh");

        // 🔁 If refresh worked → retry the original failed request

        return axiosClient(originalRequest);

      } catch (refreshError) {

        // ❌ If refresh also failed (refreshToken expired or invalid) → force logout

        window.location.href = "/login"; // Redirect user to login page

        return Promise.reject(refreshError);

      }

    }

    // ❌ If error is NOT 401, or retry already happened → reject the error

    return Promise.reject(error);

  }

);

// ✅ Export axiosClient so you can use it anywhere in your frontend

export default axiosClient;

authservice.js

// authService.js

import api from "./axios";

// ✅ Login → backend sets HttpOnly cookies

export async function login(username, password) {

  const res = await api.post("/login", { username, password });

  return res.data; // e.g., { message: "Login successful" }

}

// ✅ Logout → backend clears cookies

export async function logout() {

  const res = await api.post("/logout");

  return res.data; // e.g., { message: "Logged out" }

}

// ✅ Fetch logged-in user (protected route)

export async function getProfile() {

  const res = await api.get("/me"); // backend reads cookie

  return res.data; // e.g., { id: 1, username: "plok" }

}