JWT Authentication System Design

i have a react front end(hooks) and node/express backend with mongoDB,company wants to JWT token for authentication, what should the detailed (salt+bcrypt+keep alive time) architecture/system design to implement the same. I want a detailed process flow end to end

Here's a **detailed end-to-end architecture and system design** for implementing **JWT-based authentication** using **React (hooks)** frontend, **Node/Express** backend, and **MongoDB**, including **bcrypt hashing**, **salting**, and **token lifecycle management** (access + refresh tokens).

🔐 1. Overview

You'll be implementing:

- **Registration (Sign Up)**
- **Login (Sign In)**
- **JWT Access Token + Refresh Token**
- **Password hashing with bcrypt + salt**
- **Protected routes (auth middleware)**
- **Token refreshing endpoint**
- **Logout (invalidate refresh token)**

🧱 2. Tech Stack

- **Frontend**: React with Hooks + Context API (or Redux)
- **Backend**: Node.js + Express
- **Database**: MongoDB
- **Auth**: JWT (JSON Web Token)
- **Security**: bcrypt for password hashing, salting, HTTP-only cookies for refresh tokens

🔄 3. High-Level Flow

🔽 Register

1. User signs up via React form.
2. Frontend sends user data (email, password) to the Express backend.
3. Backend:
    - Generates salt (bcrypt)
    - Hashes password
    - Stores user with hashed password in MongoDB

🔼 Login

1. User logs in via React.
2. Frontend sends email + password.
3. Backend:
    - Finds user
    - Compares passwords using `bcrypt.compare`
    - If valid, generates:
        - **Access token** (short-lived, e.g. 15min)
        - **Refresh token** (long-lived, e.g. 7 days)
    - Sends access token to frontend (typically in response)
    - Sends refresh token as **HttpOnly cookie**

🔁 Refresh Token

1. When access token expires, frontend silently hits `/refresh` endpoint using refresh token (from HttpOnly cookie).
2. Backend verifies refresh token, issues new access token.

🔒 Protected Routes

- Include access token in `Authorization: Bearer <token>` header.
- Middleware verifies the token and grants access.

🧩 4. Components

📦 MongoDB User Schema

```js
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  passwordHash: { type: String, required: true },
  refreshToken: { type: String }, // optional: to track current refresh token
});

module.exports = mongoose.model("User", userSchema);
```

🛡️ 5. Auth Logic – Backend

🔐 Registration

```js
const bcrypt = require("bcrypt");

app.post('/register', async (req, res) => {
  const { email, password } = req.body;

  const saltRounds = 10;
  const salt = await bcrypt.genSalt(saltRounds);
  const hash = await bcrypt.hash(password, salt);

  const user = new User({ email, passwordHash: hash });
  await user.save();

  res.status(201).send({ message: "User registered" });
});
```

🔐 Login + Token Issuing

```js
const jwt = require("jsonwebtoken");

const ACCESS_TOKEN_EXPIRY = '15m';
const REFRESH_TOKEN_EXPIRY = '7d';

function generateAccessToken(userId) {
  return jwt.sign({ userId }, process.env.ACCESS_SECRET, { expiresIn:
ACCESS_TOKEN_EXPIRY });
}

function generateRefreshToken(userId) {
  return jwt.sign({ userId }, process.env.REFRESH_SECRET, { expiresIn:
REFRESH_TOKEN_EXPIRY });
}

app.post('/login', async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });

  if (!user || !(await bcrypt.compare(password, user.passwordHash))) {
    return res.status(401).send("Invalid credentials");
  }

  const accessToken = generateAccessToken(user._id);
  const refreshToken = generateRefreshToken(user._id);

  // optionally store refreshToken in DB
  user.refreshToken = refreshToken;
  await user.save();

  res.cookie('refreshToken', refreshToken, {
    httpOnly: true,
    secure: true,
    sameSite: 'strict',
    maxAge: 7 * 24 * 60 * 60 * 1000 // 7 days
  });
```

```
  res.json({ accessToken });
});
```

🔄 Refresh Token Endpoint

```js
app.post('/refresh', async (req, res) => {
  const token = req.cookies.refreshToken;
  if (!token) return res.status(401).send("No refresh token");

  try {
    const payload = jwt.verify(token, process.env.REFRESH_SECRET);
    const user = await User.findById(payload.userId);

    if (!user || user.refreshToken !== token) {
      return res.status(403).send("Invalid refresh token");
    }

    const accessToken = generateAccessToken(user._id);
    res.json({ accessToken });

  } catch (err) {
    return res.status(403).send("Token expired or invalid");
  }
});
```

🔓 Auth Middleware

```js
function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader?.split(' ')[1];
  if (!token) return res.sendStatus(401);
```

```
  jwt.verify(token, process.env.ACCESS_SECRET, (err, payload) => {
    if (err) return res.sendStatus(403);
    req.userId = payload.userId;
    next();
  });
}
```

🚪 Logout

```
app.post('/logout', async (req, res) => {
  const token = req.cookies.refreshToken;
  if (token) {
    const payload = jwt.verify(token, process.env.REFRESH_SECRET);
    const user = await User.findById(payload.userId);
    if (user) {
      user.refreshToken = null;
      await user.save();
    }
  }
  res.clearCookie('refreshToken');
  res.sendStatus(204);
});
```

🧠 6. React Frontend (Hooks)

📥 Login Form

```
const handleLogin = async () => {
  const res = await fetch('/login', {
    method: 'POST',
    credentials: 'include', // to send cookies
    headers: { 'Content-Type': 'application/json' },
```

```
    body: JSON.stringify({ email, password }),
  });
  const data = await res.json();
  setAccessToken(data.accessToken);
};
```

🔁 Refresh Access Token Automatically

Create an axios instance with an interceptor:

```js
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:5000',
  withCredentials: true,
});

api.interceptors.response.use(
  res => res,
  async err => {
    if (err.response.status === 403 && !err.config._retry) {
      err.config._retry = true;
      const res = await api.post('/refresh');
      const newToken = res.data.accessToken;
      // save token and retry request
      err.config.headers['Authorization'] = `Bearer ${newToken}`;
      return api(err.config);
    }
    return Promise.reject(err);
  }
);
```

📊 7. Token Expiry Strategy

Token Type	Expiry	Storage Method	Purpose
Access Token	15 min	React state / memory	API auth (short-lived)

Token Type	Expiry	Storage Method	Purpose
Refresh Token	7 days	HttpOnly secure cookie	Renew access token

---

🔐 8. Security Tips

- Store **refresh token** in an HttpOnly, secure cookie (not localStorage).
- Use **HTTPS** in production.
- Invalidate refresh tokens on logout by removing from DB.
- Rotate refresh tokens on use for better security (optional).
- Rate-limit login attempts.

Would you like me to generate the folder structure or full code boilerplate for this setup?