JAVASCRIPT

GENERATIVE AI

EXPRESS.JS WEB PROGRAMMING:

ADVANCE

(INTEGRATIVE-GENERATIVE AI EDITION)

STUDENT PRICE BOOK CENTER

- Session and Authentication
- Database Connectivity
- Advanced Error and Logging
  ReSTful API Design
- References

คำนำ

ในยุคที่แอปพลิเคชันเว็บมีบทบาทสำคัญต่อธุรกิจและผู้ใช้ปลายทาง การพัฒนาเว็บแอปพลิเคชันที่มีประสิทธิภาพ ปลอดภัย และปรับตัวได้อย่างรวดเร็วกลายเป็นสิ่งจำเป็น “Express.js” ซึ่งเป็นเว็บเฟรมเวิร์กยอดนิยมของ Node.js ได้กลายมาเป็นเครื่องมือหลักสำหรับนักพัฒนาในการสร้างระบบเว็บที่มีความยืดหยุ่นและสามารถปรับแต่งได้สูง ด้วยโครงสร้างที่เรียบง่ายแต่ทรงพลัง Express.js จึงเป็นทางเลือกอันดับต้น ๆ สำหรับการสร้าง API, Web Services และเว็บแอปพลิเคชันที่มีโครงสร้างซับซ้อน หนังสือ ***Express.js Web Programming: Advance*** เล่มนี้ถูกออกแบบขึ้นเพื่อต่อยอดความรู้จากระดับพื้นฐานสู่ระดับสูง ครอบคลุมหัวข้อที่สำคัญในการพัฒนาเว็บแอปพลิเคชันที่มีคุณภาพในโลกแห่งความเป็นจริง โดยเจาะลึกไปที่ 4 หัวข้อหลักซึ่งเป็นองค์ประกอบสำคัญของระบบเว็บระดับองค์กร ได้แก่ **การจัดการ Session และ Authentication**, **การเชื่อมต่อฐานข้อมูล**, **การจัดการ Error และ Logging**, และ **การออกแบบ RESTful API อย่างเป็นระบบ**

บทที่ 9: การจัดการ Session และ Authentication

บทนี้เริ่มต้นด้วยแนวคิดพื้นฐานของการจัดการ Session และกระบวนการ Authentication ซึ่งเป็นรากฐานของระบบรักษาความปลอดภัยของเว็บแอปพลิเคชัน จากนั้นได้นำเสนอการใช้งาน **express-session** สำหรับการเก็บ session ฝั่ง server อย่างมีประสิทธิภาพ และแนะนำ **Passport.js** ซึ่งเป็น middleware ยอดนิยมสำหรับจัดการ Authentication ในรูปแบบต่าง ๆ ไม่ว่าจะเป็น Local Login, OAuth, JWT หรือกลยุทธ์อื่น ๆ โดยมุ่งเน้นให้ผู้อ่านเข้าใจกลไกเบื้องหลังการยืนยันตัวตนของผู้ใช้ รวมถึงสามารถออกแบบระบบ Login/Logout ได้อย่างปลอดภัย

บทที่ 10: การเชื่อมต่อฐานข้อมูล

เพื่อให้ Express.js สามารถทำงานร่วมกับข้อมูลในระดับองค์กร บทนี้ได้ลงลึกถึงการเชื่อมต่อฐานข้อมูลทั้งแบบ NoSQL และ SQL โดยเริ่มจากการใช้งาน **Mongoose** เพื่อเชื่อมต่อกับ MongoDB อย่างมีแบบแผน พร้อมอธิบายการออกแบบ Schema และเทคนิคการ Query ข้อมูลอย่างมีประสิทธิภาพ จากนั้นจึงขยายไปสู่การเชื่อมต่อฐานข้อมูลเชิงสัมพันธ์ เช่น **MySQL** และ **PostgreSQL** พร้อมแนวทางการทำ ORM และการจัดการ Connection Pool อย่างมืออาชีพ ทั้งหมดนี้ช่วยให้ผู้อ่านสามารถเลือกฐานข้อมูลที่เหมาะสมกับระบบของตน และเชื่อมต่อกับ Express.js ได้อย่างมั่นใจ

บทที่ 11: การจัดการ Error และ Logging ขั้นสูง

ในระบบจริง การจัดการ Error และ Logging ที่ดีจะช่วยลดความเสี่ยง เพิ่มความเสถียร และอำนวยความสะดวกในการ Debug ระบบ บทนี้จึงเน้นการจัดการ Error อย่างเป็นระบบผ่าน Middleware โดยอธิบายการตั้งค่า Error Handler อย่างเหมาะสม และแนะนำการใช้ **Winston** และ **Morgan** ซึ่งเป็นเครื่องมือ Logging ยอดนิยมใน Node.js เพื่อเก็บข้อมูลเหตุการณ์ที่สำคัญ เช่น Access Logs, System Errors, หรือ Activity Logs พร้อมตัวอย่างโปรเจกต์ที่บูรณาการการจัดการ Error และ Logging อย่างครบวงจร ช่วยให้นักพัฒนาสามารถวิเคราะห์ปัญหาได้แม่นยำและรวดเร็วขึ้น

บทที่ 12: RESTful API Design

RESTful API เป็นหัวใจสำคัญของระบบสมัยใหม่ที่ต้องการการสื่อสารระหว่าง Client และ Server อย่างเป็นระบบ บทนี้นำเสนอหลักการออกแบบ REST API อย่างถูกต้อง ครอบคลุมเรื่องการตั้งชื่อ Route, การใช้ HTTP Method ที่เหมาะสม, การจัดการ Status Code, การจัดโครงสร้าง Resource และการออกแบบ Versioning สำหรับ API ที่เปลี่ยนแปลงได้ในอนาคต นอกจากนี้ยังมีตัวอย่างการเขียน **API Documentation** อย่างมืออาชีพ รวมถึงโปรเจกต์บูรณาการที่ช่วยให้ผู้อ่านสามารถนำแนวคิดไปประยุกต์ใช้ในระบบจริงได้ทันที

หนังสือเล่มนี้เหมาะสำหรับนักพัฒนาระดับกลางถึงขั้นสูงที่ต้องการยกระดับทักษะการใช้ Express.js ในการพัฒนาเว็บแอปพลิเคชันอย่างมีระบบแบบแผน โดยทุกบทมาพร้อมตัวอย่างโค้ดที่เข้าใจง่าย อธิบายกระบวนการอย่างชัดเจน และเน้นการลงมือปฏิบัติจริง เพื่อให้ผู้อ่านสามารถนำความรู้ไปใช้กับโปรเจกต์ของตนได้ทันที

หวังเป็นอย่างยิ่งว่าหนังสือ *Express.js Web Programming: Advance* เล่มนี้จะเป็นคู่มือที่มีคุณค่า และช่วยให้คุณก้าวสู่การเป็น Full-Stack Developer ที่สามารถสร้างระบบ Express.js ได้อย่างมั่นใจ แข็งแรง และพร้อมสำหรับการใช้งานจริงในระดับมืออาชีพ

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 9

การจัดการ Session และ Authentication

(Session and Authentication)

เนื้อหา

- การจัดการ Session และ Authentication
- การจัดการ Session และ Authentication (เชิงลึก)
- การใช้งาน express-session
- การใช้ Passport.js สำหรับ Authentication

บทนำบทที่ 9: การจัดการ Session และ Authentication

ในการพัฒนาเว็บแอปพลิเคชันที่มีผู้ใช้งานหลายคน การจัดการ session และระบบยืนยันตัวตน (authentication) ถือเป็นองค์ประกอบสำคัญที่ช่วยให้ระบบสามารถจดจำผู้ใช้ และรักษาความปลอดภัยของข้อมูลได้อย่างเหมาะสม บทที่ 9 นี้จึงมุ่งเน้นการเรียนรู้วิธีจัดการ session และการสร้างระบบล็อกอินอย่างมีประสิทธิภาพด้วยเครื่องมือยอดนิยมใน Express.js

หัวข้อแรกของบทนี้จะเริ่มจากการใช้งาน **express-session** ซึ่งเป็น middleware สำหรับจัดการ session บนฝั่งเซิร์ฟเวอร์ โดย session จะทำหน้าที่เก็บข้อมูลผู้ใช้ชั่วคราวหลังจากเข้าสู่ระบบ เช่น user ID, role หรือค่าการตั้งค่าต่าง ๆ ผู้อ่านจะได้เรียนรู้การตั้งค่า session อย่างปลอดภัย รวมถึงการใช้ session ร่วมกับ cookie เพื่อเชื่อมโยงข้อมูลกับผู้ใช้งานแต่ละราย

ต่อมาจะเป็นการนำ session ไปประยุกต์ใช้ใน **ระบบล็อกอินพื้นฐาน** โดยบทนี้จะสอนวิธีสร้างหน้า login และตรวจสอบรหัสผ่าน พร้อมตัวอย่างการจัดเก็บ session เมื่อผู้ใช้เข้าสู่ระบบสำเร็จ และการป้องกันหน้าเว็บที่ต้องมีการเข้าสู่ระบบก่อนถึงจะเข้าถึงได้ (protected route) ซึ่งเป็นพื้นฐานของระบบสมาชิกในเว็บแอปพลิเคชันสมัยใหม่

หลังจากเข้าใจพื้นฐานแล้ว บทนี้จะต่อยอดไปสู่การใช้ **Passport.js** ซึ่งเป็นไลบรารียอดนิยมสำหรับจัดการ authentication แบบ modular โดยรองรับกลยุทธ์การยืนยันตัวตนหลากหลายรูปแบบ เช่น local login, OAuth, หรือ JWT ผู้อ่านจะได้เรียนรู้การติดตั้งและตั้งค่า Passport.js พร้อมใช้งานร่วมกับ express-session เพื่อสร้างระบบล็อกอินที่ยืดหยุ่นและปลอดภัยมากยิ่งขึ้น

เมื่อจบบทนี้ ผู้อ่านจะมีความเข้าใจทั้งในเชิงเทคนิคและแนวคิดเกี่ยวกับการจัดการ session และ authentication ใน Express.js และสามารถนำไปพัฒนาระบบผู้ใช้งานที่ปลอดภัยและมีประสิทธิภาพ พร้อมสำหรับการขยายต่อในโปรเจกต์ขนาดใหญ่หรือระบบ production จริง.

การจัดการ Session และ Authentication

---

1. การใช้งาน express-session

ความหมายและหน้าที่

- express-session เป็น middleware ที่ช่วยให้แอป Express สามารถเก็บข้อมูล session ของผู้ใช้แต่ละคนได้ระหว่างการใช้งาน
- Session คือการเก็บข้อมูลสถานะของผู้ใช้บนเซิร์ฟเวอร์ เช่น ข้อมูลการล็อกอิน ข้อมูลการตั้งค่าเฉพาะตัวผู้ใช้ เป็นต้น
- โดยปกติ HTTP เป็น protocol แบบ stateless คือไม่มีการเก็บสถานะของ client กับ server แต่ session จะช่วยแก้ปัญหานี้ได้

การติดตั้งและใช้งานเบื้องต้น

- ติดตั้งด้วยคำสั่ง:
- npm install express-session
- ตั้งค่าใน Express app เช่น
- const session = require('express-session');
-
- app.use(session({
- secret: 'your-secret-key', // รหัสลับสำหรับเซสชัน
- resave: false, // ไม่บันทึกเซสชันถ้าไม่มีการเปลี่ยนแปลง
- saveUninitialized: true, // เซสชันที่ไม่ได้ใช้ก็เก็บไว้ได้
- cookie: { secure: false } // ตั้งค่า cookie สำหรับ session (secure: true ใช้เฉพาะ https)
- }));

การใช้งาน session

- สามารถเก็บข้อมูลใน session ผ่าน req.session
- req.session.user = { id: 1, username: 'admin' };
- ดึงข้อมูลได้ตลอดระยะเวลาการเชื่อมต่อ session นั้น ๆ

---

2. การสร้างระบบล็อกอินง่าย ๆ

แนวทางการทำงาน

- มีหน้า login form เพื่อรับ username/password
- เมื่อผู้ใช้กรอกข้อมูลและส่ง form ระบบจะตรวจสอบข้อมูลกับฐานข้อมูลหรือข้อมูลที่กำหนดไว้
- หากถูกต้อง จะบันทึกสถานะผู้ใช้ลงใน session
- ใช้ session ในการตรวจสอบสิทธิ์การเข้าถึงหน้าอื่น ๆ

ตัวอย่างขั้นพื้นฐาน

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  // ตัวอย่างตรวจสอบ username/password แบบง่าย
  if (username === 'admin' && password === '1234') {
    req.session.user = { username: 'admin' };
    res.redirect('/dashboard');
  } else {
    res.send('Invalid credentials');
  }
});

app.get('/dashboard', (req, res) => {
  if (req.session.user) {
    res.send(`Welcome ${req.session.user.username}`);
  } else {
    res.redirect('/login');
  }
});
```

ฟีเจอร์เพิ่มเติม

- การออกจากระบบ (logout) ลบ session
- การตั้งเวลา session หมดอายุ
- ป้องกัน session fixation

---

3. การใช้ Passport.js สำหรับ Authentication

Passport.js คืออะไร?

- Passport.js เป็น middleware สำหรับจัดการ Authentication ที่มีความยืดหยุ่นและขยายตัวได้ดี
- รองรับกลยุทธ์ (strategies) หลากหลาย เช่น local (username/password), OAuth (Facebook, Google), JWT เป็นต้น
- ช่วยแยกส่วนของ Authentication ออกจาก business logic ได้ง่าย

การติดตั้ง

```
npm install passport passport-local express-session
```

การตั้งค่า Passport แบบ local strategy เบื้องต้น

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

app.use(session({ secret: 'secret' }));
app.use(passport.initialize());
app.use(passport.session());

// กำหนด local strategy
passport.use(new LocalStrategy((username, password, done) => {
  // ตรวจสอบ username/password จากฐานข้อมูล
  if (username === 'admin' && password === '1234') {
    return done(null, { id: 1, username: 'admin' });
  } else {
    return done(null, false, { message: 'Incorrect credentials.' });
  }
}));

// จัดการ serialize session
passport.serializeUser((user, done) => {
  done(null, user.id);
});

passport.deserializeUser((id, done) => {
  // ดึงข้อมูลผู้ใช้จากฐานข้อมูล
  done(null, { id: 1, username: 'admin' });
});

// Route login
app.post('/login', passport.authenticate('local', {
  successRedirect: '/dashboard',
  failureRedirect: '/login',
  failureFlash: false
}));
```

```
// Route ป้องกันสำหรับหน้าที่ต้อง login
app.get('/dashboard', (req, res) => {
  if (req.isAuthenticated()) {
    res.send(`Hello ${req.user.username}`);
  } else {
    res.redirect('/login');
  }
});
```

ข้อดีของ Passport

- รองรับกลยุทธ์ login หลากหลาย
- มี plugin และ community เยอะ
- ง่ายต่อการจัดการ session ร่วมกับ Express

การจัดการ Session และ Authentication (เชิงลึก)

---

1. การใช้งาน express-session

แนวคิดและการทำงานของ Session ในเว็บ

- **HTTP protocol เป็น stateless** — หมายความว่าแต่ละคำขอ HTTP (request) จะถูกประมวลผลแยกจากกัน ไม่มีการเก็บสถานะระหว่างคำขอ
- **Session คือกลไกเก็บสถานะของผู้ใช้** ที่ฝั่งเซิร์ฟเวอร์ เพื่อให้ทราบว่า request ใดมาจากผู้ใช้คนเดียวกัน โดยเชื่อมโยงผ่าน session ID ที่ส่งใน cookie
- Session ID จะถูกสร้างขึ้นตอนแรกเมื่อ client เชื่อมต่อและเก็บไว้ใน cookie เพื่อส่งกับทุก request ถัดไป

express-session ทำงานอย่างไร

- เมื่อติดตั้งและใช้ middleware express-session ในแอป จะมีการสร้างและจัดการ session ID สำหรับผู้ใช้แต่ละคน
- Session data จะถูกเก็บไว้ใน server memory (default) หรือ external store เช่น Redis, MongoDB (แนะนำใน production เพื่อประสิทธิภาพและความปลอดภัย)
- Client จะได้รับ cookie ที่มี session ID (ชื่อ default คือ connect.sid) เพื่อใช้ส่งกับ request ถัดไป

การตั้งค่า express-session

```
app.use(session({
  secret: 'your-secret-key',          // รหัสลับสำหรับเข้ารหัส session ID cookie
  resave: false,                      // ถ้า session ไม่เปลี่ยนแปลง จะไม่เซฟซ้ำ (ช่วยลด overhead)
```

```
  saveUninitialized: false,          // ไม่สร้าง session ใหม่ถ้ายังไม่ได้ใช้ (ช่วยลดจำนวน session ว่าง)
  cookie: {
    secure: false,                   // ต้องเป็น true ถ้าใช้ https (ป้องกัน cookie ถูกส่งผ่าน HTTP ธรรมดา)
    maxAge: 1000 * 60 * 60 * 2       // อายุ cookie 2 ชั่วโมง (หน่วยเป็นมิลลิวินาที)
  }
}));
```

- **secret:** ต้องตั้งให้มั่นคงและปลอดภัย เพื่อป้องกันการปลอมแปลง session
- **resave:** ควรตั้งเป็น false เพื่อไม่ให้เซฟ session ทุกครั้งถ้าไม่มีการเปลี่ยนแปลง
- **saveUninitialized:** ตั้ง false เพื่อไม่สร้าง session ถ้าผู้ใช้ไม่ได้ล็อกอิน หรือไม่ได้เก็บข้อมูลอะไร

ตัวอย่างการใช้งาน session ใน route

```
app.get('/set', (req, res) => {
  req.session.user = { id: 123, username: 'testuser' };
  res.send('Session data set');
});

app.get('/get', (req, res) => {
  if (req.session.user) {
    res.send(`Hello, ${req.session.user.username}`);
  } else {
    res.send('No session data');
  }
});
```

การเก็บ session ใน production

- **ไม่แนะนำให้เก็บ session ในหน่วยความจำ server ตรง ๆ** เพราะจะสูญหายเมื่อ restart server หรือเมื่อ scaling
- ควรใช้ store เช่น connect-redis, connect-mongo เพื่อเก็บ session ในฐานข้อมูลหรือ cache ที่ persistent และแชร์ข้าม server ได้

---

2. การสร้างระบบล็อกอินง่าย ๆ ด้วย session

ขั้นตอนหลักของระบบล็อกอิน

1. สร้างฟอร์ม login รับข้อมูล username/password จากผู้ใช้
2. เมื่อผู้ใช้ส่งข้อมูลเข้ามา ตรวจสอบ username/password กับฐานข้อมูลหรือข้อมูลในระบบ

3. หากตรวจสอบผ่าน ให้สร้าง session เพื่อเก็บสถานะผู้ใช้ (เช่น userId, username)
4. ใช้ session ในทุก route ที่ต้องการตรวจสอบการล็อกอิน
5. มี route สำหรับ logout เพื่อล้าง session

ตัวอย่างระบบล็อกอินง่าย ๆ

```
// middleware เพื่ออ่าน body form data
app.use(express.urlencoded({ extended: true }));

// หน้า login form
app.get('/login', (req, res) => {
  res.send(`
    <form method="POST" action="/login">
      <input name="username" required />
      <input name="password" type="password" required />
      <button>Login</button>
    </form>
  `);
});

// ตรวจสอบข้อมูลล็อกอิน
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  // ตรวจสอบกับฐานข้อมูลจริงหรือ hardcode (ตัวอย่าง)
  if (username === 'admin' && password === '1234') {
    req.session.user = { username: 'admin' };
    res.redirect('/dashboard');
  } else {
    res.send('Invalid credentials');
  }
});

// หน้า dashboard ที่ต้องล็อกอินก่อนเข้า
app.get('/dashboard', (req, res) => {
  if (!req.session.user) {
```

```
    return res.redirect('/login');
  }
  res.send(`Welcome, ${req.session.user.username}! <a href="/logout">Logout</a>`);
});

// Logout - ลบ session
app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) {
      return res.send('Error logging out');
    }
    res.redirect('/login');
  });
});
```

ฟีเจอร์และการพัฒนาเพิ่มเติม

- ตั้งเวลา session หมดอายุ (timeout)
- ป้องกัน Session Fixation (เช่น regenerate session id หลัง login)
- ใช้ HTTPS เพื่อป้องกัน cookie session ถูกขโมย
- การเข้ารหัส password ด้วย bcrypt ก่อนเก็บในฐานข้อมูล
- การใช้ CSRF token เพื่อป้องกันการโจมตี cross-site request forgery

---

3. การใช้ Passport.js สำหรับ Authentication

เหตุผลที่ควรใช้ Passport.js

- Passport.js เป็น middleware สำหรับจัดการ Authentication โดยเฉพาะ
- มีระบบกลยุทธ์ (strategies) หลากหลาย ครอบคลุมทั้งแบบ local (username/password), OAuth, OpenID, JWT ฯลฯ
- ช่วยแยกส่วนโค้ด authentication ออกจาก business logic
- มี community และ plugin เยอะ ทำให้ขยายระบบง่าย
- จัดการ session ให้แบบอัตโนมัติร่วมกับ express-session

แนวคิดหลักของ Passport.js

- **Strategies** คือโมดูลที่รับผิดชอบวิธีตรวจสอบตัวตน เช่น passport-local
- **serializeUser** และ **deserializeUser** ใช้เก็บและดึงข้อมูลผู้ใช้เข้าสู่ session
- Middleware passport.initialize() และ passport.session() เพื่อเชื่อมกับ Express

ตัวอย่างการใช้งาน Passport.js (Local Strategy)

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

app.use(session({ secret: 'secret', resave: false, saveUninitialized: false }));
app.use(passport.initialize());
app.use(passport.session());

// กำหนด local strategy
passport.use(new LocalStrategy((username, password, done) => {
  // ตัวอย่าง: ตรวจสอบ username/password แบบง่าย
  if (username === 'admin' && password === '1234') {
    return done(null, { id: 1, username: 'admin' });
  } else {
    return done(null, false, { message: 'Incorrect credentials.' });
  }
}));

// บันทึกข้อมูลผู้ใช้ใน session
passport.serializeUser((user, done) => {
  done(null, user.id);
});

// ดึงข้อมูลผู้ใช้จาก session
passport.deserializeUser((id, done) => {
  // ในระบบจริงจะดึงข้อมูล user จากฐานข้อมูล
  done(null, { id: 1, username: 'admin' });
});

// Route login ใช้งาน passport.authenticate
app.post('/login', passport.authenticate('local', {
  successRedirect: '/dashboard',
  failureRedirect: '/login'
}));
```

```
// Route dashboard ตรวจสอบว่า logged in หรือไม่
function ensureAuthenticated(req, res, next) {
  if (req.isAuthenticated()) return next();
  res.redirect('/login');
}

app.get('/dashboard', ensureAuthenticated, (req, res) => {
  res.send(`Hello, ${req.user.username} <a href="/logout">Logout</a>`);
});

// Logout
app.get('/logout', (req, res) => {
  req.logout(() => {
    res.redirect('/login');
  });
});
```

การขยายระบบด้วยกลยุทธ์อื่น

- OAuth (Google, Facebook, Twitter)
- JWT (สำหรับ API)
- LDAP, SAML สำหรับองค์กร

ข้อควรระวังและแนวปฏิบัติ

- ตั้งค่า session secret ให้ปลอดภัยและไม่เปิดเผย
- ใช้ HTTPS เสมอเพื่อปกป้อง cookie session
- ไม่เก็บข้อมูลสำคัญ เช่น password ใน session โดยตรง
- ใช้ bcrypt หรือ library ที่ปลอดภัยสำหรับ hash รหัสผ่าน
- ตรวจสอบ input และป้องกัน injection

การใช้งาน express-session

---

1. express-session คืออะไร?

- เป็น **middleware** สำหรับจัดการ session ใน Express
- ช่วยให้เก็บข้อมูลสถานะผู้ใช้ (session data) ไว้ที่ฝั่ง server และเชื่อมโยงกับ client ผ่าน session ID ใน cookie

- ใช้สำหรับเก็บข้อมูลสำคัญ เช่น การล็อกอิน, ตะกร้าสินค้า, ค่าที่ใช้ในระหว่าง session ของผู้ใช้แต่ละคน

---

2. การติดตั้ง express-session

npm install express-session

---

3. วิธีใช้งานเบื้องต้น

```
const express = require('express');
const session = require('express-session');

const app = express();

app.use(session({
  secret: 'your-secret-key',  // รหัสลับ ใช้สำหรับเซ็น session ID cookie (ต้องตั้งให้ปลอดภัย)
  resave: false,              // ไม่เซฟ session ใหม่ถ้าไม่มีการเปลี่ยนแปลงข้อมูลใน session
  saveUninitialized: false,   // ไม่เซฟ session ถ้า session ยังไม่ได้ถูกแก้ไข (ช่วยลดจำนวน session เปล่าๆ)
  cookie: {
    secure: false,            // ต้องเป็น true ถ้าใช้ HTTPS (ป้องกัน cookie ถูกส่งใน HTTP ธรรมดา)
    maxAge: 1000 * 60 * 60    // อายุ cookie 1 ชั่วโมง (หน่วยมิลลิวินาที)
  }
}));

app.get('/', (req, res) => {
  // เช็คว่ามี session อยู่ไหม ถ้าไม่มี สร้างใหม่
  if (!req.session.views) {
    req.session.views = 0;
  }
  req.session.views++;

  res.send(`คุณเข้ามาดูหน้านี้ ${req.session.views} ครั้ง`);
});

app.listen(3000, () => {
```

```
  console.log('Server running on http://localhost:3000');
});
```

---

4. ค่าคอนฟิกสำคัญใน express-session

ตัวเลือก	รายละเอียด	คำแนะนำ
secret	รหัสลับที่ใช้เซ็น session ID cookie เพื่อป้องกันการปลอมแปลง ต้องตั้งให้ปลอดภัยและเป็นความลับ	ตั้งรหัสที่ซับซ้อนและเก็บให้ดี
resave	ถ้า false จะไม่เซฟ session ถ้าไม่มีการเปลี่ยนแปลง (ลดภาระ)	ควรตั้งเป็น false
saveUninitialized	ถ้า false จะไม่เซฟ session ที่ยังไม่ได้ถูกแก้ไข เช่น session เปล่า ๆ	ควรตั้งเป็น false
cookie.secure	ถ้าตั้งเป็น true จะส่ง cookie เฉพาะ HTTPS เท่านั้น	สำหรับ production ให้ตั้ง true
cookie.maxAge	กำหนดอายุ cookie (หน่วย ms) เช่น 1 ชั่วโมง = 1000*60*60	ตั้งตามความต้องการ

---

5. การใช้งาน session ใน route ต่าง ๆ

- เขียนข้อมูลลงใน session

```
req.session.username = 'admin';
```

- อ่านข้อมูลจาก session

```
const username = req.session.username;
```

- ลบข้อมูล session หรือทำลาย session

```
req.session.destroy(err => {
  if (err) console.error(err);
  else console.log('Session destroyed');
});
```

---

6. การเก็บ session ใน production

- โดยดีฟอลต์ express-session จะเก็บ session ใน **หน่วยความจำของ server** (MemoryStore) ซึ่งไม่เหมาะกับ production
- ควรใช้ external store เช่น
    - Redis: connect-redis
    - MongoDB: connect-mongo
    - MySQL: express-mysql-session

เพื่อให้ session คงอยู่ได้แม้ server รีสตาร์ท และรองรับระบบแบบหลาย instance

---

7. ตัวอย่างใช้งาน Redis เป็น store

```
npm install connect-redis redis
const session = require('express-session');
const RedisStore = require('connect-redis')(session);
const redis = require('redis');

const redisClient = redis.createClient();

app.use(session({
  store: new RedisStore({ client: redisClient }),
  secret: 'your-secret',
  resave: false,
  saveUninitialized: false,
  cookie: { secure: false, maxAge: 3600000 }
}));
```

---

8. ข้อควรระวัง

- อย่าใช้ secret ที่เดาง่าย หรือเปิดเผยสู่สาธารณะ
- อย่าลืมตั้งค่า cookie.secure เป็น true เมื่อใช้ HTTPS
- อย่าเก็บข้อมูลสำคัญหรือความลับลงใน session โดยตรง ควรเก็บแค่ id หรือตัวชี้วัด
- ระวังการโจมตี session fixation, session hijacking โดยตั้ง session ใหม่หลังล็อกอินสำเร็จ

นี่คือตัวอย่างโปรแกรม Express.js 3 ตัวอย่างพื้นฐาน + 3 ตัวอย่างแนวประยุกต์ ที่ใช้ **express-session** พร้อมโครงสร้างไฟล์, คำอธิบายโค้ด และผลการรัน

ตัวอย่างโปรแกรมพื้นฐาน 3 ตัวอย่าง

ตัวอย่าง 1: นับจำนวนครั้งที่เข้าเว็บ (Session Counter)

โครงสร้างโปรเจกต์

```
session-basic-1/
  └── app.js
  └── package.json
```

app.js

```
const express = require('express');
const session = require('express-session');

const app = express();

app.use(session({
  secret: 'mySecretKey123',
  resave: false,
  saveUninitialized: false,
  cookie: { maxAge: 60000 }
}));

app.get('/', (req, res) => {
  if (!req.session.views) {
    req.session.views = 0;
  }
  req.session.views++;
  res.send(`คุณเข้าหน้านี้มาแล้ว ${req.session.views} ครั้ง`);
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

วิธีรันและผลการรัน

```
npm init -y
npm install express express-session
node app.js
```

เปิดเบราว์เซอร์ไปที่ http://localhost:3000

ผลลัพธ์จะแสดงข้อความ "คุณเข้าหน้านี้มาแล้ว x ครั้ง" โดยนับเพิ่มทุกครั้งที่รีเฟรช (สำหรับ session เดียวกัน)

ตัวอย่าง 2: เก็บข้อมูลผู้ใช้ลงใน session

โครงสร้างโปรเจกต์

```
session-basic-2/
  └── app.js
  └── package.json
```

app.js

```
const express = require('express');
const session = require('express-session');

const app = express();

app.use(express.urlencoded({ extended: true }));

app.use(session({
  secret: 'mySecretKey123',
  resave: false,
  saveUninitialized: false,
  cookie: { maxAge: 60000 }
}));

app.get('/', (req, res) => {
  if (req.session.username) {
    res.send(`สวัสดี, ${req.session.username}! <a href="/logout">ออกจากระบบ</a>`);
  } else {
    res.send(`
      <form method="POST" action="/login">
        ชื่อผู้ใช้: <input name="username" />
        <button type="submit">เข้าสู่ระบบ</button>
      </form>
    `);
  }
});

app.post('/login', (req, res) => {
  const { username } = req.body;
  if (username) {
```

```
    req.session.username = username;
    res.redirect('/');
  } else {
    res.send('กรุณากรอกชื่อผู้ใช้');
  }
});

app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) return res.send('เกิดข้อผิดพลาด');
    res.redirect('/');
  });
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

วิธีรันและผลการรัน

- รัน node app.js
- เข้า http://localhost:3000
- กรอกชื่อผู้ใช้ในฟอร์ม
- หน้าเว็บจะแสดงข้อความต้อนรับและลิงก์ logout
- logout จะลบ session และกลับไปหน้า login

ตัวอย่าง 3: กำหนด session timeout (หมดอายุ session)

โครงสร้างโปรเจกต์

```
session-basic-3/
  └── app.js
  └── package.json
```

app.js

```
const express = require('express');
const session = require('express-session');

const app = express();
```

```
app.use(session({
  secret: 'mySecretKey123',
  resave: false,
  saveUninitialized: false,
  cookie: { maxAge: 10000 } // หมดอายุ session หลัง 10 วินาที
}));

app.get('/', (req, res) => {
  if (!req.session.views) {
    req.session.views = 1;
  } else {
    req.session.views++;
  }
  res.send(`คุณเข้าหน้านี้มาแล้ว ${req.session.views} ครั้ง (session จะหมดอายุใน 10 วินาที)`);
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

วิธีรันและผลการรัน

- เปิดเว็บและรีเฟรชหน้าหลายครั้ง จะเห็นนับ views เพิ่มขึ้น
- ปล่อยทิ้งไว้เกิน 10 วินาทีแล้วรีเฟรช จะเริ่มนับใหม่เพราะ session หมดอายุ

ตัวอย่างโปรแกรมแนวประยุกต์ 3 ตัวอย่าง

ตัวอย่าง 1: ระบบล็อกอินด้วย username/password ง่าย ๆ เก็บสถานะด้วย session

โครงสร้างโปรเจกต์

```
session-app-1/
  └── app.js
  └── package.json
```

app.js

```
const express = require('express');
const session = require('express-session');
```

```
const app = express();

app.use(express.urlencoded({ extended: true }));

app.use(session({
  secret: 'mySecretKey123',
  resave: false,
  saveUninitialized: false,
  cookie: { maxAge: 600000 }
}));

// ตัวอย่างฐานข้อมูลผู้ใช้แบบง่าย
const users = {
  admin: '1234',
  user1: 'abcd'
};

app.get('/login', (req, res) => {
  res.send(`
    <form method="POST" action="/login">
      ชื่อผู้ใช้: <input name="username" />
      รหัสผ่าน: <input type="password" name="password" />
      <button type="submit">เข้าสู่ระบบ</button>
    </form>
  `);
});

app.post('/login', (req, res) => {
  const { username, password } = req.body;
  if (users[username] && users[username] === password) {
    req.session.user = username;
    res.redirect('/dashboard');
  } else {
```

```
    res.send('ชื่อผู้ใช้หรือรหัสผ่านไม่ถูกต้อง');
  }
});

function requireLogin(req, res, next) {
  if (!req.session.user) {
    return res.redirect('/login');
  }
  next();
}

app.get('/dashboard', requireLogin, (req, res) => {
  res.send(`ยินดีต้อนรับ ${req.session.user} <a href="/logout">ออกจากระบบ</a>`);
});

app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) return res.send('เกิดข้อผิดพลาด');
    res.redirect('/login');
  });
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

วิธีรันและผลการรัน

- เปิดเว็บ http://localhost:3000/login
- ล็อกอินด้วย username และ password จาก users
- เข้า dashboard ได้ถ้าล็อกอินถูกต้อง
- ออกจากระบบได้ด้วย logout

ตัวอย่าง 2: นับจำนวนผู้เข้าชมแยกตาม session และแสดง session ID

โครงสร้างโปรเจกต์

session-app-2/

```
└── app.js
└── package.json
```

app.js

```
const express = require('express');
const session = require('express-session');

const app = express();

app.use(session({
  secret: 'mySecretKey123',
  resave: false,
  saveUninitialized: true,
  cookie: { maxAge: 600000 }
}));

app.get('/', (req, res) => {
  if (!req.session.views) {
    req.session.views = 0;
  }
  req.session.views++;
  res.send(`
    Session ID: ${req.sessionID} <br>
    คุณเข้าหน้านี้ ${req.session.views} ครั้งใน session นี้
  `);
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

วิธีรันและผลการรัน

- เปิดเว็บและสังเกต session ID จะไม่เปลี่ยนแปลงจนกว่าจะหมดอายุหรือ cookie ถูกลบ
- จำนวนครั้งที่เข้าจะนับแยกตาม session ของแต่ละผู้ใช้

ตัวอย่าง 3: ระบบเก็บตะกร้าสินค้า (Shopping Cart) ด้วย session

โครงสร้างโปรเจกต์

```
session-app-3/
  └── app.js
  └── package.json
```

app.js

```
const express = require('express');
const session = require('express-session');

const app = express();

app.use(express.urlencoded({ extended: true }));

app.use(session({
  secret: 'mySecretKey123',
  resave: false,
  saveUninitialized: true,
  cookie: { maxAge: 600000 }
}));

// สินค้าตัวอย่าง
const products = {
  1: { name: 'Laptop', price: 25000 },
  2: { name: 'Smartphone', price: 15000 },
  3: { name: 'Tablet', price: 12000 }
};

app.get('/', (req, res) => {
  let productList = '';
  for (const id in products) {
    productList += `
      <li>${products[id].name} - ${products[id].price} บาท
      <form method="POST" action="/add-to-cart" style="display:inline;">
        <input type="hidden" name="productId" value="${id}" />
        <button type="submit">เพิ่มลงตะกร้า</button>
```

```
      </form>
      </li>
    `;
  }

  const cart = req.session.cart || {};
  let cartList = '<ul>';
  let total = 0;
  for (const id in cart) {
    const qty = cart[id];
    const product = products[id];
    cartList += `<li>${product.name} x ${qty} = ${product.price * qty} บาท</li>`;
    total += product.price * qty;
  }
  cartList += '</ul>';

  res.send(`
    <h1>สินค้า</h1>
    <ul>${productList}</ul>
    <h2>ตะกร้าสินค้า</h2>
    ${cartList}
    <p>ราคารวม: ${total} บาท</p>
  `);
});

app.post('/add-to-cart', (req, res) => {
  const { productId } = req.body;
  if (!req.session.cart) {
    req.session.cart = {};
  }
  if (!req.session.cart[productId]) {
    req.session.cart[productId] = 0;
  }
  req.session.cart[productId]++;
```