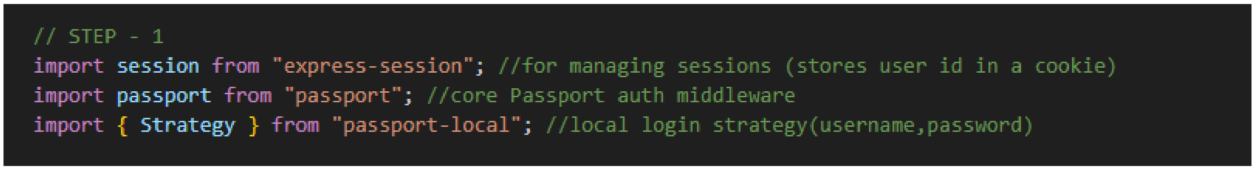
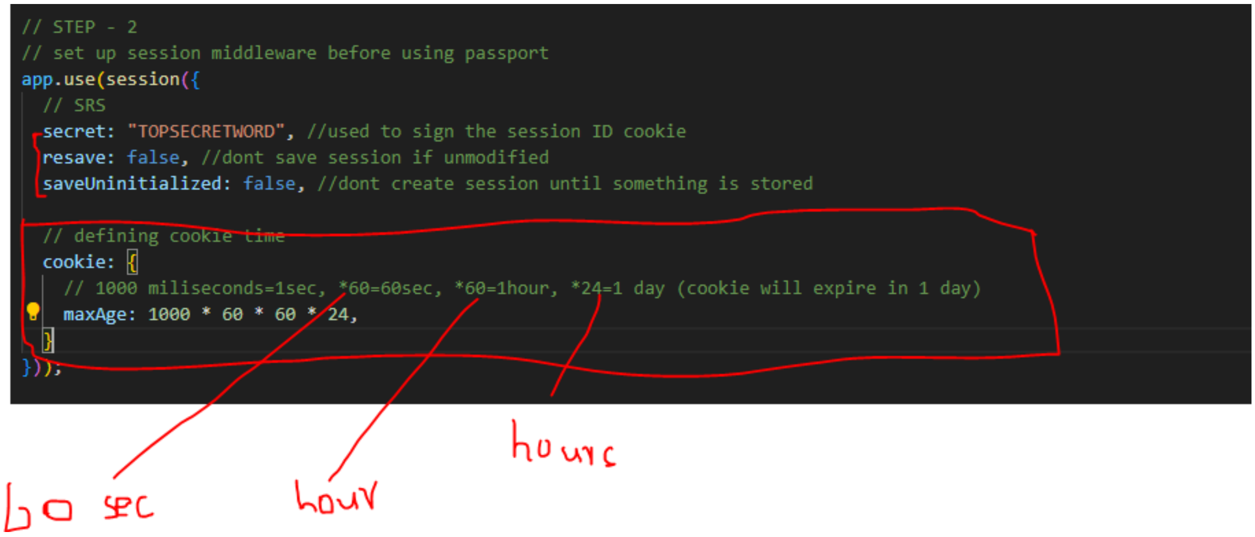
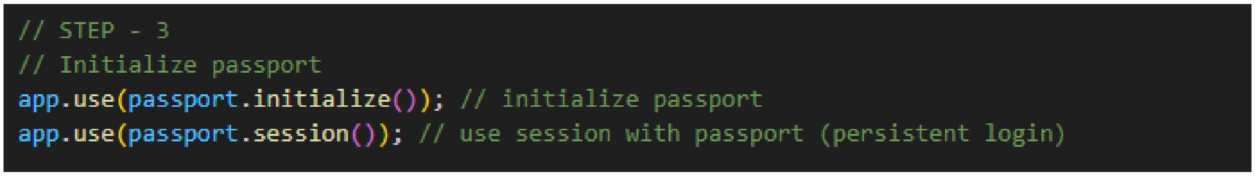
Login and Session and implementing Session and Cookie



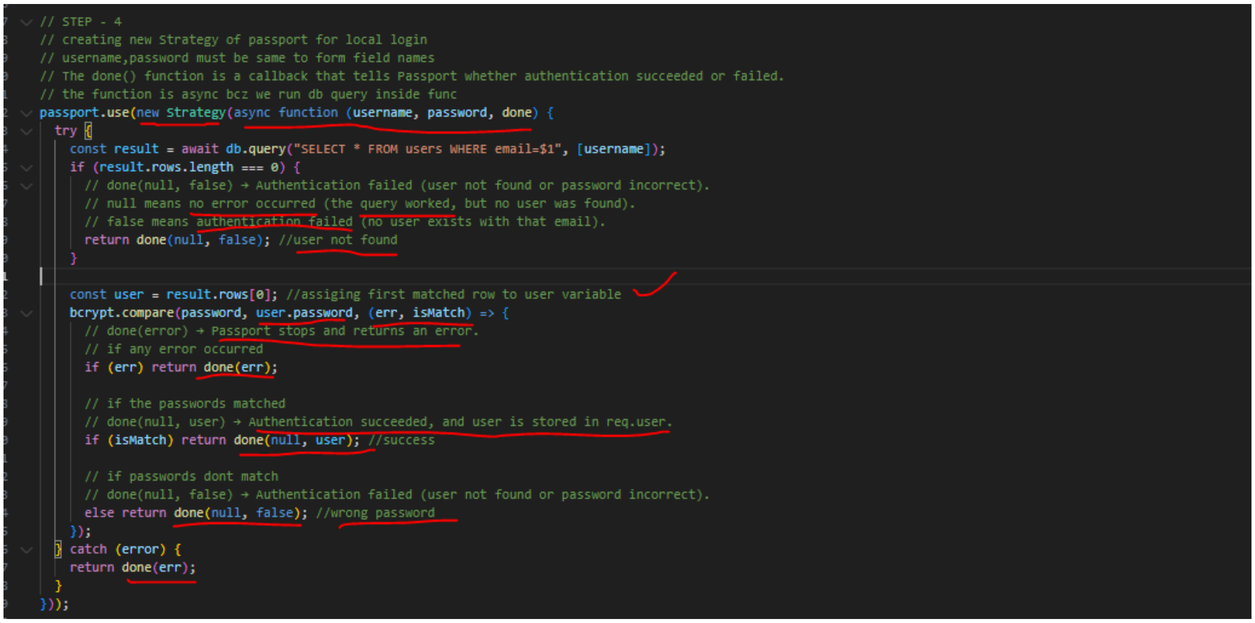
```
// STEP - 1
import session from "express-session"; //for managing sessions (stores user id in a cookie)
import passport from "passport"; //core Passport auth middleware
import { Strategy } from "passport-local"; //local login strategy(username,password)
```



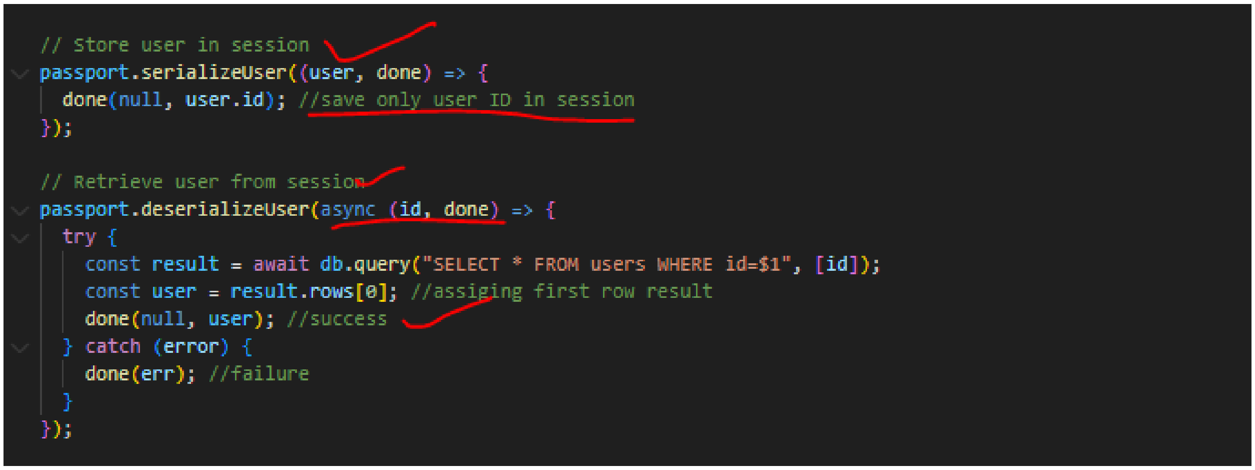
```
// STEP - 2
// set up session middleware before using passport
app.use(session({
  // SRS
  secret: "TOPSECRETWORD", //used to sign the session ID cookie
  resave: false, //dont save session if unmodified
  saveUninitialized: false, //dont create session until something is stored

  // defining cookie time
  cookie: {
    // 1000 miliseconds=1sec, *60=60sec, *60=1hour, *24=1 day (cookie will expire in 1 day)
    maxAge: 1000 * 60 * 60 * 24,
  }
}));
```



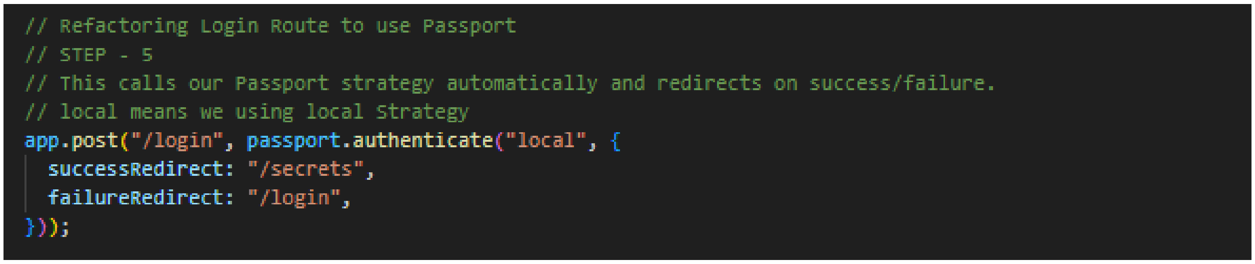
```
// STEP - 3
// Initialize passport
app.use(passport.initialize()); // initialize passport
app.use(passport.session()); // use session with passport (persistent login)
```



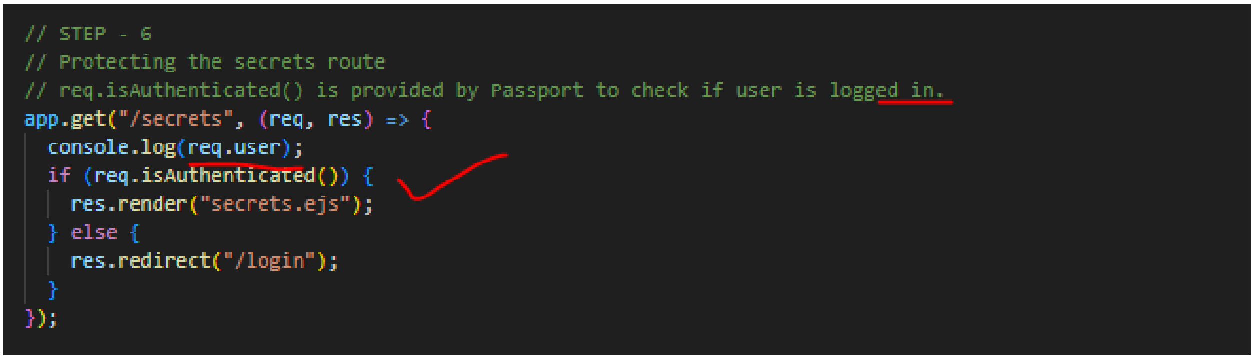
```
// STEP - 4
// creating new Strategy of passport for local login
// username,password must be same to form field names
// The done() function is a callback that tells Passport whether authentication succeeded or failed.
// the function is async bcz we run db query inside func
passport.use(new Strategy(async function (username, password, done) {
  try {
    const result = await db.query("SELECT * FROM users WHERE email=$1", [username]);
    if (result.rows.length === 0) {
      // done(null, false) → Authentication failed (user not found or password incorrect).
      // null means no error occurred (the query worked, but no user was found).
      // false means authentication failed (no user exists with that email).
      return done(null, false); //user not found
    }

    const user = result.rows[0]; //assiging first matched row to user variable
    bcrypt.compare(password, user.password, (err, isMatch) => {
      // done(error) → Passport stops and returns an error.
      // if any error occurred
      if (err) return done(err);

      // if the passwords matched
      // done(null, user) → Authentication succeeded, and user is stored in req.user.
      if (isMatch) return done(null, user); //success

      // if passwords dont match
      // done(null, false) → Authentication failed (user not found or password incorrect).
      else return done(null, false); //wrong password
    });
  } catch (error) {
    return done(err);
  }
}));
```



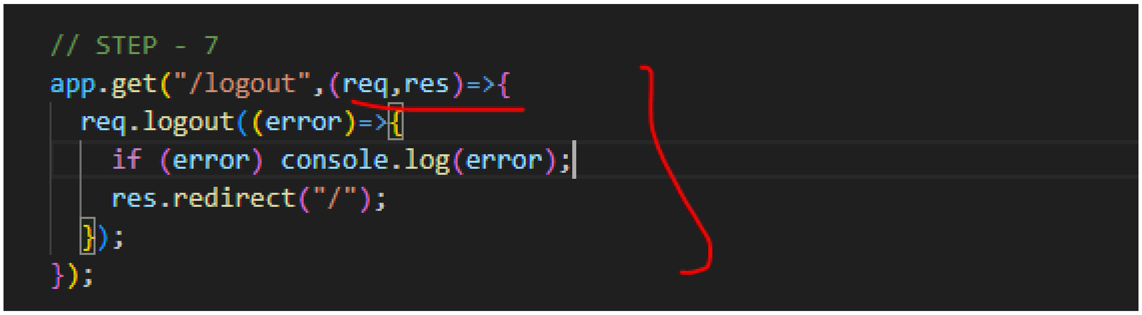
```
// Store user in session
passport.serializeUser((user, done) => {
  done(null, user.id); //save only user ID in session
});

// Retrieve user from session
passport.deserializeUser(async (id, done) => {
  try {
    const result = await db.query("SELECT * FROM users WHERE id=$1", [id]);
    const user = result.rows[0]; //assiging first row result
    done(null, user); //success
  } catch (error) {
    done(err); //failure
  }
});
```



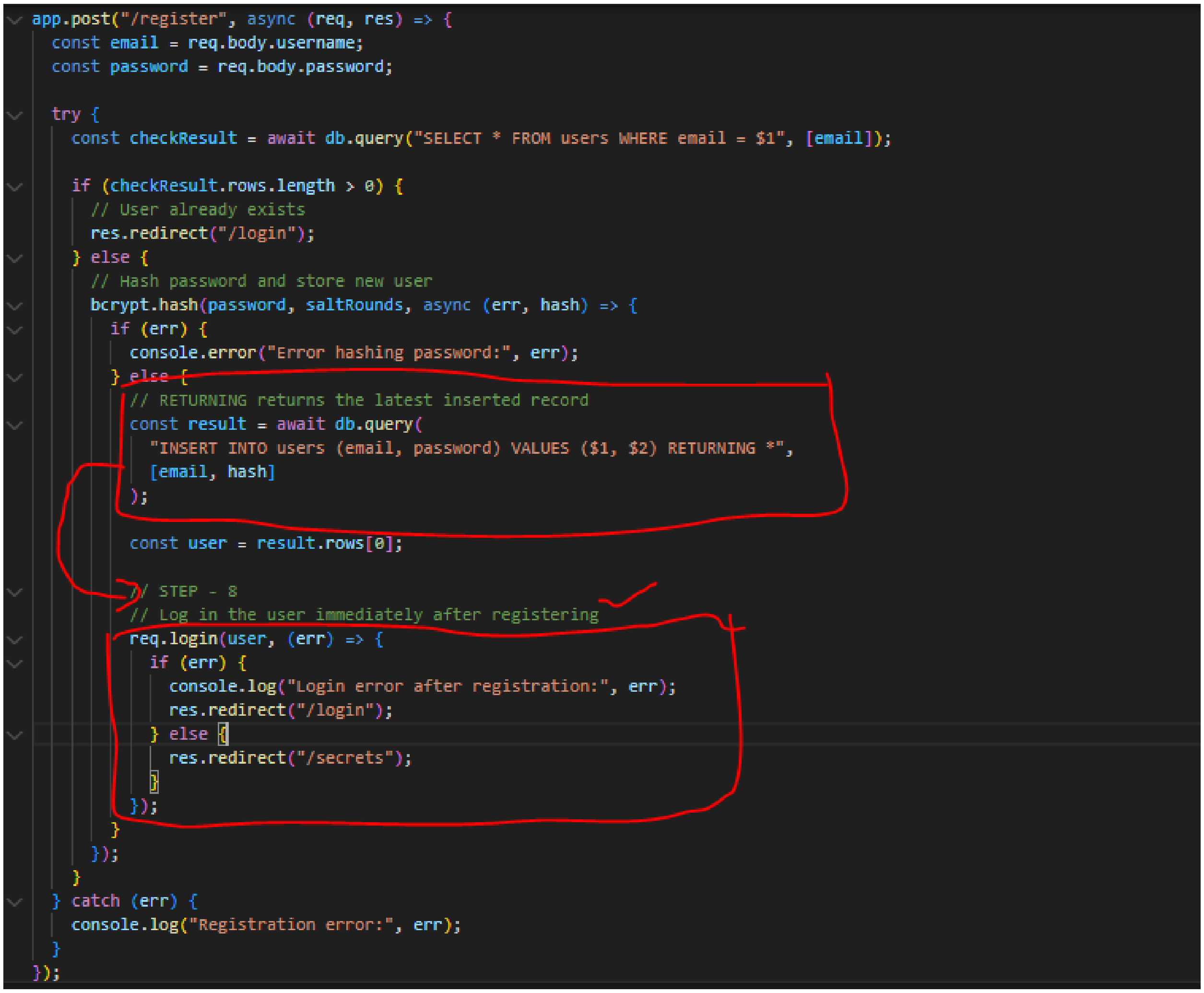
```
// Refactoring Login Route to use Passport
// STEP - 5
// This calls our Passport strategy automatically and redirects on success/failure.
// local means we using local Strategy
app.post("/login", passport.authenticate("local", {
  successRedirect: "/secrets",
  failureRedirect: "/login",
}));
```



```
// STEP - 6
// Protecting the secrets route
// req.isAuthenticated() is provided by Passport to check if user is logged in.
app.get("/secrets", (req, res) => {
  console.log(req.user);
  if (req.isAuthenticated()) {
    res.render("secrets.ejs");
  } else {
    res.redirect("/login");
  }
});
```



```
// STEP - 7
app.get("/logout",(req,res)=>{
  req.logout((error)=>{
    if (error) console.log(error);
    res.redirect("/");
  });
});
```



```
app.post("/register", async (req, res) => {
  const email = req.body.username;
  const password = req.body.password;

  try {
    const checkResult = await db.query("SELECT * FROM users WHERE email = $1", [email]);

    if (checkResult.rows.length > 0) {
      // User already exists
      res.redirect("/login");
    } else {
      // Hash password and store new user
      bcrypt.hash(password, saltRounds, async (err, hash) => {
        if (err) {
          console.error("Error hashing password:", err);
        } else {
          // RETURNING returns the latest inserted record
          const result = await db.query(
            "INSERT INTO users (email, password) VALUES ($1, $2) RETURNING *",
            [email, hash]
          );

          const user = result.rows[0];

          // STEP - 8
          // Log in the user immediately after registering
          req.login(user, (err) => {
            if (err) {
              console.log("Login error after registration:", err);
              res.redirect("/login");
            } else {
              res.redirect("/secrets");
            }
          });
        }
      });
    }
  } catch (err) {
    console.log("Registration error:", err);
  }
});
```