

Nachdem mit dem vorigen Arbeitsblatt klar geworden ist, wie Cookies funktionieren, können wir mit diesem Mechanismus Sessions implementieren. Doch was ist eine Session?

Erinnere dich, dass HTTP ein stateless Protokoll ist und zwischen den einzelnen Requests keine Erinnerung an vorhergehende Aktionen bleibt. Eine Session speichert die Daten, auf die du über Anfragen hinweg zugreifen möchtest. Jeder Nutzer, der deine Webseite besucht, hat eine eigene Session. Du kannst Sessions verwenden, um Benutzerdaten zu speichern und auf sie zuzugreifen, während sie deine Anwendung verwenden.

Dabei hast du die Wahl: Speichern der Daten am Client, oder am Server. **In jedem Fall wirst du aber zumindest am Client eine Userid speichern müssen.** Dazu gibt es mehrere Möglichkeiten. Die bei Weitem populärste sind Cookies. Aber es gibt auch:

Lokaler Speicher:

- **Local Storage** (Die Objekte (Schlüssel-Wert-Paare) werden dauerhaft gespeichert und bleiben bestehen, bis sie vom Benutzer oder der Website entfernt werden. Ein Objekt kann bis zu 5 MB groß sein, was einen erheblichen Vorteil gegenüber Cookies darstellt.)
- **Session Storage** (ähnlich wie Local Storage, die gespeicherten Objekte sind nur für das aktuelle Browser-Fenster verfügbar und werden gelöscht, wenn das Fenster geschlossen wird)
- **IndexedDB** (lokale NoSQL Datenbank)

Des Weiteren gibt es auch zahlreiche Tricks wie die Einbettung von UserIds in gecachten Dokumenten, Verwendung von Response Header (ETag, Last-Modified), und einige mehr.

Wir wollen uns aber hier auf Cookies konzentrieren, welche die Userid speichern. Die Daten der Session selbst wollen wir am Server speichern.

1. Starte mit dem Code von **Express Sessions - Vue CLI - START.zip**. Mache dich mit dem Code vertraut.

Dieser Server bietet bereits folgende Routen (die aber noch nicht implementiert sind):

HTTP Verb	Pfad	Bedeutung
GET	/	Anzeige der Willkommen-Seite via /public
POST	/register	Übermitteln der Daten zum Registrieren
POST	/login	Übermitteln der Daten zum Login
GET	/secretdata	Anwendungsspezifische Daten
GET	/logout	Löschen der aktuellen Session und des Cookies

Der Client bietet bereits die Views und folgende Routen:

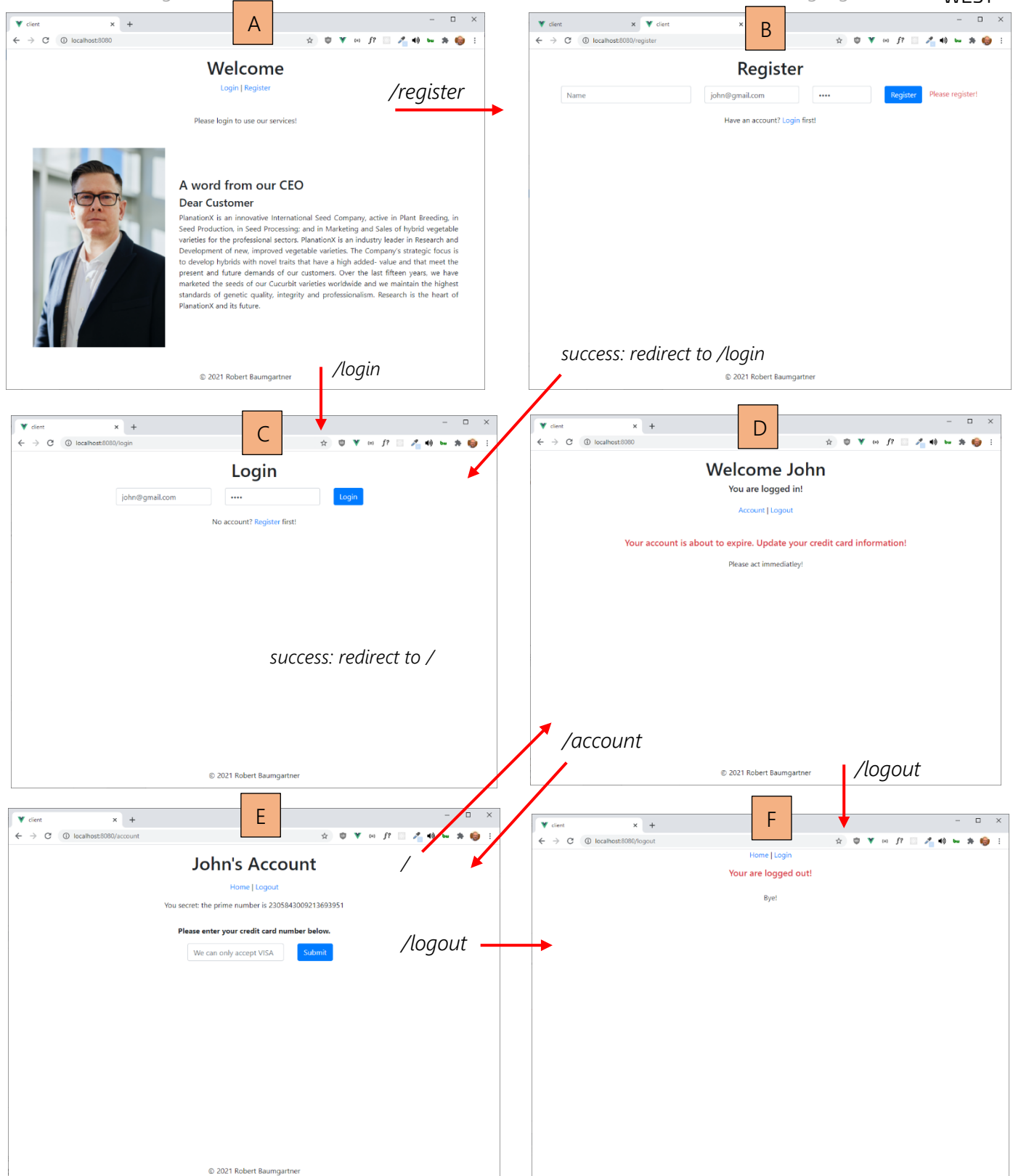
Pfad	Bedeutung
/	Anzeige der Willkommen-Seite
/register	Formular zur Registrierung des Users
/login	Anzeigen des Formulars zum Einloggen des Users
/logout	Bestätigung, dass der User ausgeloggt ist
/account	Anwendungsspezifische Daten

In der Startversion ist die Willkommen-Seite immer die Gleiche. Schön wäre es, dass der User, wenn er eingeloggt ist, persönlich mit Namen begrüßt würde. Die Seite, die über **/secretdata** aufgerufen wird, sollte überhaupt nur eingeloggten Usern zur Verfügung stehen.

Wir gehen davon aus, dass der Vue Client als kompakte Version in **/public** sich befindet und durch Vue Router am Client geroutet wird.

Robert Baumgartner

SEW 4. Jahrgang



Der User wird, wenn er nicht bereits eingeloggt ist, mit einem neutralen Welcome (A) begrüßt. Nun kann er sich entweder registrieren (B) oder einloggen (C). Ist er dann erfolgreich eingeloggt, wird er wieder auf die Home Route (D) redirected. Nun aber wird er persönlich begrüßt und eventuelle Nachrichten werden angezeigt. Von der Home Route kann er persönliche Daten anzeigen (E), von E wieder auf die Home Seite wechseln (D) oder ausloggen. In dem Fall kommt er wieder auf die neutrale Welcome Seite (F).

2. Installiere das npm Modul **express-sessions**.

express-session ist eine Middleware, welche Tätigkeiten in Bezug auf die Session für uns erledigt, d. h. das Erstellen der Session, das Setzen des Session-Cookies und das Erstellen des Session-Objektes im `req`-Objekt. Die Middleware verwendet Cookies um eine Session-ID am Client zu speichern. Der Browser sendet die Session-ID danach mit jedem Request mit. Auch die Speicherung der aktuellen Sessions übernimmt **express-session**. Diese Middleware speichert die Daten am Server.

In diesem Arbeitsblatt verwenden wir den Defaultort zum Speichern von Sitzungen, d. h. **MemoryStore**. Natürlich sind dann alle Sessions weg, wenn wir den Server restarten oder wenn er crashed! Außerdem aus der express Doku:

***Warning** The default server-side session storage, MemoryStore, is **purposely** not designed for a production environment. It will leak memory under most conditions, does not scale past a single process, and is meant for debugging and developing.¹*

Daher verwende niemals den MemoryStore in der Produktion.

Wir werden im nächsten Arbeitsblatt PostgreSQL dazu verwenden.

Alternative: **cookie-session**. Diese Middleware speichert die Daten im Cookie am Client (Bsp: Shopping Cart).

3. Um die **express-session** Middleware zu registrieren, verwende wieder `app.use`. Bei dieser Gelegenheit kann **express-session** konfiguriert werden.

Wiederum macht es Sinn, Werte im `.env` File abzuspeichern.

Hier ein Beispiel:

```
PORT=3000
SESSION_LIFETIME=2
SESSION_NAME=sid
SESSION_SECRET=I love SEW and INSY
NODE_ENV=development
```

und diese dann als Konstante in `app.js` zu verwenden.

```
const { PORT, NODE_ENV, SESSION_LIFETIME, SESSION_NAME, SESSION_SECRET } = process.env;
```

Registrierung der Middleware:

```
app.use(
  session({
    secret: SESSION_SECRET,
    name: SESSION_NAME,
    saveUninitialized: false,
    resave: false,
    cookie: {
      maxAge: SESSION_LIFETIME * 1000 * 60 * 60,
      httpOnly: false,
      sameSite: true,
      secure: NODE_ENV === 'production',
    },
  }),
);
```

SESSION_SECRET: Ein String, der von **express-session** für das Signieren des Cookies verwendet wird. Im Endeffekt wird ein Hashwert aus dem Secret und den Daten des Cookies berechnet.

¹ <http://expressjs.com/en/resources/middleware/session.html>

SESSION_NAME: Name für das Session Cookie. Der Standardwert ist `connect.sid`.

saveUnitialized: Ob erzwungen werden soll, dass nicht initialisierte (neue, aber nicht geänderte) Sessions in den Speicher übertragen werden. Der Standardwert ist `true` (veraltet). Für Login Sessions macht es keinen Sinn, leere Sitzungen für nicht authentifizierte Anfragen zu speichern, da sie noch nicht mit wertvollen Daten verknüpft sind und somit Speicherplatz verschwenden. Wir speichern die neue Sitzung erst, wenn sich der Benutzer angemeldet hat.

resave: Ob erzwungen werden soll, dass nicht geänderte Sessions in den Speicher zurückgeschrieben werden sollen. Standard ist `true` (veraltet). Wir speichern nur geänderte Sessions!

Soweit unsere Session Parameter. Die anderen Parameter sind Cookie Parameter. Siehe voriges Arbeitsblatt.

Unter **/model/users.js** sind bereits einige User eingetragen. Diese können wir für den Login Prozess verwenden. Wenn der User das Login Formular ausfüllt (C) und auf den Login Button drückt, soll er eingeloggt werden und eine personalisierte Home Page (D) angezeigt werden.

Dazu ist eine **POST /login** Route nötig, welche die übermittelten Daten mit den registrierten Usern vergleicht.

4. Erstelle die **POST /login** Route.

Suche zunächst den User in dem Array aus **/model/users.js**.

Ist er vorhanden, dann setze die `userId` in dem Session Objekt, dass von **express-session** automatisch als Property vom `req` Objekt erzeugt wird!

```
if (email && password) {
  const user = users.find(el => el.email === email &&
                                el.password === password);

  if (user) {
    req.session.userId = user.id;
    res.status(200).json({ id: user.id, name: user.name });
  } else res.status(401).send('Wrong email or password');
} else res.status(400).send('Login failed');
```

In **Login.vue** nimm `id` und `name` entgegen und speichere sie in `localStorage`.

Redirecte auf **Home.vue**. In **Home.vue** hole dir das Objekt wieder aus `localStorage`!

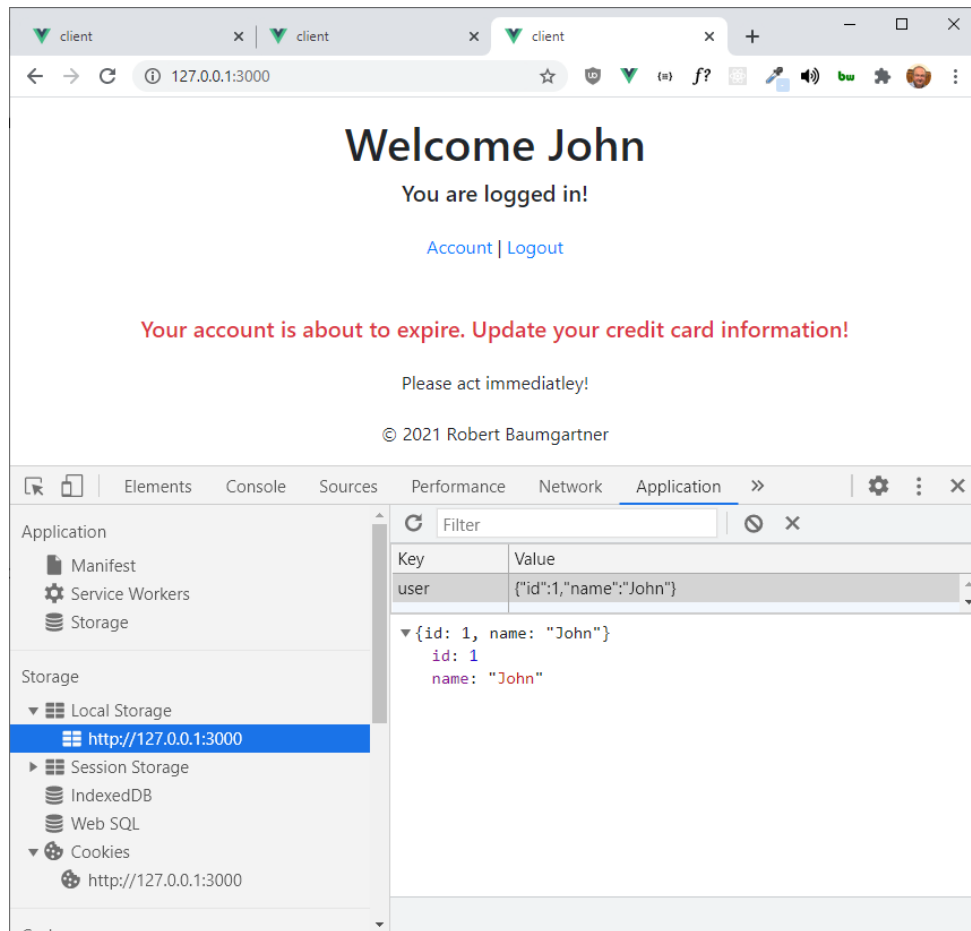
Um jedoch den Development Server verwenden zu können und relative Routen wie **/login etc.** am Client zu haben, stellen wir einen Proxy Server ein.

Lege dazu im Root Verzeichnis die Datei **vue.config.js** an!

```
const path = require('path');

module.exports = {
  outputDir: path.resolve(__dirname, '../server/public'),
  devServer: {
    proxy: 'http://127.0.0.1:3000',
  },
};
```

Überprüfe, ob alles gesetzt ist! Erinnere dich: Das Cookie wird automatisch gesetzt und ist das Zeichen, dass der User eingeloggt ist.



5. Erstelle eine **GET /logout** Route.

```
req.session.destroy();  
res.clearCookie(process.env.SESSION_NAME);
```

In **Logout.vue** rufe diese Route am Server auf und lösche danach **localStorage**.

6. Erstelle eine **POST /register** Route.

Überprüfe ob **email**, **password** und **name** ausgefüllt wurden und suche nach der E-Mail Adresse in den Daten am Server.

Wenn sie nicht vorhanden ist, dann vergib die nächste **id** und speichere die Daten ab. Sende **200 OK** an den Client. Im anderen Fall sende **409 'E-Mail already registered'**. Fehlen Daten, sende **400 'Registration failed'**.

7. Erstelle eine **GET /secretdata** Route

```
return res.status(200).end('the prime number is 2305843009213693951');
```


Verwende diese Route in **Account.vue** um persönliche Daten (in diesem Fall die Primzahl) anzuzeigen.

Nun müssen die Routen abgesichert werden. Dazu verwenden wir am Server eine Hilfsfunktion:

```
const redirectLogin = (req, res, next) => {  
  if (!req.session.userId) res.status(400).send('You are not logged in!');  
  else next();  
};
```

Diese prüft, ob die **userId** im Session Objekt gesetzt ist (dann ist der User eingeloggt).

8. Verwende diese Funktion als Middleware um die Routen abzusichern:




```
router.get('/secretdata', redirectLogin, (req, res) => {  
  return res.status(200).end('the prime number is 2305843009213693951');  
});
```

Jetzt die Client Routen. Dazu reicht es festzustellen, ob ein Cookie vorhanden ist!

```
{  
  path: '/account',  
  name: 'Account',  
  component: Account,  
  beforeEnter: (to, from, next) => {  
    if (!isAuthenticated()) next({ name: 'Login' });  
    next();  
  },  
},
```

```
function isAuthenticated() {  
  if (Vue.$cookies.get('sid')) return true;  
  else return false;  
}
```



9. Um auf die Cookies in Vue Components zuzugreifen, installiere **vue-cookies**.

In **main.js** importiere sie:

```
import VueCookies from 'vue-cookies';  
Vue.use(VueCookies);
```

Ein Letztes bleibt noch zu tun: Damit der History Mode im Vue Router funktioniert, müssen Routen, die irrtümlich zum Server geschickt wurden, dort abgefangen werden.

10. Installiere am Server in **app.js** das Modul **connect-history-api-fallback**. Verwende es folgendermaßen:

```
app.use(express.static(path.join(__dirname, '/public')));  
app.use(history());  
app.use(express.static(path.join(__dirname, '/public')));
```

Fertig. Im nächsten Arbeitsblatt geben wir den Session Store von Memory in die PostgreSQL Datenbank!