Software Engineering

Session 9

Today

Lab

- 'post' routes review
- Passwords and authentication
  - Hashing and salting
  - Sessions and cookies

Seminar

- Sprint 3 reviews

'Post' routes

NOTE: see lab 7.2 - adding data to the database

'Post' routes review...

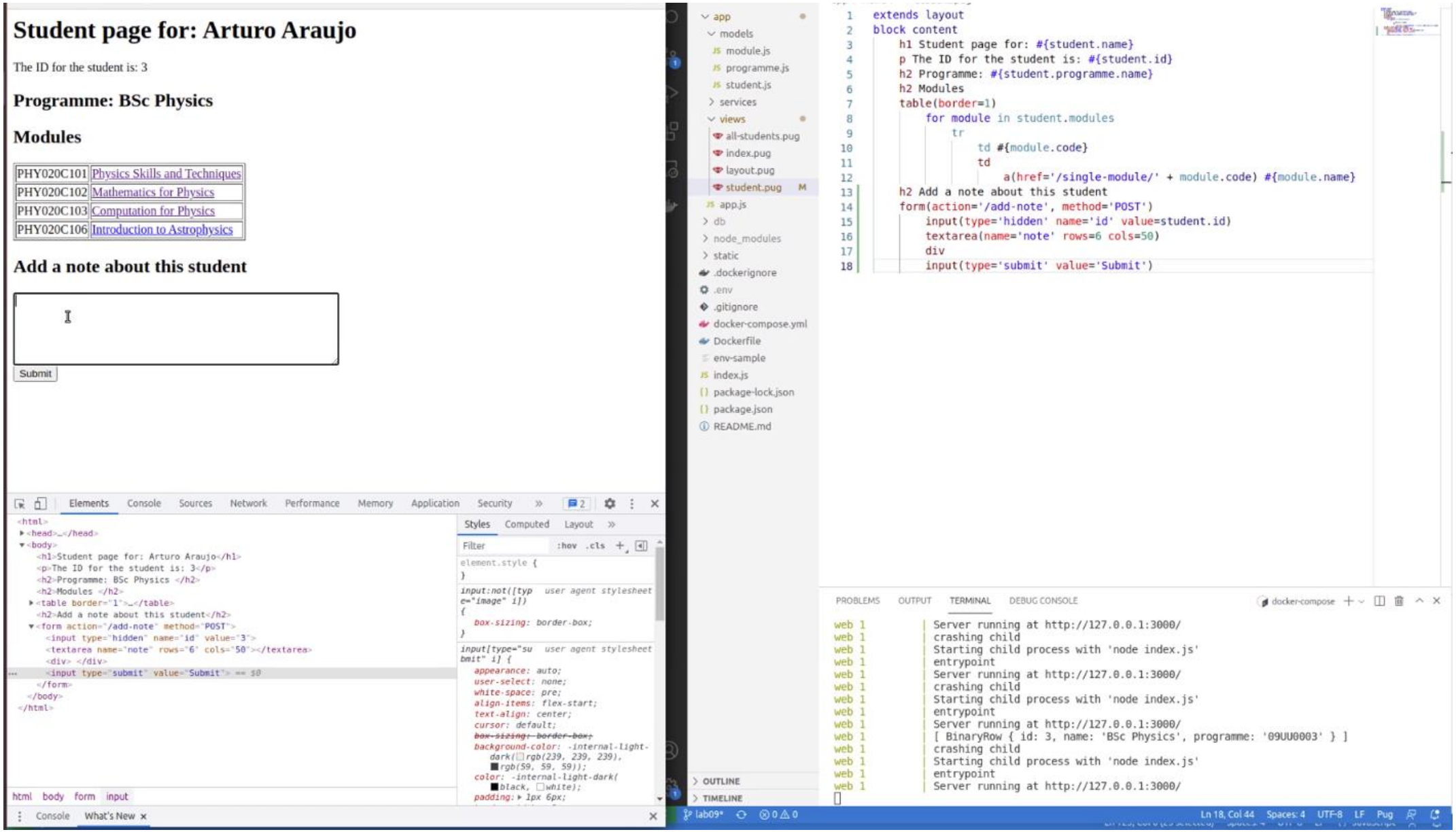
User story

As an admin I'd like to add a note to a students page with a summary of their progress so that lecturers can see an overview of the student at a glance

Task list (from the user story)

- Add a column to the student table to store the note
- Add a textarea form field to the student template for the user to enter the note
- Create a route in app.js so that the user input can be captured in the backend
- Add to our backend code so that:
    - The note can be stored in the students table
    - The Student model can be updated with the new data
- Add to the student template so that the new field can be displayed to the user

Task 1: add a 'note' for a student(1): PUG form



app.js

```
94
95  app.post('/add-note', async function (req, res) {
96      params = req.body;
97      var student = new Student(params.id)
98      // Adding a try/catch block which will be useful later when we add to the database
99      try {
100         await student.addStudentNote(params.note);
101         res.redirect('/single-student/' + params.id);
102     } catch (err) {
103         console.error(`Error while adding note `, err.message);
104     }
105 });
106
```

Student.js

```
52
53      async addStudentNote(note) {
54          var sql = "UPDATE Students SET note = ? WHERE Students.id = ?"
55          const result = await db.query(sql, [note, this.id]);
56          // Ensure the note property in the model is up to date
57          this.note = note;
58          return result;
59      }
60
```

Passwords and authentication

What is authentication?

What is authentication?

1. Capturing user information
2. Storing passwords securely
3. Comparing stored password with submitted password
4. Setting a 'session' variable per user so that each page load can be checked to see if the viewer is authenticated.

Verifying a users identity with each page load, so that information personal to them can be shown

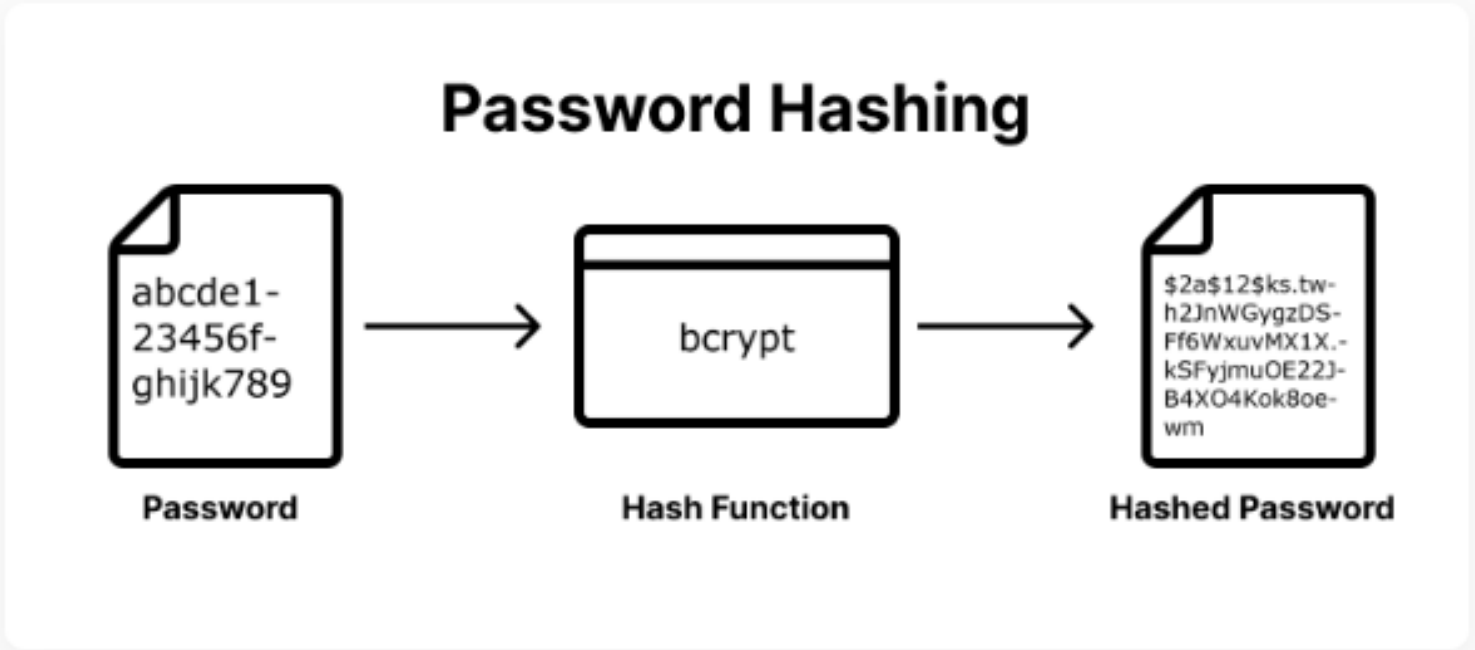
What do we need to know?

1. How to capture information submitted by the user
2. How passwords can be stored securely but still be compared to user input
3. How we can set and retrieve 'sessions' for individual users so that they don't have to log in for every page and we can send them personalised content

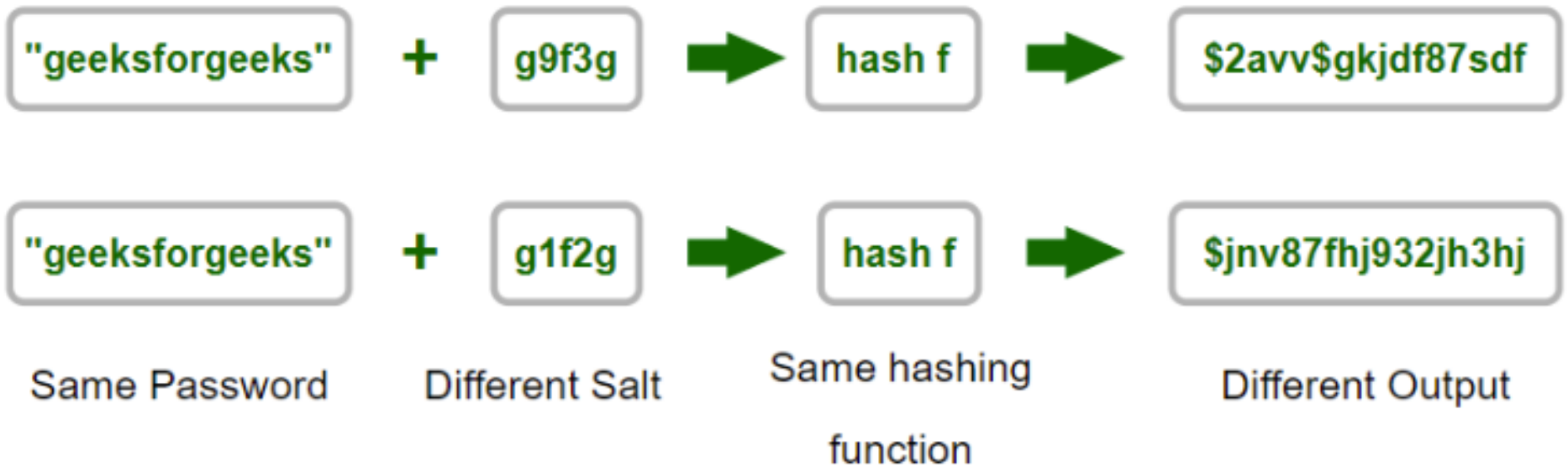
Key concepts

- Passwords must be stored in 'hashed' format ie. A one-way encrypted format.
- This means that an entered password can be compared to one that is stored, but it is impossible to work out the exact content of the stored password.
- 'salting' is essential otherwise passwords can be recovered via 'rainbow tables.'
- *Always use up to date libraries and frameworks for authentication! They will ensure that everything is working as it should. - we use the bcryptjs library.*

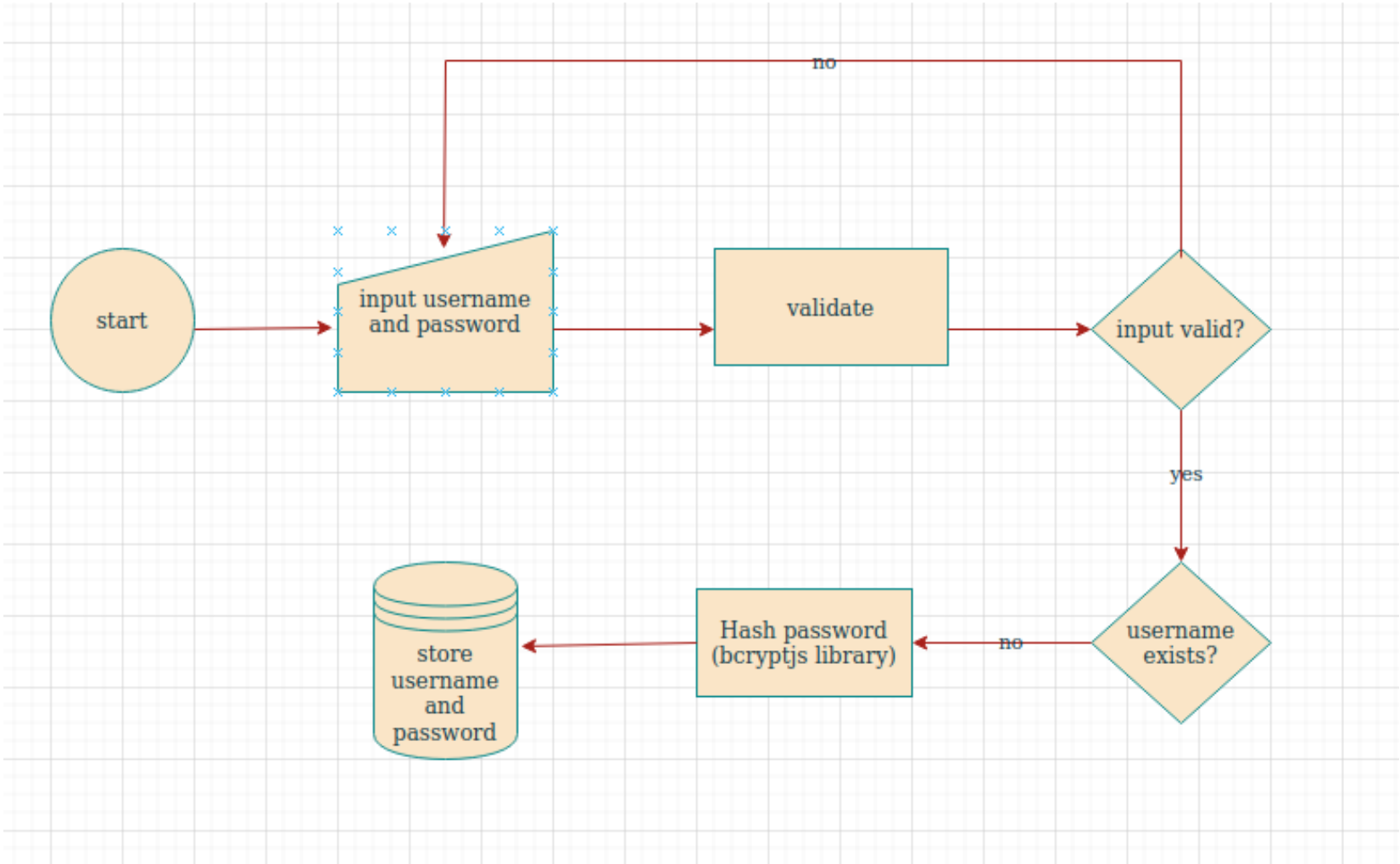
Password hashing



Password hashing with 'salt'



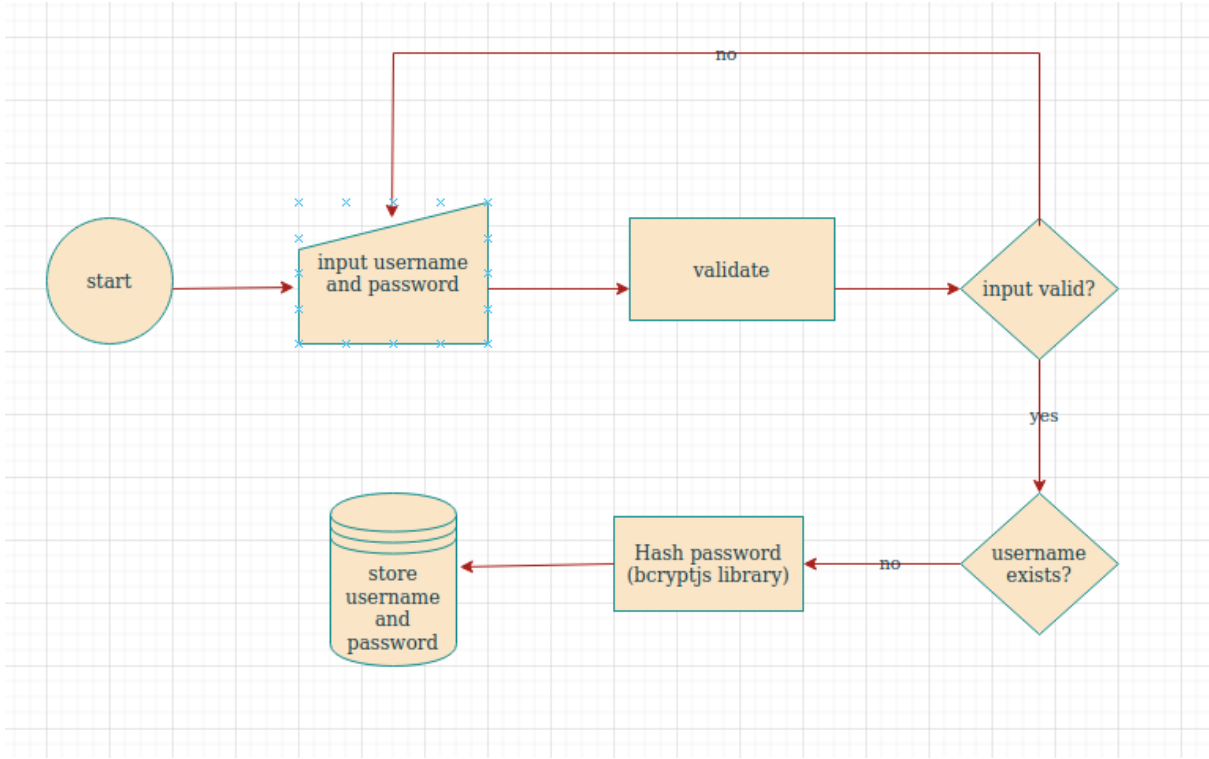
Set password: flowchart



Set password: express/pug

- VIEW: Pug template – password input form
- CONTROLLER: app.js: receive input, check and send to model
- MODEL: User.js (model), encrypt (bcryptjs.js library) and save to database

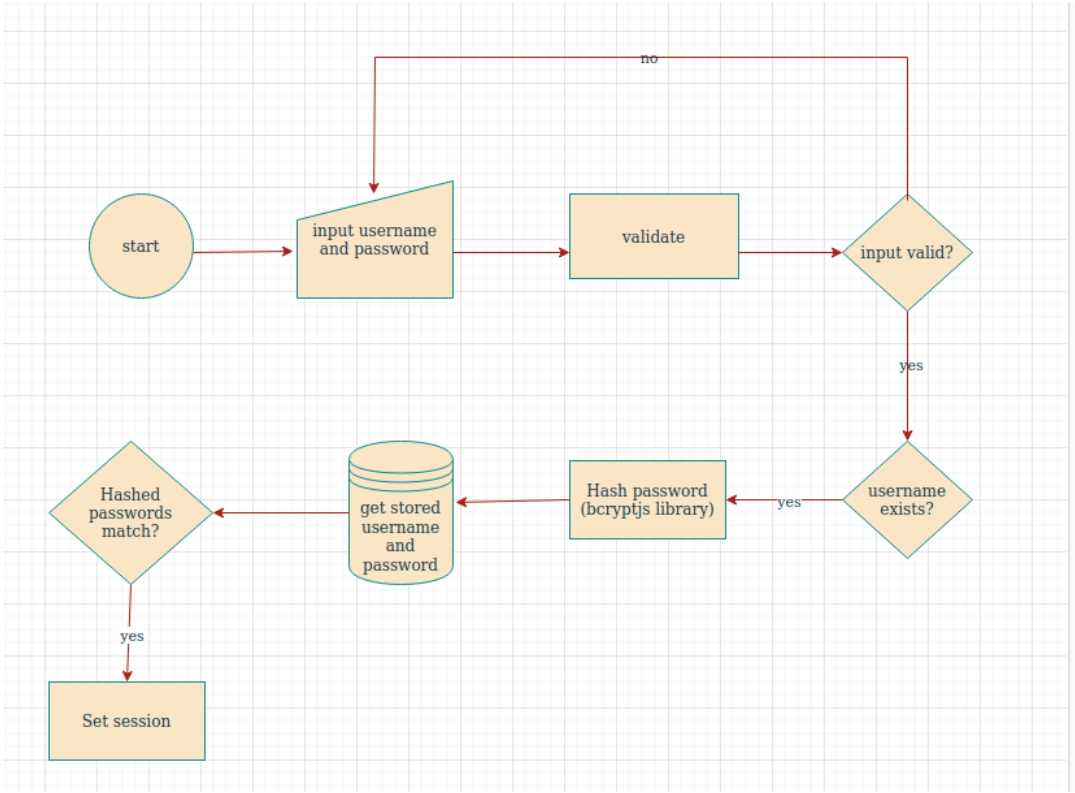
```
// Add a new record to the users table
  async addUser(password) {
      const pw = await bcrypt.hash(password, 10);
      var sql = "INSERT INTO Users (email, password) VALUES (? , ?)";
      const result = await db.query(sql, [this.email, pw]);
      console.log(result.insertId);
      this.id = result.insertId;
      return true;
  }
```



Log in: Express and PUG

- VIEW: Pug template – form
- CONTROLLER : app.js: receive input, check and send to model
- MODEL: User.js (model), retrieve stored, hashed password, compare to user input, set a session, send message to user

```
// Test a submitted password against a stored password
async authenticate(submitted) {
        // Get the stored, hashed password for the user
        var sql = "SELECT password FROM Users WHERE id = ?";
        const result = await db.query(sql, [this.id]);
        const match = await bcrypt.compare(submitted, result[0].password);
        if (match == true) {
            return true;
        }
        else {
            return false;
        }
    }
```



Cookies and Sessions

Cookies and sessions are used to store information about specific users between page loads. This is needed because **http is a 'stateless' protocol,** meaning it does not, by default, keep any relationship between one page load and another.

Sessions are stored on the server, and cookies are stored in the users browser.

https://www.freecodecamp.org/news/everything-you-need-to-know-about-cookies-for-web-development/

Sessions in express.js

- In express.js we set up the session framework in app.js

```
// Set the sessions
var session = require('express-session');
app.use(session({
  secret: 'secretkeysdfjsflyoifasd',
  resave: false,
  saveUninitialized: true,
  cookie: { secure: false }
}));
```

Sessions in express.js

- We can then set values onto the session when we authenticate

```
if (match) {
  req.session.uid = uId;
  req.session.loggedIn = true;
}
```

- The values will be on each session for that user when they send a request

```
if (req.session.uid) {
  res.send('Welcome back, ' + req.session.uid + '!');
} else {
  res.send('Please login to view this page!');
}
```

Cookies

- Tiny text files that are **stored in your browser**
- Use developer tools (Application tab) to examine cookies for a specific site
- You can also delete browser cookies manually in developer tools to terminate a session.

Logout

- **How to you think you can implement logout in your code?**

Lab

- Make sure to go through the lab methodically – run the code and check that the results are as expected each time you add code
- Don't miss any steps!!

- Consider how you can use the 'session' to adjust the code for any page and personalise it

LAB: Follow the lab sheet on Moodle

Sprint 3 reviews (Mon am):

- MyRecipePal 11.00
- Tepid Pizza 11.20
- Group 7   11.40
- Next Gen Devs 12.0
- Syntax Terror 12.20
- HSNE 12.40
- Kernal Kings 13.00
- Lexicon 13.20
- Backlog bashers 13.40

LAB: Follow the lab sheet on Moodle

Sprint 3 reviews (Tue am):

- Commitment issues 11.00
- 404 Founders 11.20
- Tue Group 9 11.40
- Connect 4 12.00
- Challengers 12.20
- Tue Group 9 12.40
- Tues am Group 4 13.00
- Tues am Group 6 13.15
- Yacht club 13.30
- Group 7 13.45