EXERCISE

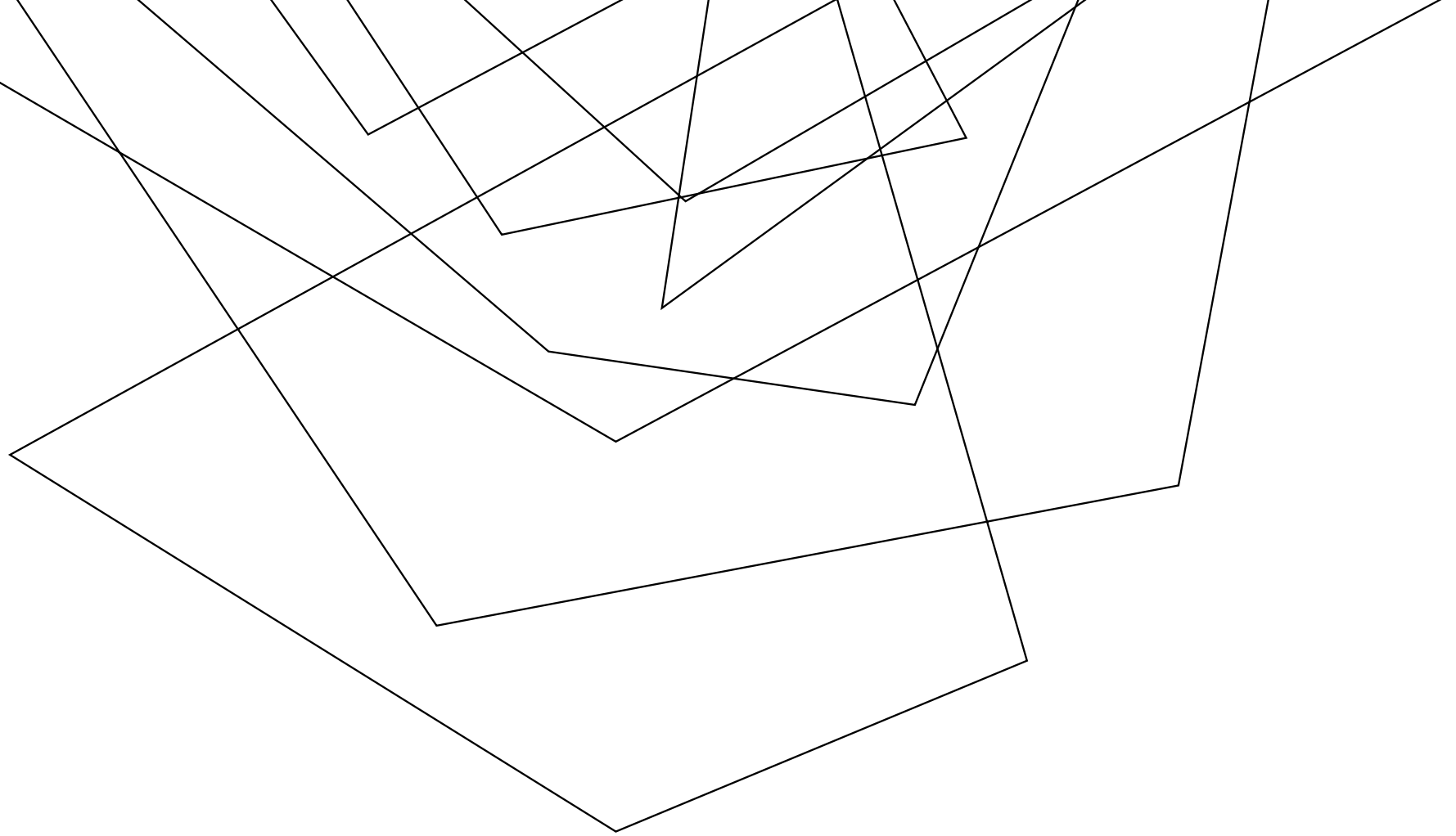
WEB SECURITY REVIEW

What is sub-resource integrity (SRI)? What threat model does it defeat?

ADMINISTRIVIA AND ANNOUNCEMENTS

Quiz 3

- Topic List posted
- 2025 review session re-posted



WEB SERVER SECURITY

EECS 677: Software Security Evaluation

Drew Davidson

LAST TIME

REVIEW: WEB CLIENT SECURITY

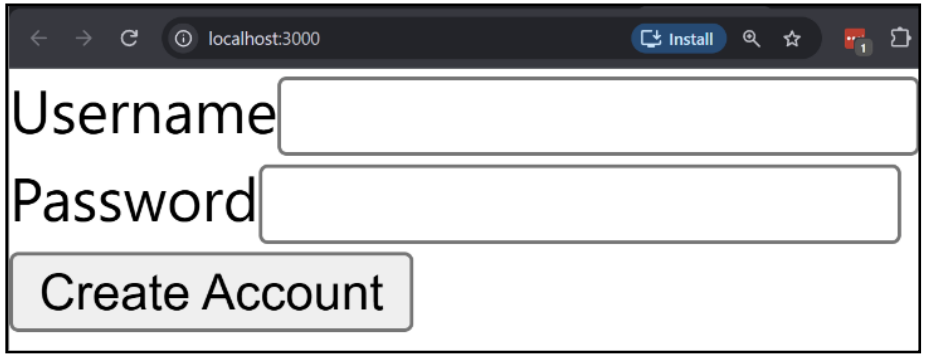
Frontend Overview

- React overview
- Password fields
- Sub-resource integrity
- POST requests

Backend Overview

- Set up a database
- Insert POST data

frontend: src/App.js



backend: src/index.js

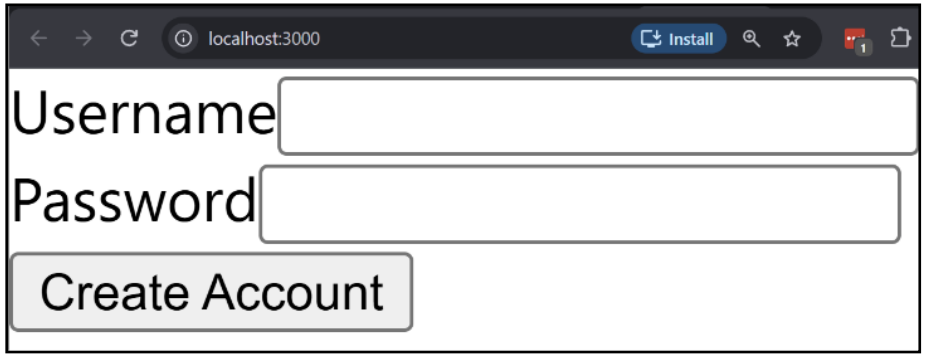
```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

TODAY

WEB SERVER SECURITY

- **Backend Security**
  - Validating SQL input
  - Securing storing password
- **Authentication tokens**
  - Generating tokens
  - Using tokens

frontend: src/App.js



backend: src/index.js

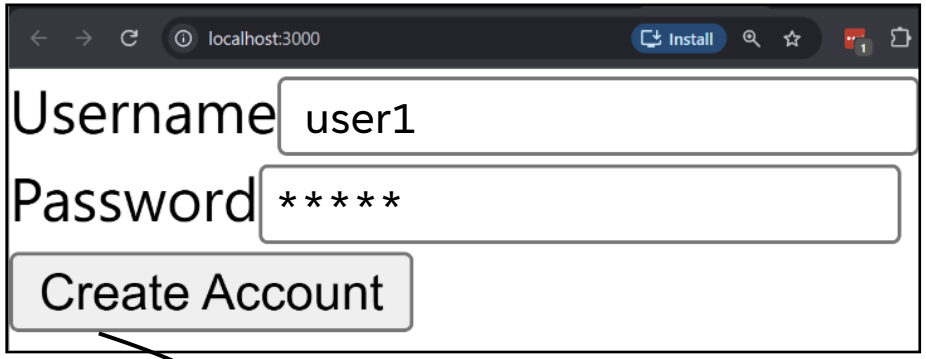
```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
```

frontend: src/App.js



backend: src/index.js

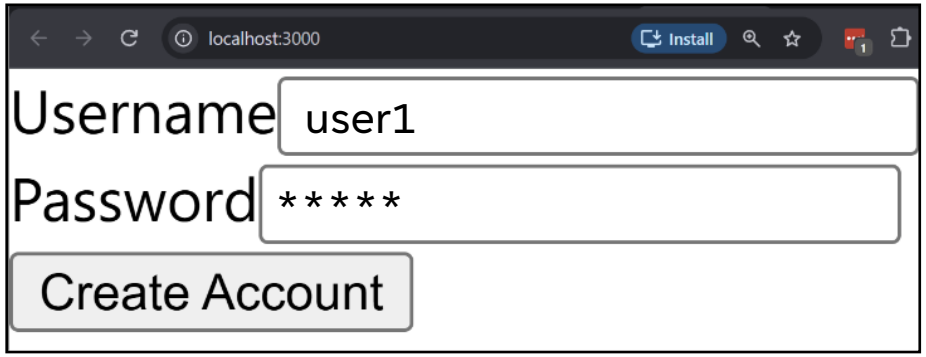
```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
})
…
```

VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
```

frontend: src/App.js



(on the server)
sqlite3 auth.db

```
sqlite> select * from users;
user1|pass1
```

backend: src/index.js

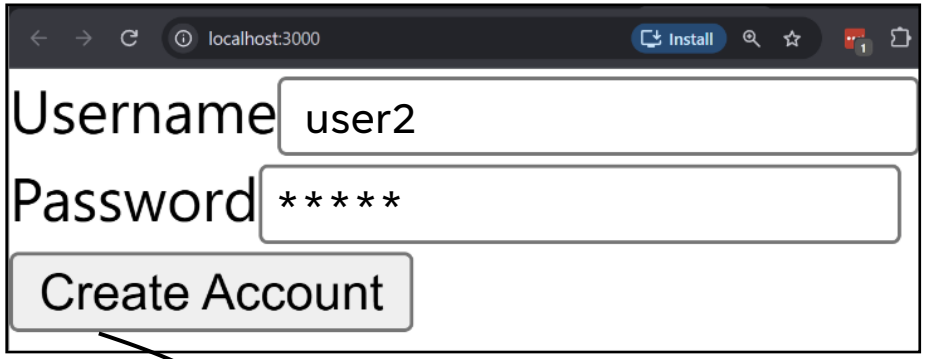
```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
INSERT INTO users VALUES ('user2','pass2')
```

frontend: src/App.js



backend: src/index.js

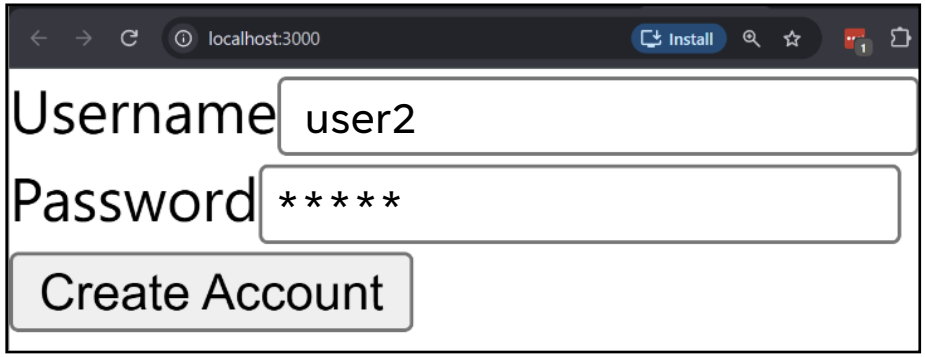
```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
})
…
```

VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
INSERT INTO users VALUES ('user2','pass2')
```

frontend: src/App.js



(on the server)
sqlite3 auth.db

```
sqlite> select * from users;
user1|pass1
user2|pass2
```

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

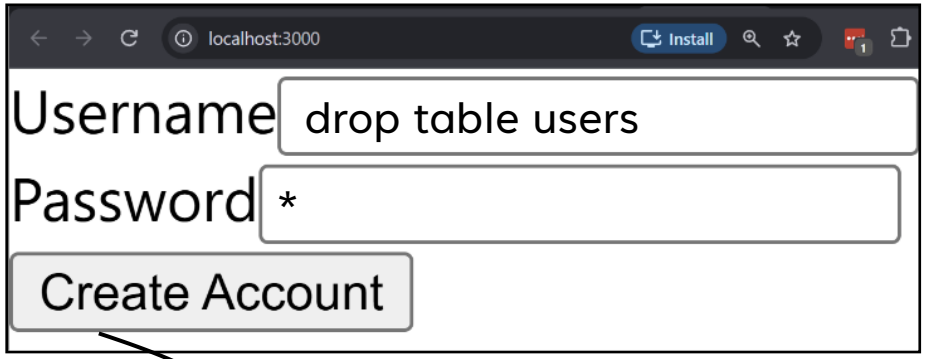
VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
INSERT INTO users VALUES ('user2','pass2')

INSERT INTO users VALUES ('drop table users','p')
```

frontend: src/App.js



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
})
…
```

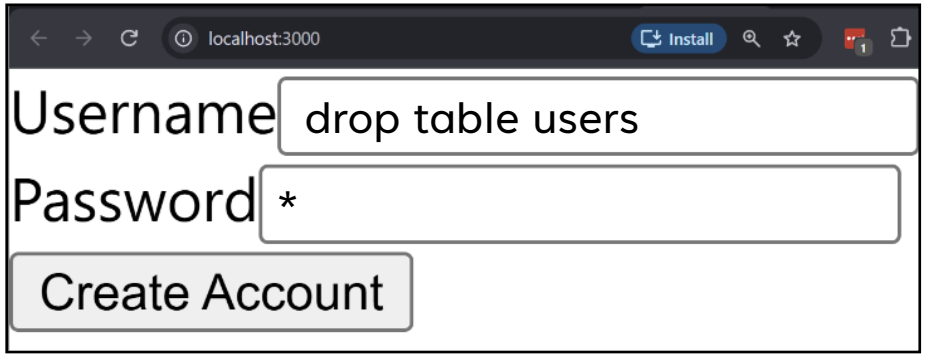
VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
INSERT INTO users VALUES ('user2','pass2')

INSERT INTO users VALUES ('drop table users','p')
```

frontend: src/App.js



(on the server)
sqlite3 auth.db

```
sqlite> select * from users;

user1|pass1
user2|pass2
drop table users|p
```

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

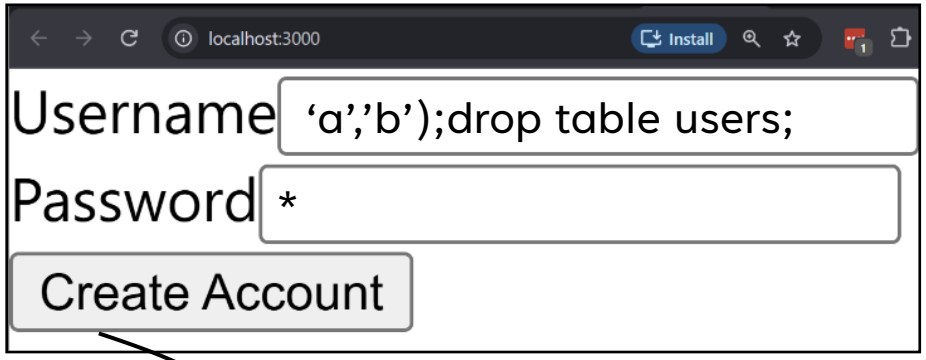
VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
INSERT INTO users VALUES ('user2','pass2')

INSERT INTO users VALUES ('drop table users','p')
INSERT INTO users VALUES ('a','b');drop table users;','p')
```

frontend: src/App.js



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
})
…
```

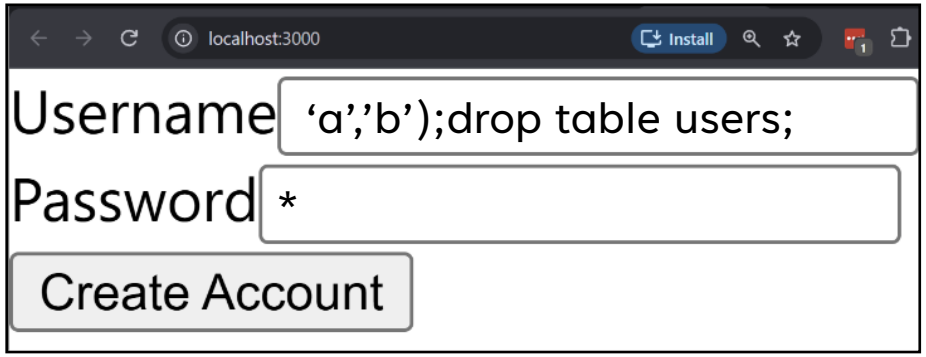
VALIDATING SQL INPUT

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
INSERT INTO users VALUES ('user2','pass2')

INSERT INTO users VALUES ('drop table users','p')
INSERT INTO users VALUES ('a','b');drop table users;','p')
```

frontend: src/App.js



(on the server)
sqlite3 auth.db

```
sqlite> select * from users;
Parse error: no such table: users
```

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

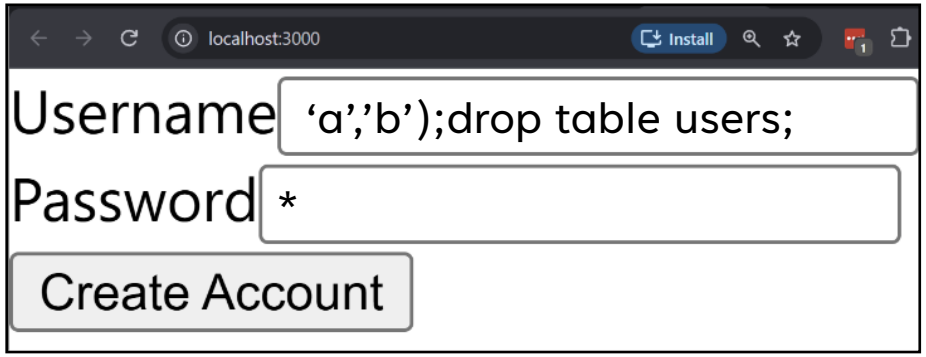
PREPARED STATEMENTS

WEB SERVER SECURITY

```
INSERT INTO users VALUES ('user1','pass1')
INSERT INTO users VALUES ('user2','pass2')

INSERT INTO users VALUES ('drop table users','p')
INSERT INTO users VALUES ('a','b');drop table users;','p')
```

frontend: src/App.js



(on the server)
sqlite3 auth.db

```
sqlite> select * from users;
Parse error: no such table: users
```

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

PREPARED STATEMENTS

WEB SERVER SECURITY

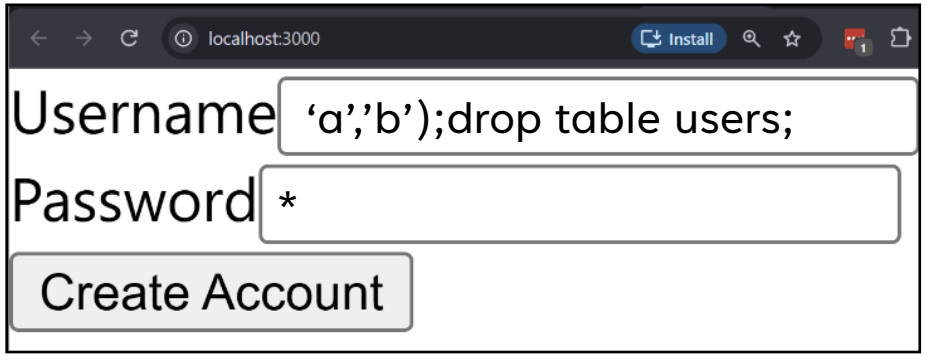
Special character confusion

The user-entered ' is indistinguishable from the query template '

SQL has a feature for this

prepared statements

frontend: src/App.js




```
INSERT INTO users VALUES ('a','b');drop table users;','p')
```

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
 })
 …
```

PREPARED STATEMENTS

WEB SERVER SECURITY

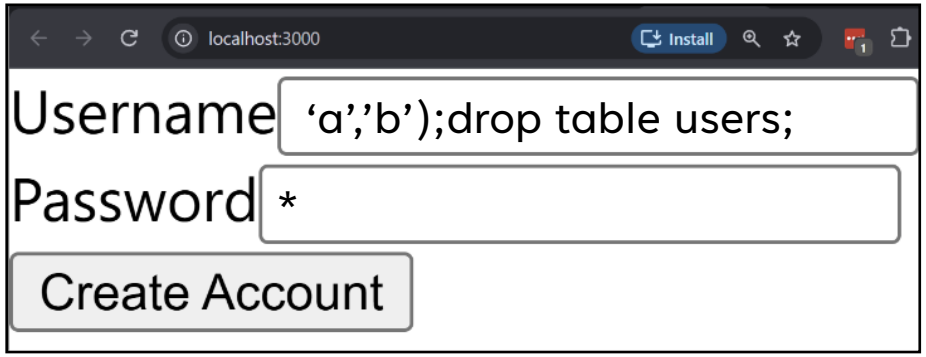
Special character confusion

The user-entered ' is indistinguishable from the query template '

SQL has a feature for this

prepared statements

frontend: src/App.js




```
INSERT INTO users VALUES ('a','b');drop table users;','p')
```

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
  …
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  db.exec(`INSERT INTO users VALUES ('${u}','{$p}'`)
  …
})
…
```

```
q.run(u,p);
```

PREPARED STATEMENTS

WEB SERVER SECURITY

Special character confusion

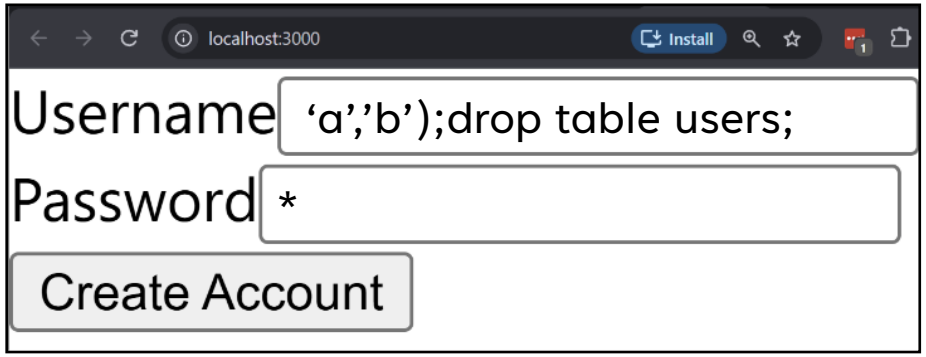
The user-entered ' is indistinguishable from the query template '

SQL has a feature for this

prepared statements

This prevents a SQL injection, but not a Javascript injection

frontend: src/App.js




```
INSERT INTO users VALUES ('a','b');drop table users;','p')
```

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
…
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
 })
 …
```

PREPARED STATEMENTS

WEB SERVER SECURITY

Special character confusion

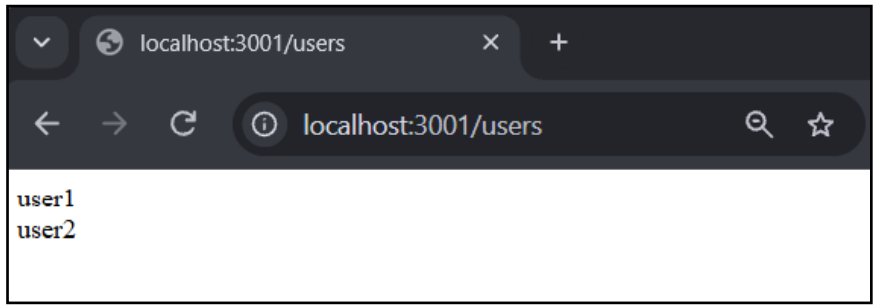
The user-entered ' is indistinguishable from the query template '

SQL has a feature for this

prepared statements

This prevents a SQL injection, but not a Javascript injection

Direct access /users endpoint



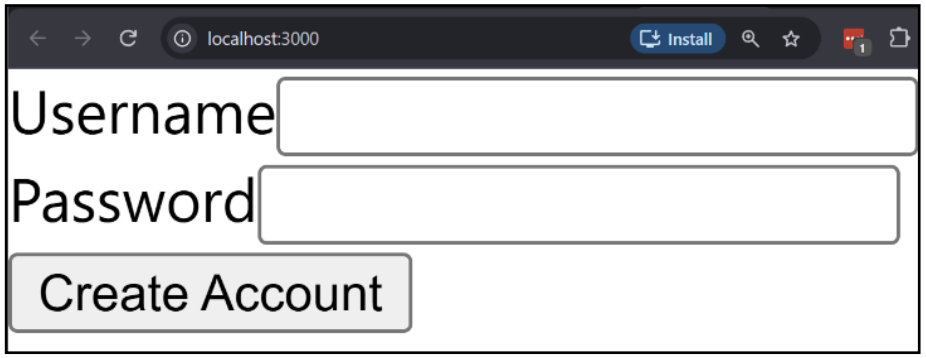
backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = rows.map((r) => r.name).join("<br/>");
    res.send(resp);
}
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

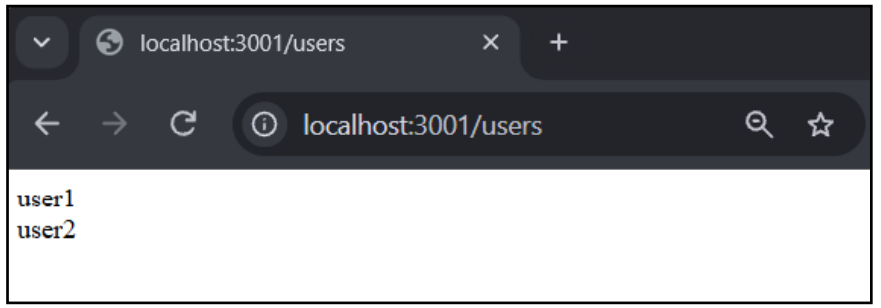
PREPARED STATEMENTS

WEB SERVER SECURITY

frontend: src/App.js



Direct access /users endpoint



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);

…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = rows.map((r) => r.name).join("<br/>");
    res.send(resp);
}

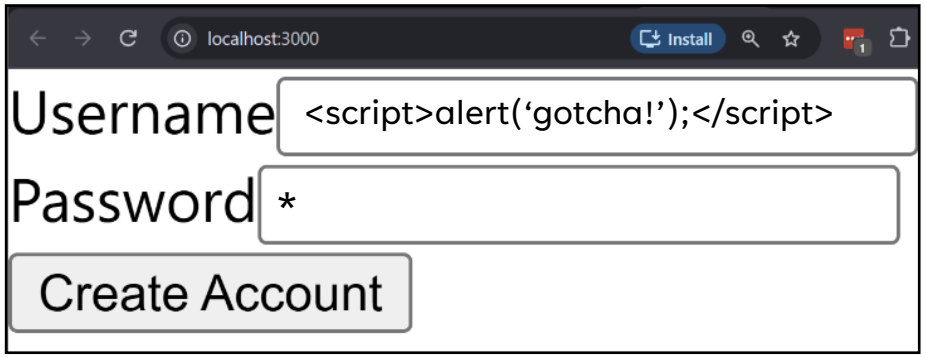
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

This prevents a SQL injection, but not a Javascript injection

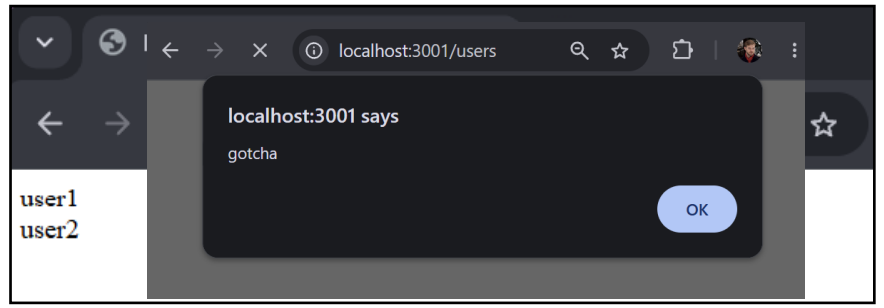
PREPARED STATEMENTS

WEB SERVER SECURITY

frontend: src/App.js



Direct access /users endpoint



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);

…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = rows.map((r) => r.name).join("<br/>");
    res.send(resp);
}

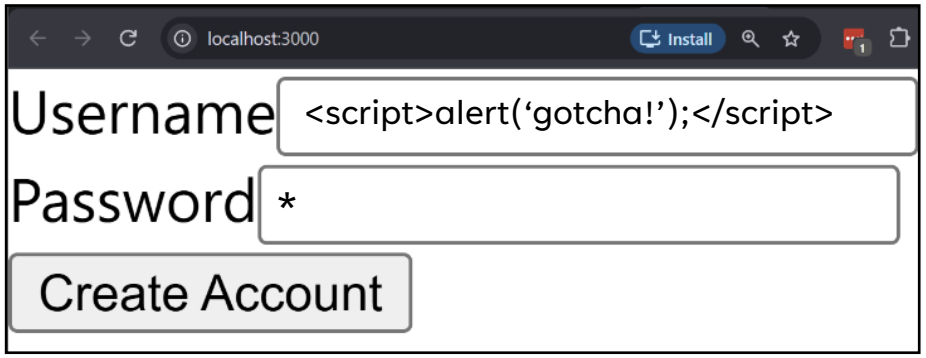
app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

This prevents a SQL injection, but not a Javascript injection

STILL NEED TO VALIDATE

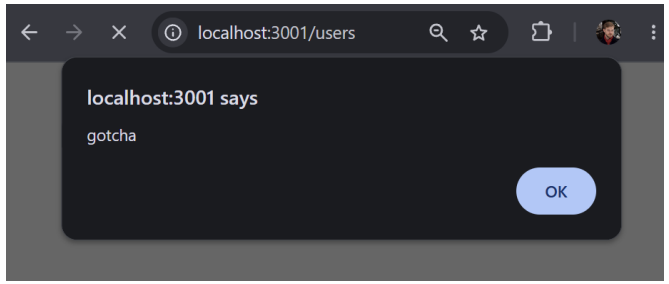
WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

Direct access /users endpoint



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);

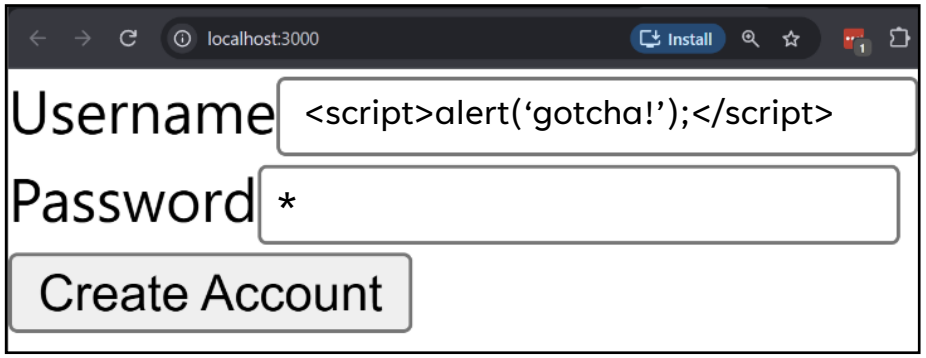
…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = rows.map((r) => r.name).join("<br/>");
    res.send(resp);
}

app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

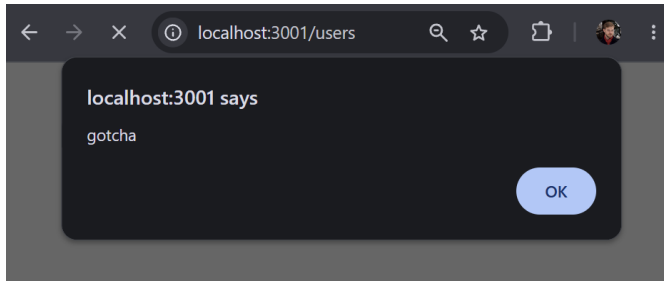
frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

Direct access /users endpoint



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);

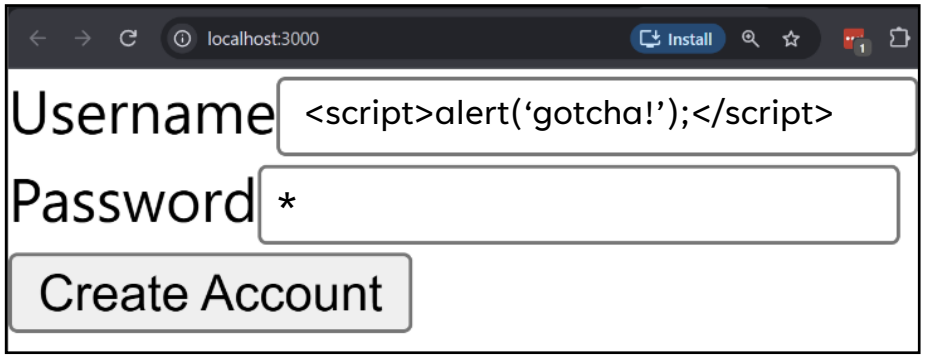
…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = rows.map((r) => r.name).join("<br/>");
    res.send(resp);
}

app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw;validator.escape(req.body.user)
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

frontend: src/App.js

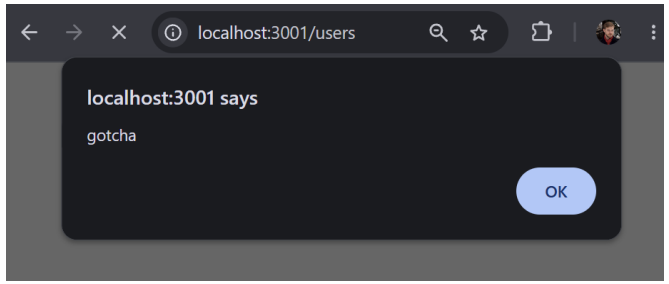


This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

Direct access /users endpoint



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);

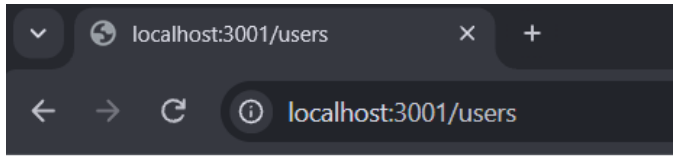
…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = rows.map((r) => r.name).join("<br/>");
    res.send(resp);        JSON.stringify(rows);
}

app.post("/register", (req, res) => {
  let u = validator.escape(req.body.user)
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

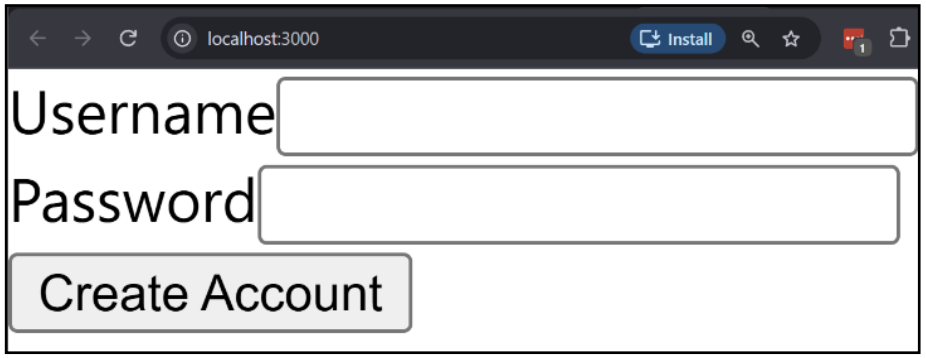
STILL NEED TO VALIDATE

WEB SERVER SECURITY

Direct access /users endpoint



frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);

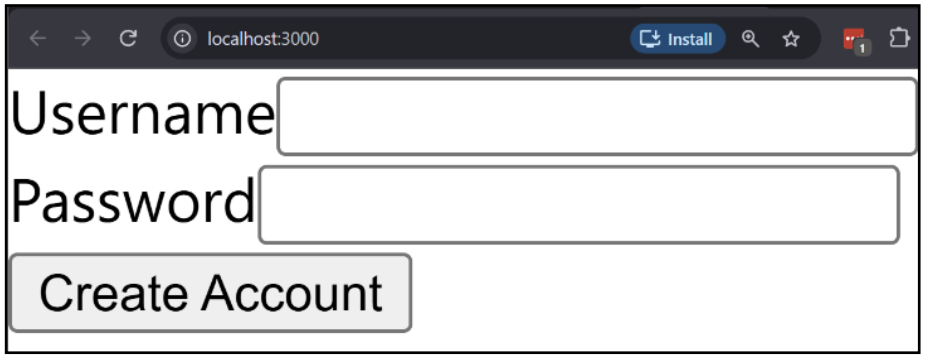
…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = JSON.stringify(rows);
    res.send(resp);
}

app.post("/register", (req, res) => {
  let u = validator.escape(req.body.user)
  let p = req.body.pw;
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

frontend: src/App.js

```
import { useState } from 'react'
…

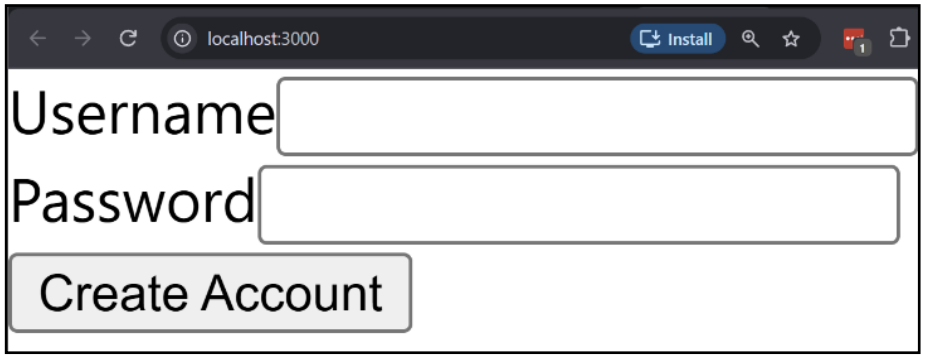
function App()(

…
return (<div>
    <UserList />}
    <span>Username</span>
    <input type="text" onChange={…}/>
    <br />
    <span>Password</span>
    <input type="password" onChange={…}/>
    <br />
    <input type="button" onClick={…} value="Create Account"/>
  </div>)
}
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

frontend: src/App.js

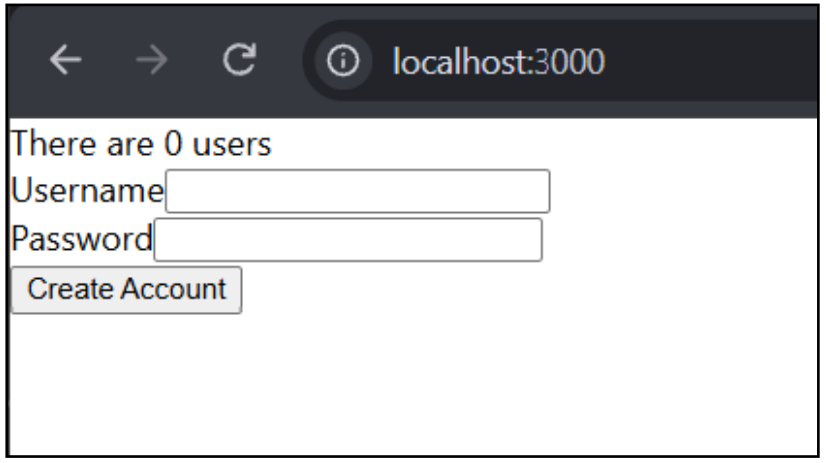
```
import { useState } from 'react'
…
function UserList() {
  const [users, setUsers] = useState([]);
  return (<div>
      <div>There are { users.length} users</div>
  </div>)
}

function App()(
…
return (<div>
    <UserList />}
    <span>Username</span>
    <input type="text" onChange={…}/>
    <br />
    <span>Password</span>
    <input type="password" onChange={…}/>
    <br />
    <input type="button" onClick={…} value="Create Account"/>
  </div>)
}
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

frontend: src/App.js

```
import { useState } from 'react'
…
function Users(){
  const [users, setUsers] = useState([]);
  return (<div>
      <div>There are { users.length} users</div>
  </div>)
}

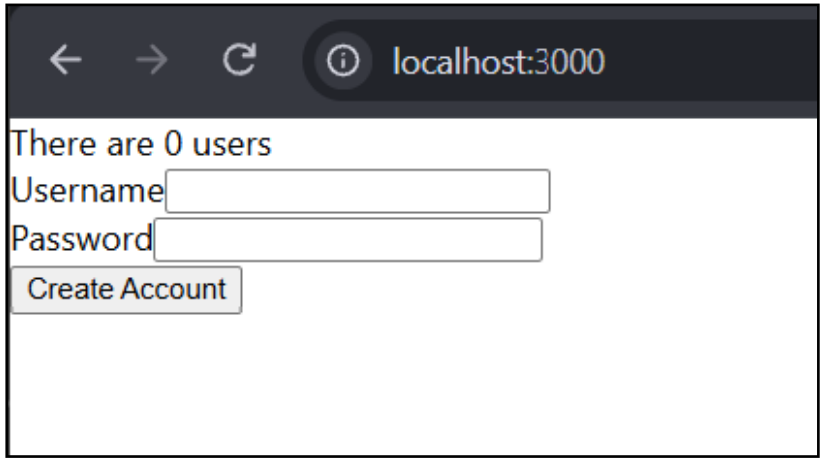
function App()(
…
return (<div>
    <UserList />}
    <span>Username</span>
    <input type="text" onChange={…}/>
    <br />
    <span>Password</span>
    <input type="password" onChange={…}/>
    <br />
    <input type="button" onClick={…} value="Create Account"/>
 </div>)
}
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY



frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

frontend: src/App.js

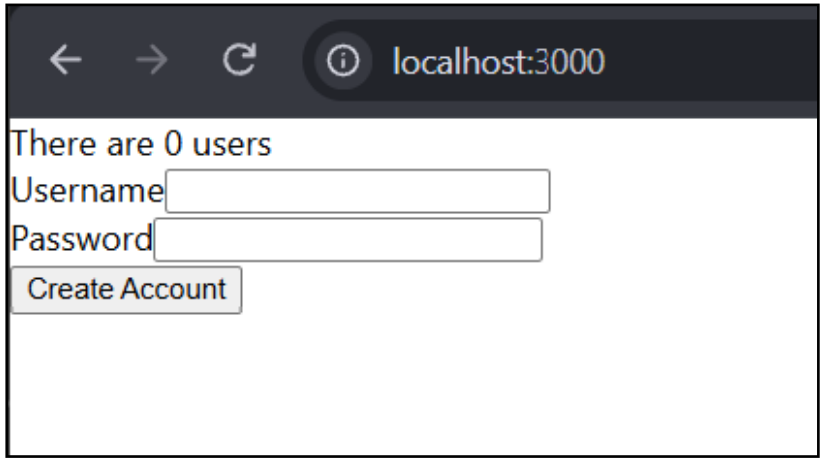
```
import { useState } from 'react'
…
function Users(){
  const [users, setUsers] = useState([]);
  axios.get("http://localhost:3001/users")
    .then(resp => { setUsers(resp.data) })
  return (<div>
      <div>There are { users.length} users</div>
  </div>)
}
```

```
function App()(
…
return (<div>
    <UserList />}
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

frontend: src/App.js

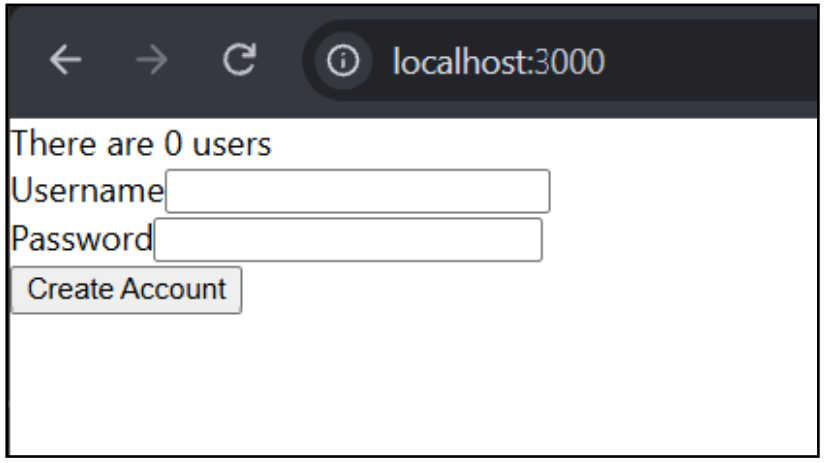
```
import {useEffect, useState } from 'react'
…
function Users(){
  const [users, setUsers] = useState([]);
   axios.get("http://localhost:3001/users")
     .then(resp => { setUsers(resp.data) })
   return (<div>
       <div>There are { users.length} users</div>
  </div>)
}
```

```
function App()(
…
return (<div>
    <UserList />}
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

frontend: src/App.js

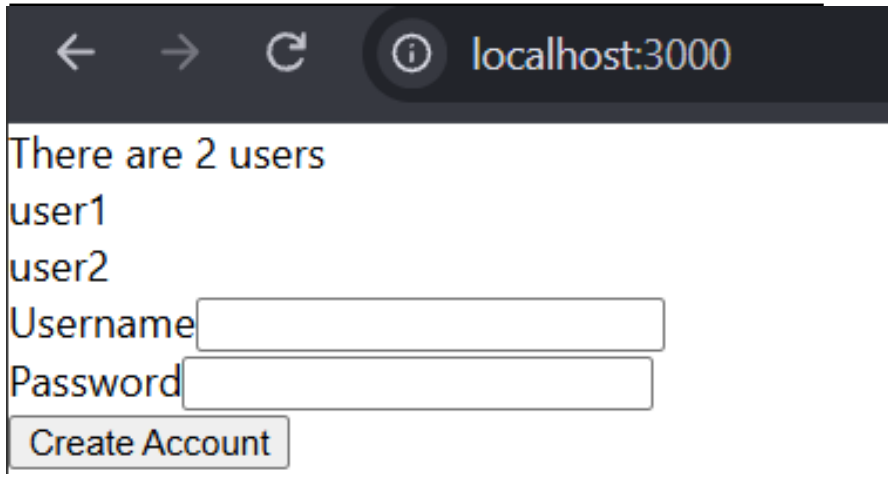
```
import {useEffect, useState } from 'react'
…
function Users(){
  const [users, setUsers] = useState([]);
  useEffect(() => {
     axios.get("http://localhost:3001/users")
       .then(resp => { setUsers(resp.data) })
  }, [])
  return (<div>
      <div>There are { users.length} users</div>
  </div>)
}
```

```
function App()(
…
return (<div>
    <UserList />}
```

STILL NEED TO VALIDATE

WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

1) Sanitize input regardless

2) Endpoints shouldn't really deliver HTML

frontend: src/App.js

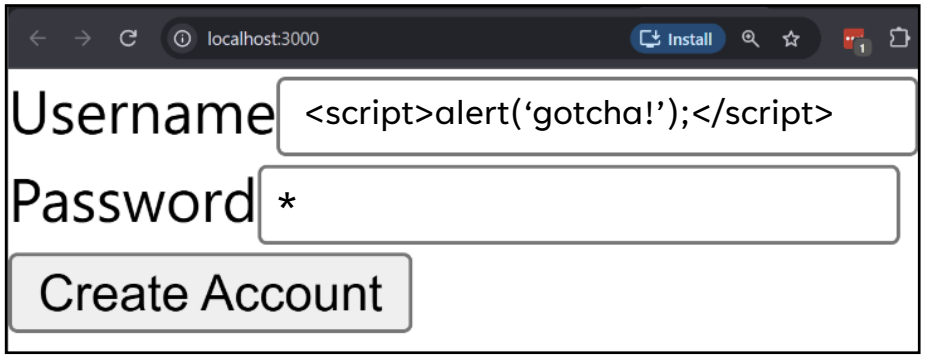
```
import {useEffect, useState } from 'react'
…
function Users(){
  const [users, setUsers] = useState([]);
   useEffect(() => {
      axios.get("http://localhost:3001/users")
        .then(resp => { setUsers(resp.data) })
   }, [])
   return (<div>
       { users.map((u) => {return <div>{u.name}</div>})}
       <div>There are { users.length} users</div>
  </div>)
}
```

```
function App()(
…
return (<div>
    <UserList />}
```

STILL NEED TO VALIDATE

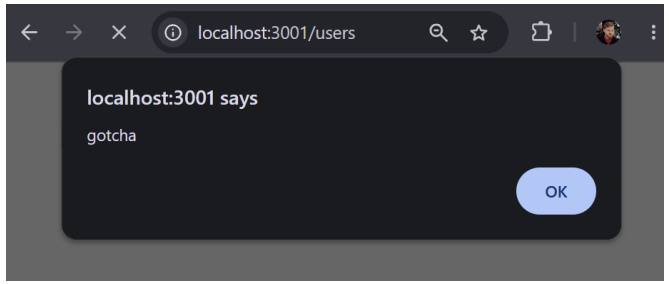
WEB SERVER SECURITY

frontend: src/App.js



This prevents a SQL injection, but not a Javascript injection

Direct access /users endpoint



backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);

…
app.get("/users", (req, res) => {
    const q = db.prepare('SELECT name from users');
    const rows = q.all();
    let resp = rows.map((r) => r.name).join("<br/>");
    res.send(resp);
}

app.post("/register", (req, res) => {
  let u = req.body.user;
  let p = req.body.pw; validator.validate(req.body.user)
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
})
…
```

SECURING STORED PASSWORDS

WEB SERVER SECURITY

PASSWORD STORAGE

Sensitive data should not be stored in the clear!

Bad

Store a password in cleartext

insert(user, password given)

lookup(user, password given) == password stored

db leak or unauthorized access
instantly compromises passwords

Better

Store a hash of the password

pw_hash = HASH(password given)
insert(pw_hash)

lookup(user, pw_hash) == HASH(password given))

db leak or unauthorized access
quickly compromises common passwords

SECURING STORED PASSWORDS

WEB SERVER SECURITY

PASSWORD STORAGE

Sensitive data should not be stored in the clear!

Better

Store a hash of the password

pw_hash = HASH(password given)
insert(pw_hash)

lookup(user, pw_hash) == HASH(password given))

db leak or unauthorized access
quickly compromises common passwords

Best Practice

Store a hash of the password + salt

saltedpw_hash = HASH(password given + salt)
insert(user, saltedpw_hash, salt)

lookup(pw_hash) == HASH(password given))

db leak or unauthorized access
eventually compromises common passwords

SECURING STORED PASSWORDS

WEB SERVER SECURITY

PASSWORD STORAGE

Sensitive data should not be stored in the clear!

Best Practice

Store a hash of the password + salt

saltedpw_hash = HASH(password given + salt)
insert(user, saltedpw_hash, salt)

lookup(pw_hash) == HASH(password given))

backend: src/index.js

```
…
db.exec(`CREATE TABLE IF NOT EXISTS users (
    name TEXT UNIQUE NOT NULL, pw TEXT NOT NULL)`);
…

app.post("/register", (req, res) => {
  let u = validator.escape(req.body.user)
  let p = req.body.pw;
  p = await bcrypt.hash(p, 10);
  let q = db.prepare(`INSERT INTO users VALUES(?,?)`);
  q.run(u,p);
  …
 })
 …
```

TODAY

WEB SERVER SECURITY

- **Backend Security**
  - Validating SQL input
  - Securing storing password
- **Authentication tokens**
  - Generating tokens
  - Using tokens

AUTHENTICATION TOKENS

WEB SERVER SECURITY

We want to touch as little plaintext key material as possible

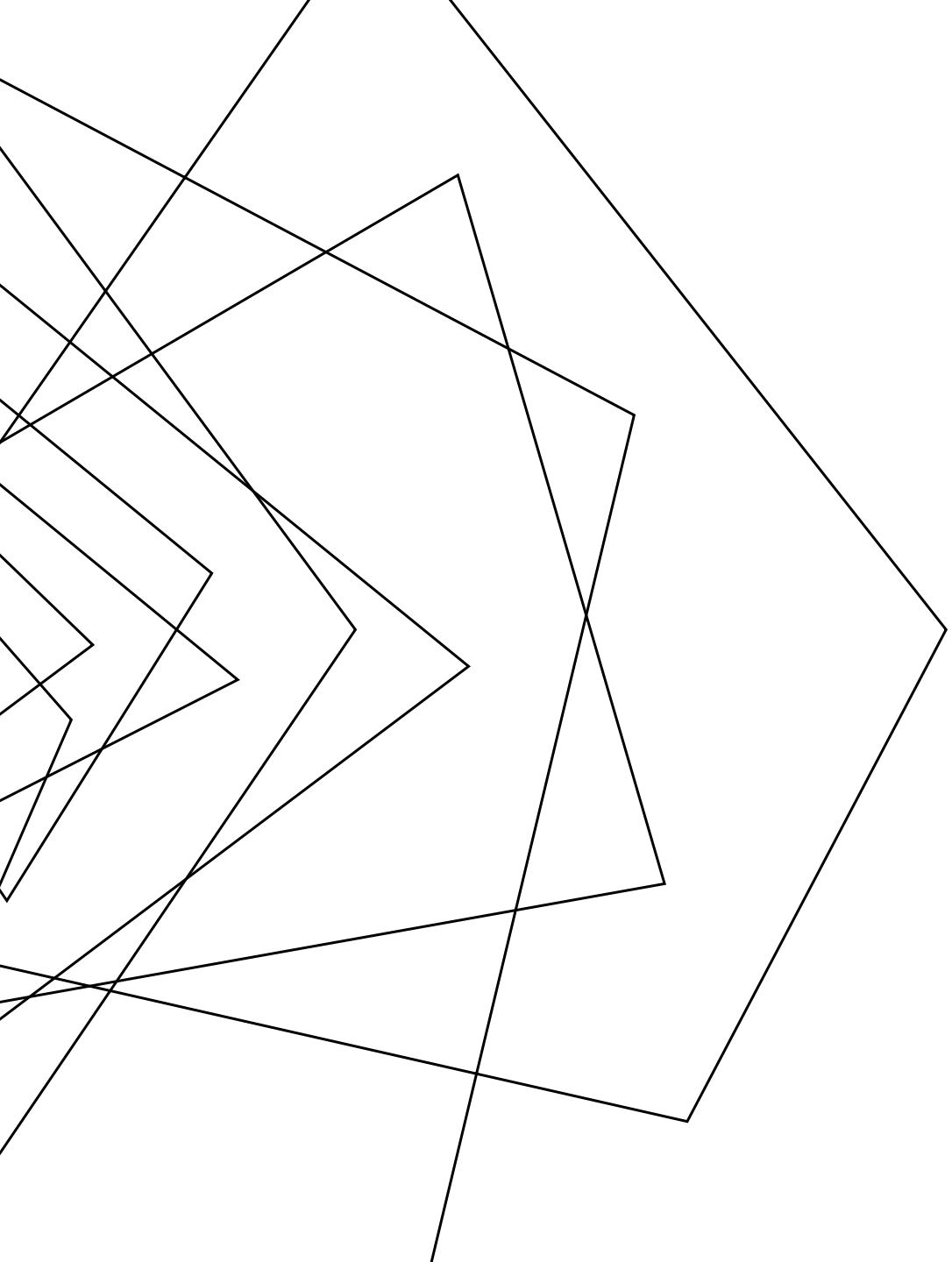
- Much better to hold/pass **proof** of authentication

AUTHENTICATION TOKENS

WEB SERVER SECURITY

We want to touch as little plaintext key material as possible

- Much better to hold/pass **proof** of authentication



LECTURE END!

That's basically all I wanted to talk about for Web Security

Quiz 3 is coming up, then we'll have a grab bag of topics

*If **YOU** have a topic you'd like discussed, let me know!*