User Authentication

Activity

Crack your hash [Due: Thursday]

Implement authentication [Due: Next Tuesday]
-Choose any database for persistence
-Implement user registration where user choose a username and password
-Implement user login using username/password
-On successful login, display message with the authenticated username (No need for cookies and sessions for this activity)

Databases

MySQL

- A program that must be downloaded, installed, and ran
- Is a server
  - By default, listens on port 3306
- Install a library for your language that will connect to the MySQL server

MySQL

- After MySQL is running and you install a library to connect to it..
- Connect to MySQL Server by providing
  - url of database
  - username/password for the database
    - Whatever you chose when setting up the database

```
val url = "jdbc:mysql://localhost/mysql?serverTimezone=UTC"
val username = "root"
val password = "12345678"

var connection: Connection = DriverManager.getConnection(url, username, password)
```

MySQL - Security

- For real apps that you deploy
  - Do not check your password into version control!
    - A plain text password in public GitHub repo is bad
    - Attacker can replace localhost with the IP for your app and can access all your data
  - Common to save the password in a environment variable to prevent accidentally pushing it to git
  - Do not use the default password for any servers you're running
    - This is what caused the Equifax leak (Not with MySQL)
- Attacker have bots that scan random IPs for such vulnerabilities

```
val url = "jdbc:mysql://localhost/mysql?serverTimezone=UTC"
val username = "root"
val password = "12345678"

var connection: Connection = DriverManager.getConnection(url, username, password)
```

MySQL

- Once connected we can send SQL statements to the server

```
val statement = connection.createStatement()
statement.execute("CREATE TABLE IF NOT EXISTS players (username TEXT, points INT)")
```

- If using inputs from the user always use prepared statements
  - Indices start at 1 in this example 😢

```
val statement = connection.prepareStatement("INSERT INTO players VALUE (?, ?)")

statement.setString(1, "mario")
statement.setInt(2, 10)

statement.execute()
```

MySQL - Security

- Not using prepared statements?
    - Vulnerable to SQL injection attacks
- If you concatenate user inputs directly into your SQL statements
    - Attacker chooses a username of "';DROP TABLE players;"
    - You lose all your data
    - Even worse, they find a way to access the entire database and steal other users' data
    - SQL Injection is the most common successful attack on servers

MySQL

- Send queries to pull data from the database
- Returns a ResultSet in this example
    - The next() methods queue the next result of the query
    - next returns false if there are no more results to read
- Can read values by index of by column name

```
val statement = connection.createStatement()
val result: ResultSet = statement.executeQuery("SELECT * FROM players")

var allScores: Map[String, Int] = Map()

while (result.next()) {
  val username = result.getString("username")
  val score = result.getInt("points")
  allScores = allScores + (username -> score)
}
```

SQL

- SQL is based on tables with rows and column
    - Similar in structure to CSV except the values have types other than string
- How do we store an array or key-value store?
    - With CSV our answer was to move on to JSON
    - SQL answer is to create a separate table and use JOINs
    - Or, try MongoDB

MongoDB

- A document-based database
- Instead of using tables, store data in a structure very similar to JSON
- In python/JS
    - Insert dictionaries/objects directly
- Each object is stored in a collection

```
chat_collection.insert_one({'username': 'hartloff', 'message': 'hello'})
```

MongoDB

- Retrieve documents using find
- Find takes a key-value store and returns all documents with those values stored at the given keys
    - Ex. {'username': 'hartloff'} returns all documents with a username of hartloff
- To retrieve all documents, use an empty key-value store {}

```
collection.find({'username': 'hartloff'})
collection.find({})
```

MongoDB vs. SQL

- MongoDB is unstructured
  - Can add objects in any format to a collection
  - Can mix formats in a single collection
    - Ie. In a single collection the documents can have different attributes
- SQL is structured (That's what the S stands for)
  - Table columns must be pre-defined
    - All rows have the same attributes
    - Adding a column can be difficult
  - Fast!

MongoDB vs. SQL

- Hot Take
    - MongoDB is best for prototyping when the structure of your data is constantly changing
        - Take advantage of the flexibility
    - SQL is best once your data has a defined structure
        - Take advantage of the efficiency

Chat app code

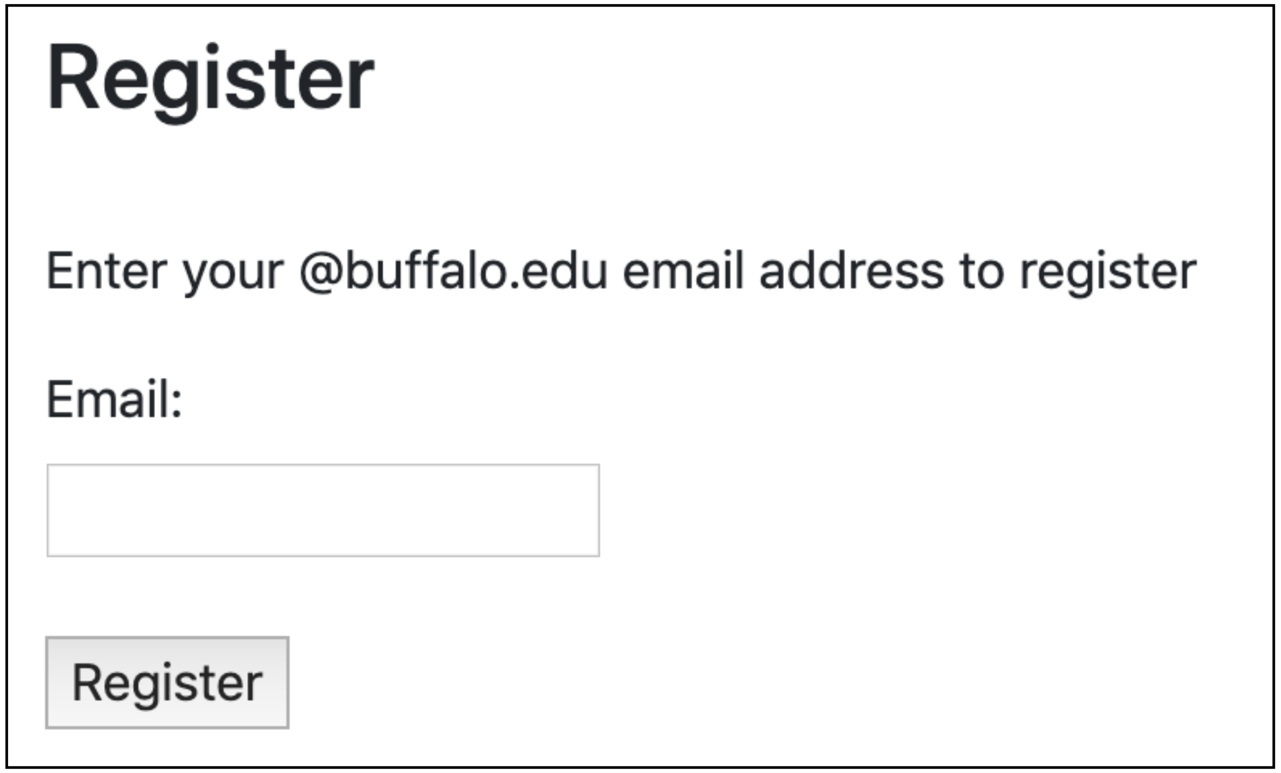
Authentication

Authentication

- Registration
    - Users can create an account on your app
- Authentication
    - Verify that a user is [likely] a registered account holder
    - Log them into your app

Authentication

- Registration
    - Can be a simple web form
    - Read the user inputs on the server
- Common to affiliate an account with a valid email address
    - And verify that email

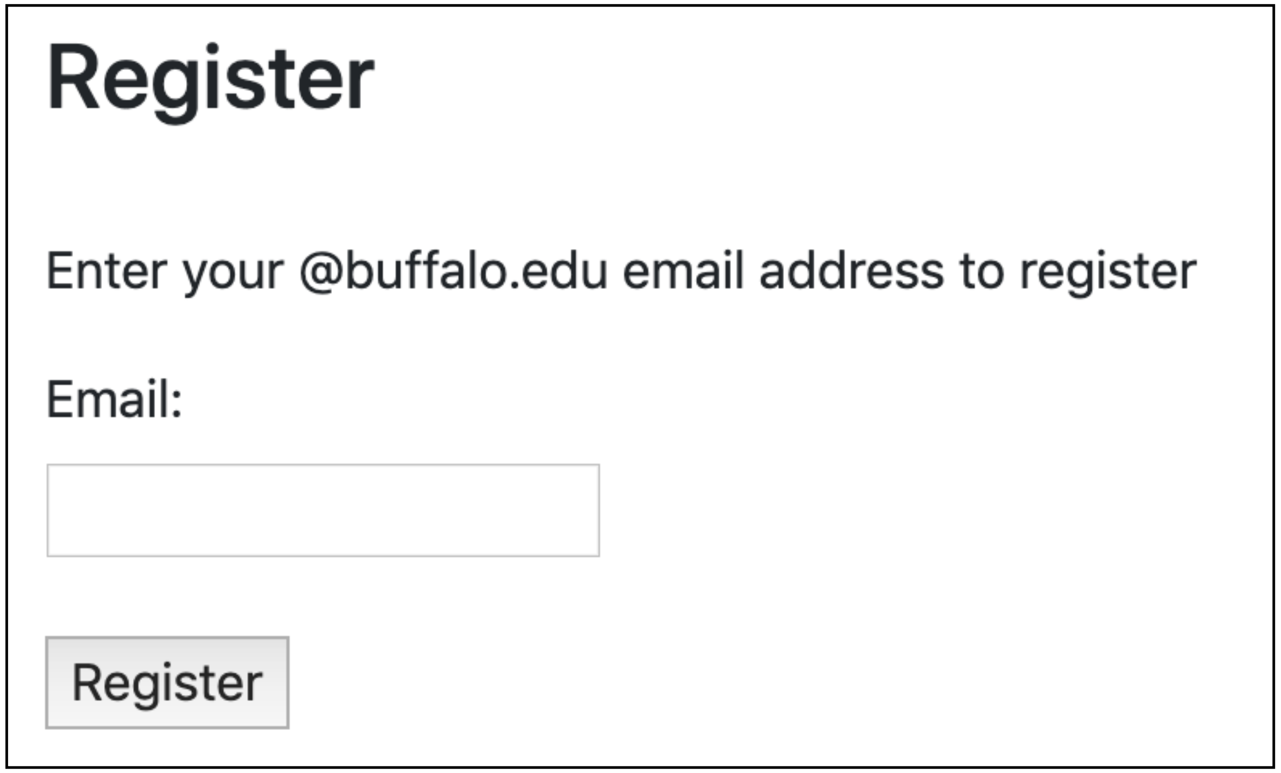


Register

Enter your @buffalo.edu email address to register

Authentication

- In this example
  - User enters their @buffalo.edu email address
  - The app sends them an email with a personalized set password link
  - User clicks that link to set their password
  - Their username is their UBIT

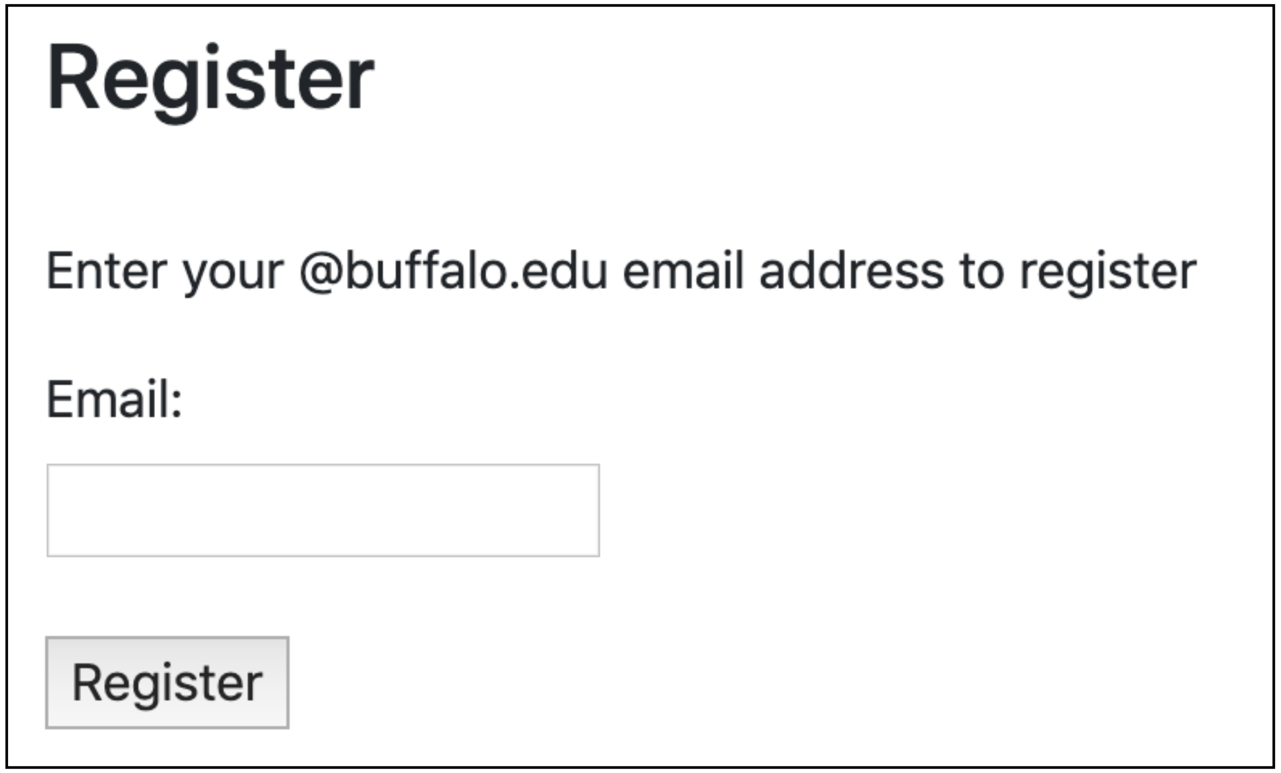


Register

Enter your @buffalo.edu email address to register

Authentication

- In this example
  - User enters their @buffalo.edu email address
  - The app sends them an email with a personalized set password link
  - User clicks that link to set their password
  - Their username is their UBIT



Register

Enter your @buffalo.edu email address to register

Email:

Authentication

- On the server
  - Store each username/password in a database
  - This data must persist so the users can log in
  - What if this database is compromised?
    - Perhaps by a SQL injection attack

Authentication

- NEVER store passwords as plain text
- Not even the admins of a website should know the passwords of their users
- We do this by **hashing** the passwords and storing only the hashes

Authentication - Hashing

- **Hashing** is a way of converting an input into a different value
- **Cryptographic Hashing** is a form of hashing with the goal of being a one-way function
  - Can easily compute the hash of a value
  - Cannot compute the original value given only its hash value
- We store only the cryptographic hash of passwords

Authentication - Hashing

- There are many attacks on hashes that have been developed over the years
- One attack is to use a Rainbow Table
  - A table containing the start and end of "chains" of hashes
  - Repeatedly rehash the start to reach the end
  - To attack a hash, rehash until you reach the end of a chain, then rehash the beginning to find the value before the hash
  - Effectively trades space for time

Authentication - Hashing

- To prevent attacks such as Rainbow Tables
- **Salt** the hashes
    - Before hashing a password, generate a random string
    - Concatenate this string with the password, then hash
    - The salt and hash are stored together

Authentication

- The bcrypt library implements hashing, salting, and other security related functions
- Available in many different languages

UB Infinite Code

Cookies

- Lastly, you don't want your users to have to authenticate for every single action they make
- But, HTTP[S] is a stateless protocol
- For this we use cookies to set a session token for the authenticated user
- This token is sent with each subsequent request
- Server verifies this token, looks up the user accosted with it, and trusts that this is a valid request

CSE199 Code

Activity

Crack your hash [Due: Thursday]

Implement authentication [Due: Next Tuesday]
-Choose any database for persistence
-Implement user registration where user choose a username and password
-Implement user login using username/password
-On successful login, display message with the authenticated username (No need for cookies and sessions for this activity)