11-27-2023

Implementation and Analysis of Security Measures for a Banking Web Application

Eric Mingeon Sung

Implementation and Analysis of Security Measures for a Banking Web Application

by

Eric Mingeon Sung

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

November 27, 2023

Honors Capstone Project Director: Dr. Jacob Hauenstein

Eric Sung 12/2/2023
Student (signature) Date

[illegible] 12/5/23
Project Director (signature) Date

Letha Etzkorn Digitally signed by Letha Etzkorn
Date: 2023.12.05 10:51:17 -06'00'
Department Chair (signature) Date

Sean Lane Digitally signed by Sean M Lane
Date: 2023.12.06 10:43:30 -06'00'
Honors College Dean (signature) Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted with the final manuscript.

Eric Sung
Student Name (printed)

Eric Sung
Student Signature

12/2/2023
Date

Table of Contents

Abstract

In the dynamic online banking environment, the need for robust security measures is growing rapidly. As financial institutions transition to web-based services, user data integrity and security of financial transactions remain a major concern. This paper, extending an ongoing Bank Management Utility project for CS499 Senior Project: Team Software Design for developing an online banking web application, presents three security features and provides a comprehensive review of each feature. The features are implemented and tested in the banking web application “Summit Financial Corporation”. The specific security features discussed in the paper are query parameterization, session-based user authentication, and password hashing, which enhances the protection of sensitive user data in the banking web application.

Introduction

According to IBM's 2023 Cost of a Data Breach Report, the global average cost of a data breach in 2023 was $4.45 million, which is a 15% increase over 3 years. [1] The financial industry is the second most affected industry with an average cost of $5.90 million per breach in 2023 after the healthcare industry. Due to the nature of handling millions of customers' financial assets and personal data, the stake is significantly higher for the financial sector to provide robust security to protect the customers' assets and maintain their trust with their customers. With the fast transition from in-person to online transactions in the recent decade, the reliability of banking services on digital devices has become a greater concern. Users now have more access points to banking services and data than ever, which implies that there are more areas with potential vulnerabilities for the data to be breached.

James Scott, a cofounder of the Institute for Critical Infrastructure Technology, stated, "There is no silver bullet solution with cyber security, a layered defense is the only viable solution." [2] Scott's insight underscores the fundamental importance of a multifaceted approach to security, adding multiple layers of security measures to achieve robust protection. Modern banks apply a similar approach to their security. Some of the common security features include multifactor authentication, encryption, privacy policies fraud detection, and firewalls. This paper presents and analyzes the layers of security features implemented in the banking web application developed for the CS499 Senior Project: Team Software Design.

The accompanying project, Bank Management Utility, focuses on developing a web-based banking service with account management, digital transactions, and administrator oversight. Acknowledging the seriousness of security requirements in the movement of private assets and personal data, the system takes a robust approach to safeguard sensitive financial data and protect users from potential harm. This paper proposes the integration of data encryption, query parameterization, the user authentication in the Bank Management Utility project to mitigate the risk of malicious attacks.

Bank Management Utility

Overview

The assignment for the CS499 Senior Project: Team Software Design was Bank Management Utility. The goal of the project was to implement a fully functional banking service as a web application that is accessible to both customers and tellers. The project was designed for a four-member team with tasks ranging from the implementation of the client-server system to user-specific tasks to enable general banking functionalities. The project defines three user types, five banking accounts, and lists of expected utilities of these categories. It is important to note that role and user type are used synonymously in this paper, and the use of the word account implies a banking account, such as a checking or savings account, rather than a user login account. See the project requirements document [1] for detailed descriptions of requirements.

The banking web application, which we named Summit Financial Corporation (SFC), consists of three user roles: customer, teller, and administrator. The customer is the everyday user of the bank; multiple customers can exist in the system, and they can review their account information, make deposits and withdrawals, transfer funds between two accounts, and make bill payments. Multiple customers can also make transactions and request data simultaneously. The teller is the manager of customers' information and banking accounts. Multiple tellers can exist in the system, and the tellers are able to review all customer account information, create, modify, and delete an account, make deposits, withdrawals, and transactions on customer accounts, and generate a 1099-INT form for the customer. The administrator is regarded as the system administrator. The administrator is responsible for adding new tellers to the system, modifying, and deleting teller information, and reviewing customers' information and accounts.

Design Architecture

The web application for SFC was developed entirely in JavaScript programming language powered by Node.js, a cross-platform open-source server environment that is commonly used in web development. The application is divided into two separate microservices, the frontend, and the backend to provide a division of functions, an extra layer of access for security, as well as ease of distributed development in the group. For the frontend, the React.js framework was chosen due to its sheer popularity in frontend development, which implies an abundance of community supports, and numerous utility features, such as .jsx scripting that combines JavaScript and HTML in a single file and the use of components and states to modularize complex structures into a manageable scale. The backend was developed on the Express.js framework similarly because of its popularity in backend development and compatibility with other JavaScript frameworks, such as Bcrypt and Crypto, which are used extensively for this study.

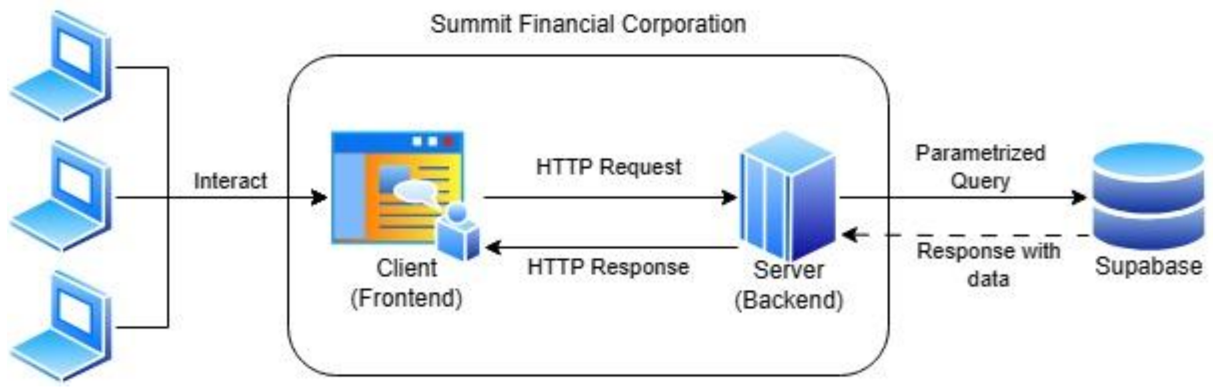


Figure 1. Summit Financial Corporation web architectural diagram

In order to build and deploy the microservices efficiently and consistently in the local host, we dockerized the microservices by creating docker containers for each microservice and executed the containers simultaneously using Docker-compose, a Docker tool used for running multiple containers at once with predefined configurations. The local host ports were assigned to the frontend and backend to enable communication between the two microservices.

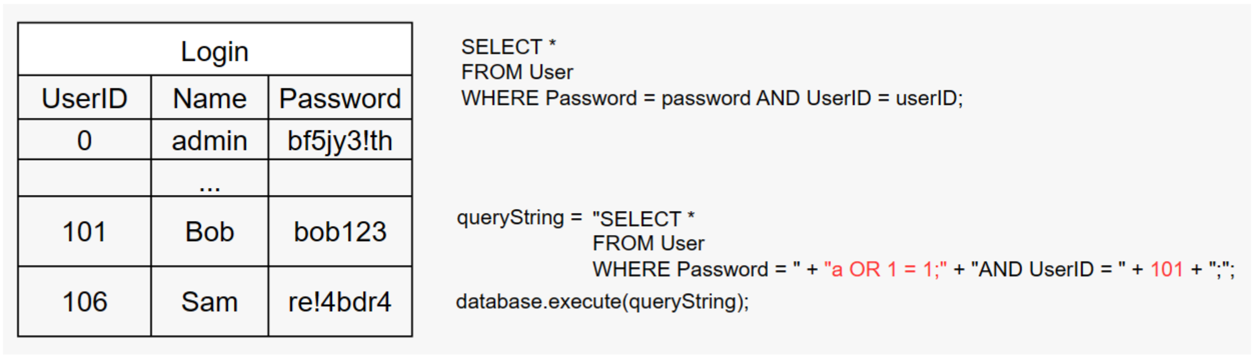
The choice of database management system for SFC was Supabase, an open-source PostgreSQL relational database management system. Supabase was ideal for our banking web application for several security enhancement reasons. Supabase provides row-level security with easy-to-serve policies, parameterized query functions for JavaScript API, and a cloud-hosted database all under a free plan. The details of the specific effects of these security features will be discussed in the following section. Using the supabase-js library, the backend server could query data from the Supabase database, and this added layer between the server and database serves another security enhancement purpose.

Security Features

The three main security features that are discussed in this section are password hashing, query parameterization, and session-based user authentication. Query parameterization provides primary protection from a data breach in a database that exploits SQL queries. User authentication provides protection against unauthorized API accesses that attempt to extract data by making direct API requests, which users should not have access to. Lastly, password hashing provides protection to the user credentials in case of a database breach or unauthorized API access that successfully extracts the user passwords. These security features in combination ensure the protection of user data through the layer of defense approach.

Query Parameterization

SQL injection is the greatest threat to relational databases that utilize SQL to query data. With poor application and database designs, users may query any data from the database without proper access privileges. For example, as shown in Figure 2, a simple query parameter “password” could shortcircuit the query condition by including the “a OR 1 = 1” phrase in the conditional statement and query the entire Login table without actual credentials. Consequently, an attacker may exploit this vulnerability to log in as an admin and extract high-profile data or implant malware, leading to a secondary data breach.



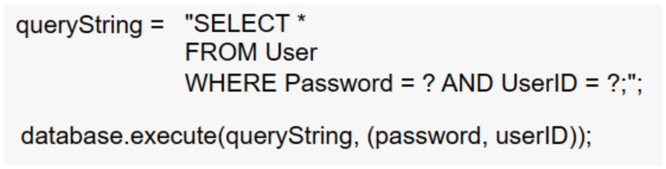
Login		
UserID	Name	Password
0	admin	bf5jy3!th
	...	
101	Bob	bob123
106	Sam	re!4bdr4

```
SELECT *
FROM User
WHERE Password = password AND UserID = userID;


queryString = "SELECT *
               FROM User
               WHERE Password = " + "a OR 1 = 1;" + "AND UserID = " + 101 + ";";
database.execute(queryString);
```

Figure 2. Query without Parameterization

Query parameterization is one of the solutions to SQL injection. It is the process of passing user input as designated parameters rather than a portion of the query string. The database management system recognizes the parameters as singular values and thus rejects any query strings like “a OR 1 = 1”, preventing SQL injections. Figure 3 shows a parameterized query as an alternative to the query shown in Figure 2.

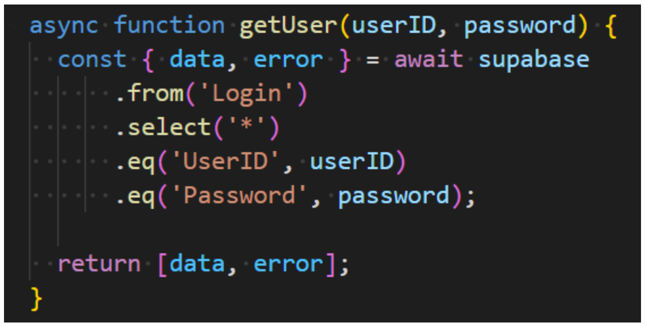


```
queryString =   "SELECT *
                FROM User
                WHERE Password = ? AND UserID = ?;";

 database.execute(queryString, (password, userID));
```

Figure 3. Query with Parameterization

The SFC web application operates a Supabase database using the JavaScript library supabase-javascript. The query function of Supabase client is modularized by the operations as shown in Figure 9. Rather than executing a complete PostgreSQL query string, individual methods are called with user inputs and executed in the Supabase backend. The modularization of queries reduces the complexity of queries for general users, and it also provides query parameterization to prevent SQL injections. The example below (Figure 4) is the Supabase query for finding a user with the corresponding UserID and Password.



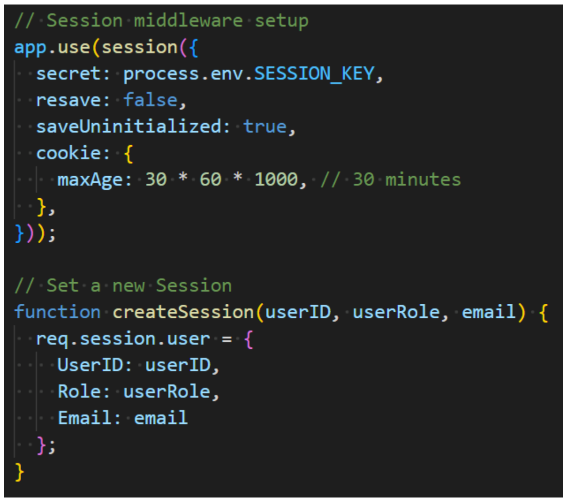
```
async function getUser(userID, password) {
  const { data, error } = await supabase
        .from('Login')
        .select('*')
        .eq('UserID', userID)
        .eq('Password', password);

  return [data, error];
}
```

Figure 4. Example query from SFC web application

User Authentication

Upon the initial login, the user's credentials must persist in the system until the user logs out of his or her account. Two common methods of preserving user credentials in web applications exist, which are session and token. A session is a set duration of time that the server preserves the user credential in the form of a cookie, and the token is a data package that the client service provides in every request made by the client. Both methods provide secure authentication, but the SFC web application utilizes the Express.js library's session to maintain user credentials and provide authentication because session-based authentication is advantageous for easy access to state contents and configuring timeout time for dropping credentials.

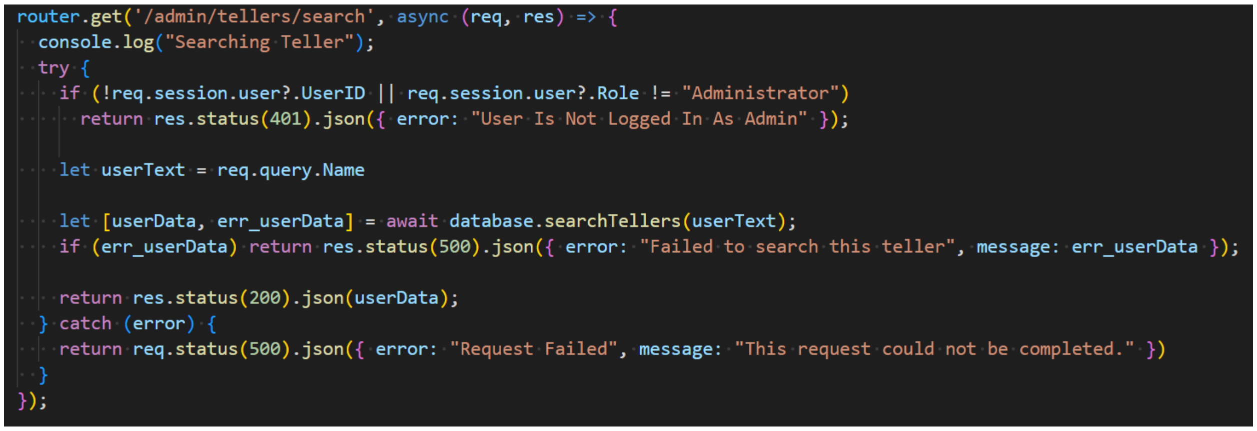


```
// Session middleware setup
app.use(session({
  secret: process.env.SESSION_KEY,
  resave: false,
  saveUninitialized: true,
  cookie: {
    maxAge: 30 * 60 * 1000, // 30 minutes
  },
}));

// Set a new Session
function createSession(userID, userRole, email) {
  req.session.user = {
    UserID: userID,
    Role: userRole,
    Email: email
  };
}
```

Figure 5. SFC session creation

Figure 5 shows the process of creating a new session for a user. The system first defines the session middleware to establish the session configuration. The “secret” option assigns the session key that the middleware uses to verify the authenticity of clients’ cookies. If the secret key does not match, the client session is invalidated. The “resave” option forces sessions to save modified session data after each request is completed. The “saveUninitialized” option creates a new session regardless of whether the user properly logged in or not to establish cookies available. “cookie” option sets the timeout time of the session to prevent the user from staying logged in indefinitely. Each session contains the UserID, Role, and Email to maintain the context of the user credential in the server.



```
router.get('/admin/tellers/search', async (req, res) => {
  console.log("Searching Teller");
  try {
    if (!req.session.user?.UserID || req.session.user?.Role != "Administrator")
      return res.status(401).json({ error: "User Is Not Logged In As Admin" });

    let userText = req.query.Name

    let [userData, err_userData] = await database.searchTellers(userText);
    if (err_userData) return res.status(500).json({ error: "Failed to search this teller", message: err_userData });

    return res.status(200).json(userData);
  } catch (error) {
    return req.status(500).json({ error: "Request Failed", message: "This request could not be completed." })
  }
});
```

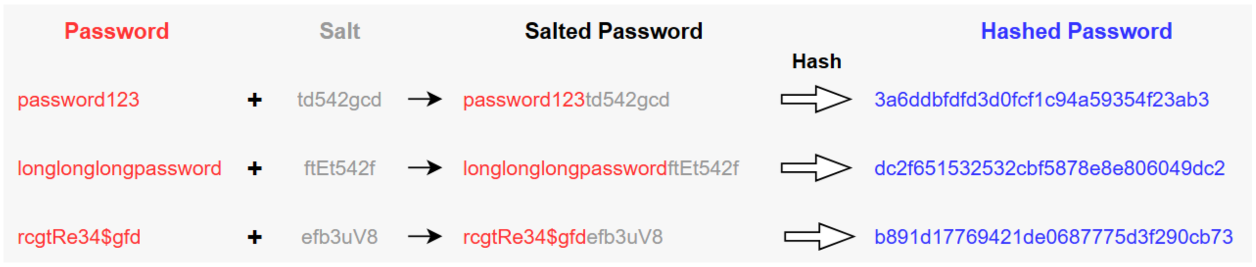
Figure 6. SFC teller search request for Administrator

Every role-specific request, such as the Administrator’s request to search for tellers by their names (Figure 6), checks the session data to verify whether the UserID is not null, and the Role matches the request access. Failure to pass this check indicates that the request was made by a user who has not properly logged in with the correct access privilege. This method prevents regular customers from accessing other customers’ information and making transactions on their behalf. Session-based authentication ensures that log requests are made by the proper users.

Password Hashing

Hashing is the process of using a hash algorithm that converts a string of characters into another string. In cryptography, hashing is a method of encoding data similar to encryption. Sensitive data like user passwords can be hashed and stored in the database to prevent secondary data breaches in case of a database breach. If an attacker gains access to the user login table, the attacker cannot immediately log in as the administrator or users to extract data without performing a form of a brute force method like collision attack and rainbow table attack.

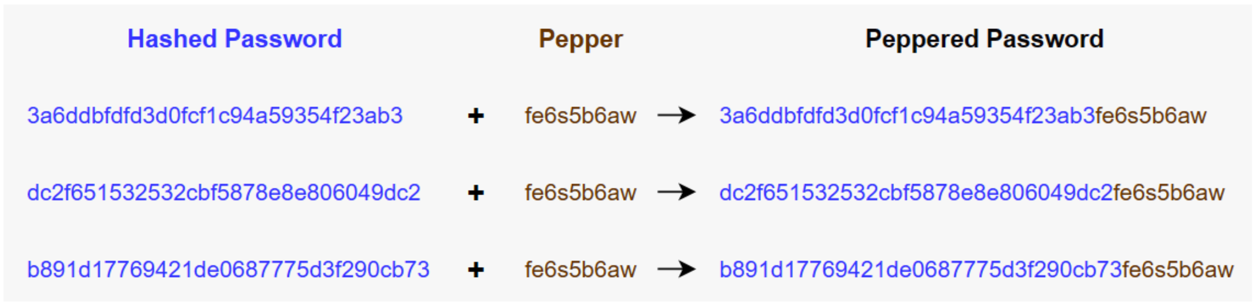
According to LastPass's report on password security, The Password Exposé, 81% of password breaches are due to weak, reused, or stolen passwords, and 41 billion passwords were breached in 2016 alone. [3] This absurd number is possible in part because password protection from hashing heavily relies on the complexity of the input string. Password cracking involves making numerous login attempts with random input values to create a rainbow table or a list of hashed values to trace the desired hashed value. Short and simple or commonly used passwords can be cracked easily using the pre-generated rainbow table, which is the main reason for the advocates of strong passwords.



Password		Salt		Salted Password	Hash	Hashed Password
password123	+	td542gcd	→	password123td542gcd	⇨	3a6ddbfdfd3d0fcf1c94a59354f23ab3
longlonglongpassword	+	ftEt542f	→	longlonglongpasswordftEt542f	⇨	dc2f651532532cbf5878e8e806049dc2
rcgtRe34$gfd	+	efb3uV8	→	rcgtRe34$gfdefb3uV8	⇨	b891d17769421de0687775d3f290cb73

Figure 7. Example salt adding process

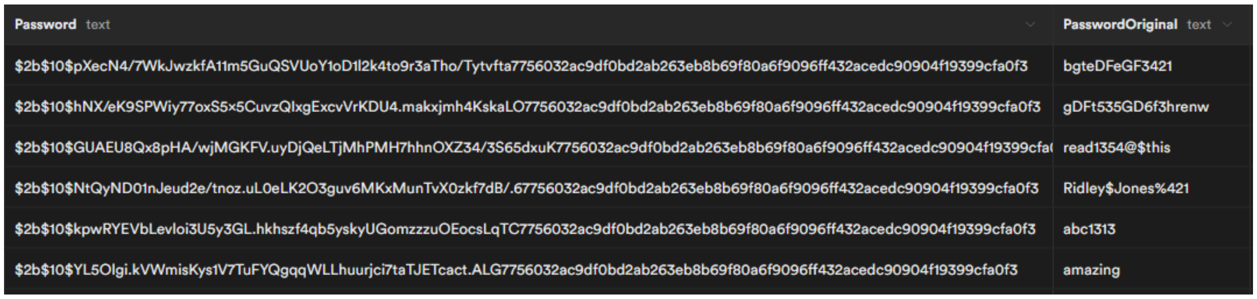
Salt and pepper are methods that provide enhanced password protection to hashing without users' involvement like intentionally creating a long and complicated password. Salt and pepper are random strings that are attached to the hashing value to increase the volume and complexity of the value when hashing. Salt is added before hashing, and it is uniquely generated for each new password. Pepper is added after hashing, and it is stored statically in the application and treated as a secret key. Figures 7 and 8 show the simplified process of adding salt and pepper during hashing.



Hashed Password		Pepper		Peppered Password
3a6ddbfdfd3d0fcf1c94a59354f23ab3	+	fe6s5b6aw	→	3a6ddbfdfd3d0fcf1c94a59354f23ab3fe6s5b6aw
dc2f651532532cbf5878e8e806049dc2	+	fe6s5b6aw	→	dc2f651532532cbf5878e8e806049dc2fe6s5b6aw
b891d17769421de0687775d3f290cb73	+	fe6s5b6aw	→	b891d17769421de0687775d3f290cb73fe6s5b6aw

Figure 8. Example pepper adding process

For the SFC web application, the JavaScript library Bcrypt is used to generate salt values and hash salted passwords. The built-in saltGen(saltRound) function produces a salt for the given amount of saltRound, which defines the duration of the function to generate a more complex salt. Then, the hash(value, salt) function hashes the password with the unique salt for the password. For the pepper, another JavaScript library Crypto is used to generate a random hexadecimal value with a length of 65 characters; the value is stored as a hidden environment variable in the system. The pepper is added to the hashed password, and the peppered password is stored in the database to complete the password hashing. As a result of hashing, a plaintext password is converted to a 125-character text as shown in Figure 9. PasswordOriginal represents the actual password entered by the users and Password represents the hashed password that is accessed by the web application. The resulting passwords are considerably more difficult to crack than the user passwords.



Password text	PasswordOriginal text
$2b$10$pXecN4/7WkJwzkfA11m5GuQSVUoY1oD1l2k4to9r3aTho/Tytvfta7756032ac9df0bd2ab263eb8b69f80a6f9096ff432acedc90904f19399cfa0f3	bgteDFeGF3421
$2b$10$hNX/eK9SPWiy77oxS5x5CuvzQIxgExcvVrKDU4.makxjmh4KskaLO7756032ac9df0bd2ab263eb8b69f80a6f9096ff432acedc90904f19399cfa0f3	gDFt535GD6f3hrenw
$2b$10$GUAEU8Qx8pHA/wjMGKFV.uyDjQeLTjMhPMH7hhnOXZ34/3S65dxuK7756032ac9df0bd2ab263eb8b69f80a6f9096ff432acedc90904f19399cfa	read1354@$this
$2b$10$NtQyND01nJeud2e/tnoz.uL0eLK2O3guv6MKxMunTvX0zkf7dB/.67756032ac9df0bd2ab263eb8b69f80a6f9096ff432acedc90904f19399cfa0f3	Ridley$Jones%421
$2b$10$kpwRYEVbLevloi3U5y3GL.hkhszf4qb5yskyUGomzzzuOEocsLqTC7756032ac9df0bd2ab263eb8b69f80a6f9096ff432acedc90904f19399cfa0f3	abc1313
$2b$10$YL5Olgi.kVWmisKys1V7TuFYQgqqWLLhuurjci7taTJETcact.ALG7756032ac9df0bd2ab263eb8b69f80a6f9096ff432acedc90904f19399cfa0f3	amazing

Figure 9. Comparison of hashed password and original password

Conclusion

The migration of consumer services to online space has achieved a great enhancement to the utility and accessibility of those services to the masses. However, with better utility and accessibility came the threats of malicious entities searching for vulnerabilities to commit digital robbery. Data breaches are a great concern to many online services such as banking as millions of dollars' worth of damage may occur for each data breach.

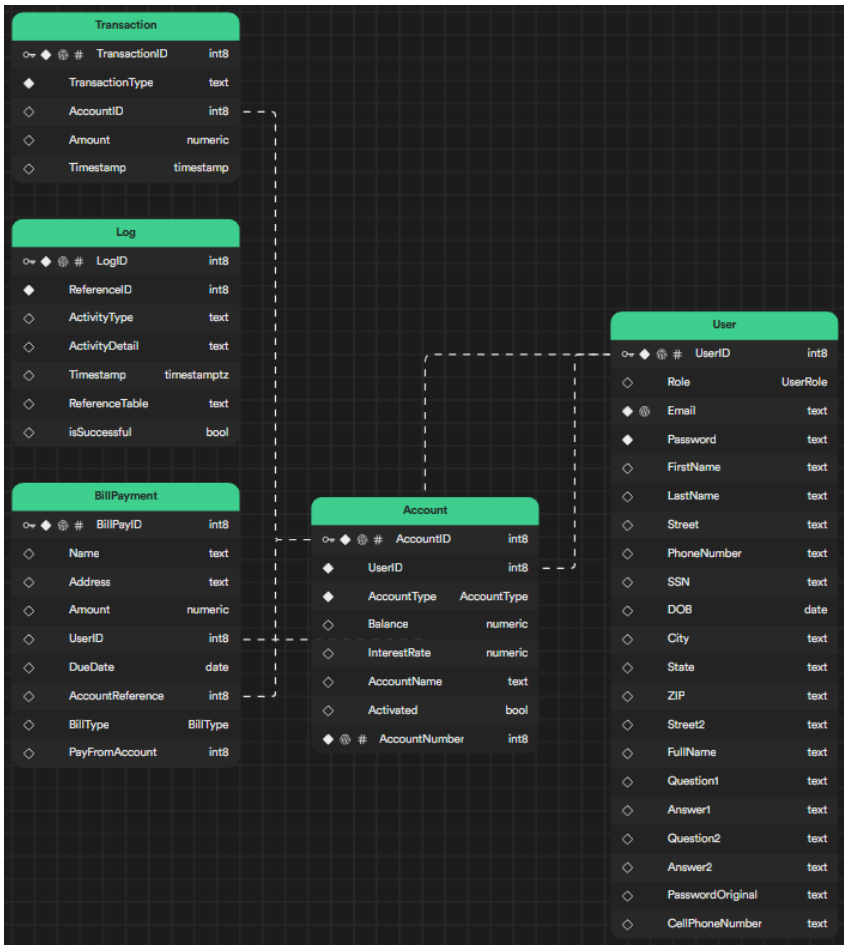
This paper discusses three security features, query parameterization, user authentication, and password hashing, which provide layers of protection from attempts of data breaches. These features were implemented and tested for the Summit Financial Corporation web application. More security features that were not mentioned in this paper were integrated into the SFC web applications, including the microservice architecture, protected routing, and cross-origin resource sharing (CORS), but for the purpose of narrow analysis, these features were not discussed. Finally, the Bank Management Utility project of CS499 Senior Project benefited from the cybersecurity perspective this paper provided as it led to the architectural decisions that ensured the implementation of reliable and secure system design for the Summit Financial Corporation website.

Appendix

The Summit Financial Corporation project repository can be found in this link.

(https://github.com/aem0424/banking499)

Appendix A. Database Diagram



Transaction	
TransactionID	int8
TransactionType	text
AccountID	int8
Amount	numeric
Timestamp	timestamp

Log	
LogID	int8
ReferenceID	int8
ActivityType	text
ActivityDetail	text
Timestamp	timestamptz
ReferenceTable	text
isSuccessful	bool

BillPayment	
BillPayID	int8
Name	text
Address	text
Amount	numeric
UserID	int8
DueDate	date
AccountReference	int8
BillType	BillType
PayFromAccount	int8

Account	
AccountID	int8
UserID	int8
AccountType	AccountType
Balance	numeric
InterestRate	numeric
AccountName	text
Activated	bool
AccountNumber	int8

User	
UserID	int8
Role	UserRole
Email	text
Password	text
FirstName	text
LastName	text
Street	text
PhoneNumber	text
SSN	text
DOB	date
City	text
State	text
ZIP	text
Street2	text
FullName	text
Question1	text
Answer1	text
Question2	text
Answer2	text
PasswordOriginal	text
CellPhoneNumber	text

Reference List

[1] “Cost of a Data Breach 2023.” *IBM*, www.ibm.com/reports/data-breach?utm_medium=OSocial&utm_source=Blog&utm_content=RSRWW&utm_id=SI-Blog-CTA-Button&_ga=2.60426413.1423812216.1700810097-2068772737.1700810097&_gl=1%2Adm720a%2A_ga%2AMjA2ODc3MjczNy4xNzAwODEwMDk3%2A_ga_FYECCCS21D%2AMTcwMDgxMDA5OS4xLjAuMTcwMDgxMDA5OS4wLjAuMA.. Accessed 27 Nov. 2023.

[2] Technologies, America One. “Information Security: What Is the Silver Bullet? - Braintree, Boston, Cape Cod.” *America One Technologies, Inc.*, 15 Nov. 2022, www.americatech.com/2022/10/information-security-what-is-the-silver-bullet/.

[3] *The Password Exposé - Lp-Cdn.Lastpass.Com*, lp-cdn.lastpass.com/lporcamedia/document-library/lastpass/pdf/en/LastPass-Enterprise-The-Password-Expose-Ebook-v2.pdf. Accessed 27 Nov. 2023.