SWE 681

Wordle

Team Members/Collaborators
Harsh Mehta
Simriti Koul

05/12/2022

Table of Contents

1. Introduction

2. Design

3. Architecture of the code
    3.1 Front end code architecture
    3.2 Back end code architecture
4. Installation and Operating Instructions
    4.1 Backend setup
    4.2 Frontend setup
5. Game Rules
6. Assurance Cases

1. Introduction

Wordle is a desktop web based application game which can be played by two players. Player who creates a game will automatically become the first player. After creating a game, the first player will share a code to another player via email to join the game. The player joining the game will become the second player. Each player will get three turns to guess a five letter word. At the end of the game the player who made the most letter guess correctly will win, else the game will get a draw.

Note: Before creating or joining the game, players will have to create an account and then login to play the game.

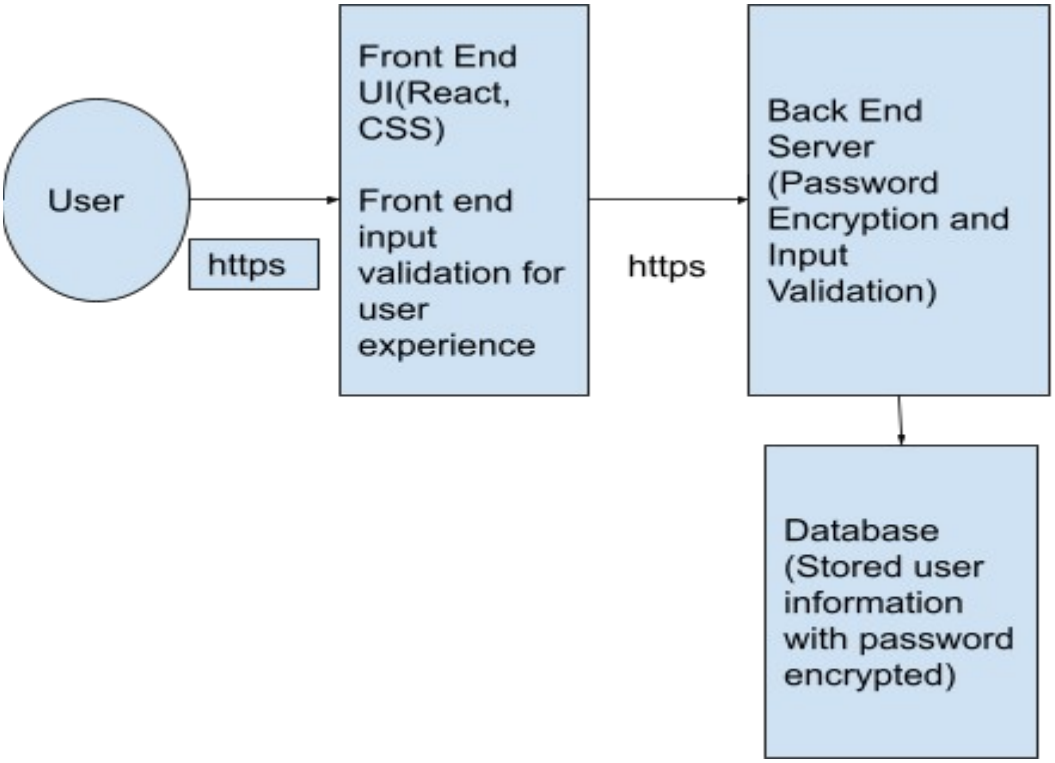
Source Code Repository

Front end source code Link:
https://github.com/hmehta8/fronendwordle/tree/main/LatestWordle-main

Back end source code Link:
https://github.com/hmehta8/springboot-public/tree/main/springbootwordle-main

Application is available at website listed below
https://multiplayerwordle.herokuapp.com/

2. Design


User
https
Front End UI(React, CSS)
Front end input validation for user experience
https
Back End Server (Password Encryption and Input Validation)
Database (Stored user information with password encrypted)

3. Architecture of the Code

3.1 Front End Code Architecture

```
onPressSignUp = () => {

    let heading = "Input Invalid !!"
    let str = "Please enter the valid input for the boxes highlighed in red."
    let value = false;

    axios.post('https://twoplayerwordle.herokuapp.com/api/signup/create-user', this.state)
    .then(response => {
      console.log(response.data)
      this.setState({
        firstNameValid: response.data.body.firstNameValid,
        lastNameValid: response.data.body.lastNameValid,
        emailValid: response.data.body.emailValid,
        passwordValid: response.data.body.passwordValid,
        repeatPasswordValid: response.data.body.repeatPasswordValid,
        userExits: response.data.body.userExits
      })

      if(this.state.firstNameValid === false || this.state.lastNameValid === false ||
         this.state.emailValid === false || this.state.passwordValid === false ||
         this.state.repeatPasswordValid === false ) {

          Swal.fire({
            position: "top",
            allowOutsideClick: false,
            title: heading,
            html: "<pre>" + str + "</pre>",
            width: 805,
            padding: "0.7em",
            // Custom CSS
            customClass: {
              heightAuto: false,
              title: "title-class",
              popup: "popup-class",
              confirmButton: "button-class",
            },
          });
```

The above function will sign up users and validate user input for user experience. This is just code snippet for full logic please review code in the code repository.

```
onPressLogIn = () => {

   let heading = "Bad Credentials !!"
   let str = "Credentials you entered are not correct"

   axios.post('https://twoplayerwordle.herokuapp.com/api/login/login-user', this.state)
   .then(response => {

     this.setState({
       loginPasswordValid: response.data.body.loginPasswordValid,
       loginEmailValid: response.data.body.loginEmailValid,
       loginSuccess: response.data.body.loginSuccess
     })

     if(this.state.loginPasswordValid === false || this.state.loginEmailValid === false|| this.state.loginSuccess === false) {
       Swal.fire({
         position: "top",
         allowOutsideClick: false,
         title: heading,
         html: "<pre>" + str + "</pre>",
         width: 805,
         padding: "0.7em",
         // Custom CSS
         customClass: {
           heightAuto: false,
           title: "title-class",
           popup: "popup-class",
           confirmButton: "button-class",
         },
       }).then(result => {
         window.location.href = "/";
```

The above function will validate user input and let users continue playing games. This is just a code snippet for full logic please review code in the code repository.

3.2 Back end Code Architecture

```
public UserDetails addUser(UserDetails user) throws MessagingException {

    boolean validateUserFirstName = validateFirstName(user.getFirstName());
    boolean validateUserLastName = validateLastName(user.getLastName());
    boolean validateEmail = validateEmail(user.getEmail());
    boolean validatePassword = validatePassword(user.getPassword());
    boolean validateRepeatPassword = validateRepeatPassword(user.getRepeatPassword(), user.getPassw

    user.setFirstNameValid(validateUserFirstName);
    user.setLastNameValid(validateUserLastName);
    user.setEmailValid(validateEmail);
    user.setPasswordValid(validatePassword);
    user.setRepeatPasswordValid(validateRepeatPassword);

    saveUser(user);

    return user;
}

public void saveUser(UserDetails user) throws MessagingException {

    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    if (user.isFirstNameValid() && user.isLastNameValid() && user.isEmailValid() &&
        user.isPasswordValid() && user.isRepeatPasswordValid()) {

        String firstName = user.getFirstName();
        String lastName = user.getLastName();
        String email = user.getEmail();
        String encryptedPassword = passwordEncoder.encode(user.getPassword());

        SignUpEntity signUpUser = new SignUpEntity(firstName, lastName, email, encryptedPassword);

        if(!checkUserExists(email)) {
            System.out.println(user);
```

The above code snippet will validate user input in the backend, encrypt user password and then save user to the database. This is just a code snippet for full logic please review code in the code repository.

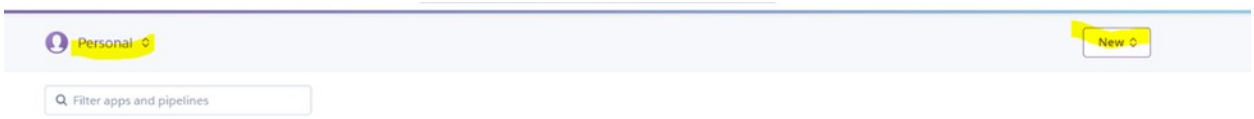
```java
public LoginDetails validateUser(LoginDetails user) {

    boolean validateEmail = validateEmail(user.getLoginEmail());
    boolean validatePassword = validatePassword(user.getLoginPassword());

    user.setLoginEmailValid(validateEmail);
    user.setLoginPasswordValid(validatePassword);

    if(validateEmail == false || validatePassword == false) {
        return user;
    }

    if(validateUserCredentials(user)) {
        user.setLoginSuccess(true);
        user.setLoginPassword(null);
    }

    return user;

}

public boolean validateUserCredentials(LoginDetails user) {

    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    List<SignUpEntity> userDetail = signUpRepo.findByEmail(user.getLoginEmail());

    if(userDetail.size() > 0) {
        String password = user.getLoginPassword();
        if(passwordEncoder.matches(password, userDetail.get(0).getPassword())) {
            return true;
        } else {
            return false;
```

The above code snippet will validate user credentials in the backend before they log in. This is just a code snippet for full logic please review code in the code repository.

4. Installation and Operating Instructions

4.1 Backend setup :

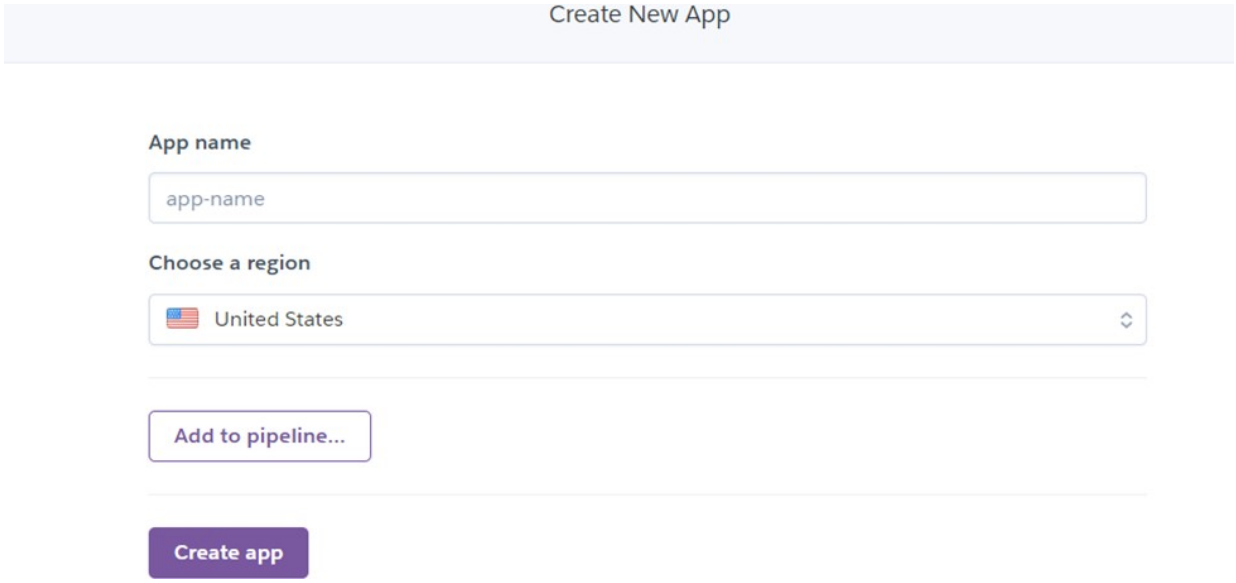
1) Download code from the link shown at the bottom of this document



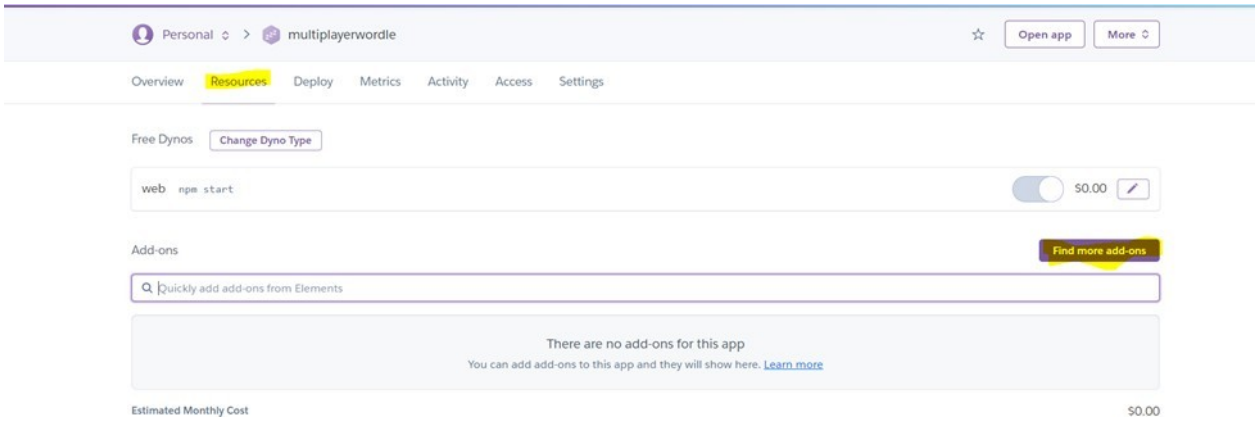
2) Create account on Heroku

3) Go to personal and click on new

4) Fill up the information shown below

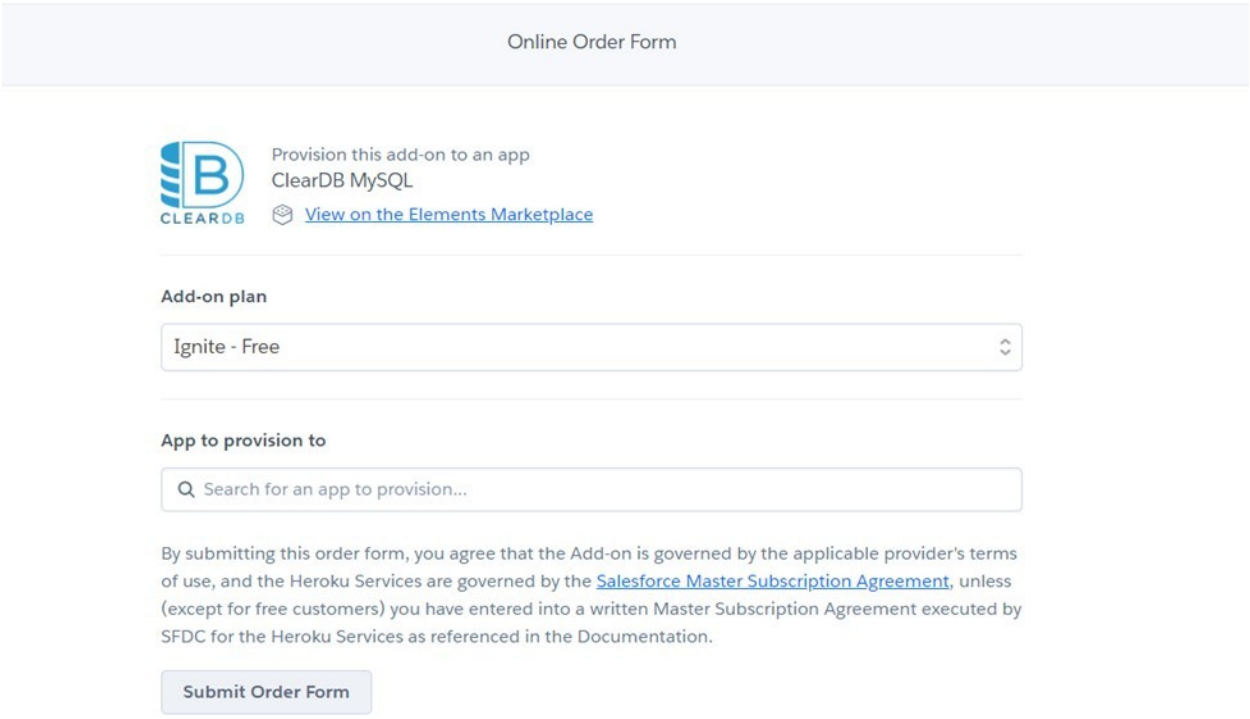


5) Go to the resources tab and click on more addons



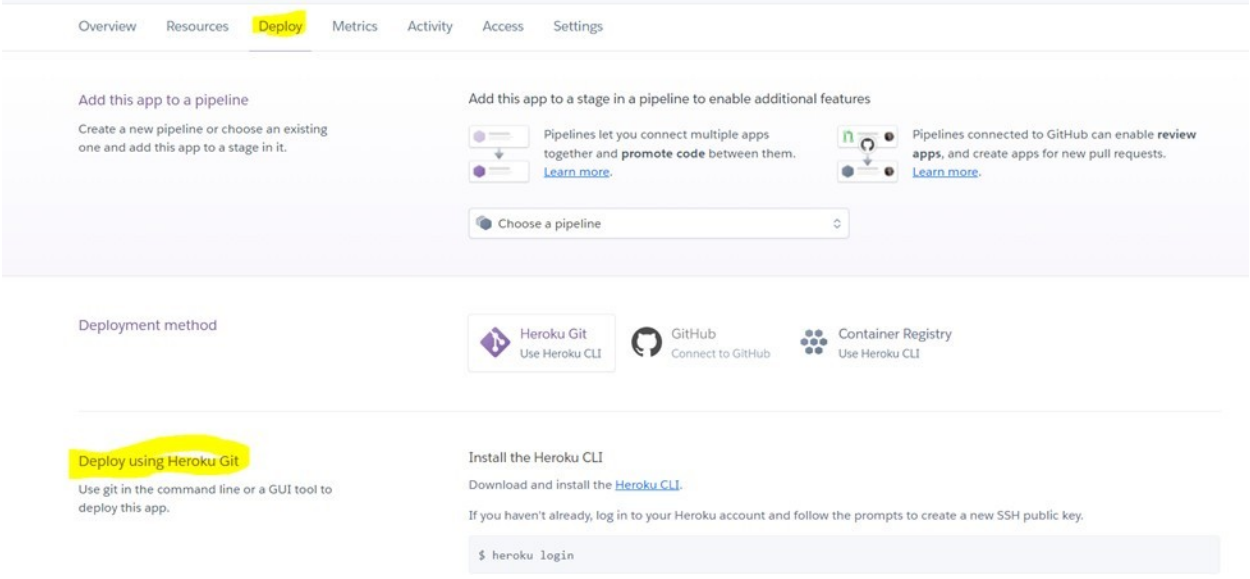
6) Go to clear db and then click on install

7) Fill up the information shown below. In the app to provision add the name of the add which user created in step 3 and click on submit order form



8) Upon successful installation user will find clearDB MYSQL in resources tab shown in step 5

9) Go to the Deploy tab shown below and follow the deployment steps shown in deploy using Heroku git



10) Replace db credentials and email credentials in the application.properties file based on admin set up during installation of clear db mysql

11) Replace the yellow highlighted line shown below with front end url which admin/user created during front end set up to avoid CORS issue

```
@RestController
@CrossOrigin(origins = "https://multiplayerwordle.herokuapp.com/")
@RequestMapping(value = "/api/login")
public class LoginController {

    @Autowired
    LoginService loginUserDetails;
```

4.2 Front End Setup

1) Creation of app and deployment step is same as back end set up
2) Replace post/get request url with backend url which user created while setting up back end

5. Game Rules

1) Each player get three turns to get the word
2) If letter exists in the word and position is correct then it will be highlighted in green
3) At the end of the game player who made maximum correct guess will win
4) If none of the players made correct guess the game will get draw

6. Assurance Cases

1) **Buffer Overflow attack**: Buffer overflow occurs when a program or process attempts to write more data to a fixed-length block of memory, than the buffer is allocated to hold. To counter buffer overflow attack software developers must perform input validation and memory safe programming language.

   Thus to make our wordle application secure from buffer overflow attack we have implemented input validation when users sign up, login to the application and at time of players move. Moreover, we have used Java programming language which is considered to be a memory safe programming language to counter buffer overflow attack. Below is the back end code snippet of wordle application which performs input validation.

```
public boolean validateEmail(String email) {
    String regex = "[a-z0-9._%+-]{1,50}@[a-z0-9]{1,50}[.][a-z]{2,3}$";

    if (email == null) {
        return false;
    }

    // Compile the ReGex
    Pattern p = Pattern.compile(regex);

    Matcher m = p.matcher(email);

    return m.matches();
}
```

   Note: This is just a code snippet for full code please check code repository

2) **Brute Force Attack**: A brute force attack happens when a hacker attempts to guess user login credentials manually or by using software. To counter brute force attack software developers must enforce a strong password policy and encrypt the password before saving it to the database. In our application we force users to create a strong password via combination of upper and lower case, Minimum 8 characters and Maximum 20 characters and mixture of letters and numbers and if the password matches this criteria we encrypt the password before saving it to the database. Below is the back end code snippet of wordle application which will force the user to create a strong password and encrypt the password before saving it to the database.

```
public boolean validatePassword(String password) {
    String regex = "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z]).{8,20}$";

    if(password == null || password.equals("")) {
        return false;
    }

    if(checkForWhiteSpace(password)) {
        return false;
    }

    // Compile the ReGex
    Pattern p = Pattern.compile(regex);


    Matcher m = p.matcher(password);


    return m.matches();
}
```

```
public void saveUser(UserDetails user) throws MessagingException {

        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

        if (user.isFirstNameValid() && user.isLastNameValid() && user.isEmailValid() &&
                user.isPasswordValid() && user.isRepeatPasswordValid()) {

                String firstName = user.getFirstName();
                String lastName = user.getLastName();
                String email = user.getEmail();
                String encryptedPassword = passwordEncoder.encode(user.getPassword());

                SignUpEntity signUpUser = new SignUpEntity(firstName, lastName, email, encryptedPassword);
```

3) **Keylogger Attack:** The attacker installs a program that captures every keystroke on the user's computer, including site visit, usernames, password, answers to security questions and more. To counter this kind of attack software developers must implement a multifactor authentication method. We have made our wordle application safe from keylogger attack by implementing a multi factor authentication method which sends a security code via users registered email address.
4) **Plain Text Storage**: Developers often hard code database credentials and push the code on public repositories like GitHub. When such credentials are exploited hackers can use the same login credentials to perform brute force or DDoS attack. To counter this kind of vulnerability database/user credentials must be passed during the build. In our application we are to counter this vulnerability. We pass credentials at the time of build which is just known to the admin. Below is a code snippet from the wordle back end application. All parameters in the code snippet will get replaced with legitimate credentials at the time of build.

```
database.url=jdbc:mysql://135.148.162.205:3306/wordle
database.username=${DB_USERNAME}
database.password=${DB_PASSWORD}
#spring.jpa.hibernate.ddl-auto=update
database.driver =com.mysql.jdbc.Driver
database.minimumIdle=1
database.maximumPoolSize=25
database.maxLifeTime=30000

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=${EMAIL}
spring.mail.password=${EMAIL_PASSWORD}
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

email=${EMAIL}
```

5) **Cross domain attacks:** Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy . However, it also provides potential for cross-domain attacks, if a website's CORS policy is poorly configured and implemented. To counter this kind of attack CORS policy should only accept requests such as POST/GET/PUT from the trusted website. In our wordle application we only accept post/get requests from the trusted frontend source. Below is the code snippet from the wordle back end application which counters cross domain attack

```
@RestController
@CrossOrigin(origins = "https://multiplayerwordle.herokuapp.com/")
@RequestMapping(value = "/api/login")
public class LoginController {

    @Autowired
    LoginService loginUserDetails;

    @GetMapping(value="deployed")
    public String welcomeMessage() {
        return "Deployed Successfully";
    }
```

6) **HTTP vulnerability**: HTTP is not secure, when exploited hackers can perform attacks such as SQL Injection, Cross Site Scripting, Broken authentication and session management, and Cross Site Request Forgery. To counter this kind of attacks software developers must deploy their application over HTTPS. To counter this kind of attacks mentioned above we have deployed our wordle application over HTTPS.

Source:

1) https://www.techtarget.com/searchsecurity/definition/buffer-overflow
2) Source code from the wordle application
3) https://www.fortinet.com/resources/cyberglossary/brute-force-attack
4) https://its.ucsc.edu/mfa/cyber-attacks.html
5) https://www.linkedin.com/pulse/storing-database-credentials-securely-siddhesh-jog
6) https://www.beyondtrust.com/resources/glossary/hardcoded-embedded-passwords#:~:text=Hardcoded%20Passwords%2C%20also%20often%20referred,into%20the%20source%20code.
7) https://portswigger.net/web-security/cors
8) https://www.purevpn.com/ddos/http-vulnerability