**Лабораторна робота №3**

**Типові схеми ідентифікації та аутентифікації**

Мета: отримати представлення про типові схеми ідентифікації та аутентифікації, набути практичних навичок створення програм для ідентифікації та аутентифікації з використанням Playfair Cipher.

Завдання:

Написати програму для створення об’єкту-еталону ідентифікації та аутентифікації користувача. Для шифрування пароля використовувати Playfair Cipher. Розмір матриці шифрування 7х5. У якості алфавіту використовувати український алфавіт та 3 додаткові символи: ’ . \_. У якості символа для відокремлення використовувати ’.

Логін: пєтухова; Пароль: волкоф; Зашифрований пароль: авілгм

Хід роботи

1. Аналіз

Реалізувати шифр Плайфеера можна напряму двома функціями (з використанням спільних додаткових функцій). Але якщо спробувати зашифрувати щось «на папері» то обов’язково буде листок з матрицею символів.

Тому, дуже зручно виділити класс PlayfairGrid який буде інкапсулювати все що пов’язанно с сіткою символів.

Таким чином у функції шифрування та дешифрування потрібно передавати не ключове слово, алфавіт тощо, а лише екземпляр PlayfairGrid.

Крім того, ми можемо створити фабрику и дуже просто створювати сітку для різних мов та алфавітів (див. «результат виконання програми»).

1. Текст програми

Тому функції шифрування та дешифрування досить прості:

import PlayfairGrid from "./PlayfairGrid";

import chunk from "lodash/chunk";

export function encode(msg: string, grid: PlayfairGrid): string {

   return prepareMsg(msg, grid.alphabet)

      .map(pair => grid.encodePair(pair)).flatMap(s => s).join('');

}

export function decode(msg: string, grid: PlayfairGrid): string {

   return prepareMsg(msg, grid.alphabet)

      .map(pair => grid.decodePair(pair)).flatMap(s => s).join('');

}

function prepareMsg(msg: string, alphabet: string[]): string[][] {

   msg = msg.toLowerCase().replace(/\s/g, '')

      .replace(new RegExp(`[^${alphabet.join('')}]`, 'g'), alphabet[0]);

   return chunk(msg.split(''), 2);

}

PlayfairGrid також достатньо прозорий:

import chunk from 'lodash/chunk';

type GridMap = Map<string, { i: number, j: number }>

export default class PlayfairGrid {

   private \_grid: string[][];

   private \_gridMap: GridMap = new Map();

   constructor(

      private \_keyword: string,

      private \_alphabet: string[],

      private \_delimiter: string,

      private \_width: number,

   ) {

      this.create();

   }

   public static createEnGrid(keyword: string) {

      const alphabet = "abcdefghijklmnopqrstuvwxyz'.,\_".split('');

      return new PlayfairGrid(keyword, alphabet, "'", 5);

   }

   public static createUaGrid(keyword: string) {

      const alphabet = "абвгдеєжзиіїйклмнопрстуфхцчшщьюя'.\_".split('');

      return new PlayfairGrid(keyword, alphabet, "'", 5);

   }

   public encodePair(pair: string[]): string[] {

      return this.processPair(pair, 'encode');

   }

   public decodePair(pair: string[]): string[] {

      return this.processPair(pair, 'decode');

   }

   public get alphabet() {

      return this.\_alphabet.slice();

   }

   private create() {

      this.\_grid = chunk(this.prepareAlphabet(this.\_keyword, this.\_alphabet), this.\_width);

      this.\_gridMap = this.createGridMap(this.\_grid);

   }

   private processPair([a, b]: string[], mode: 'encode' | 'decode'): string[] {

      const shift = (mode === 'encode') ? 1 : -1;

      if (b === a || !b) b = this.\_delimiter;

      const { i: ai, j: aj } = this.\_gridMap.get(a);

      const { i: bi, j: bj } = this.\_gridMap.get(b);

      if (ai === bi) {

         return this.getPair(ai, aj + shift, bi, bj + shift)

      }

      if (aj === bj) {

         return this.getPair(ai + shift, aj, bi + shift, bj);

      }

      return this.getPair(ai, bj, bi, aj);

   }

   private getPair(ai: number, aj: number, bi: number, bj: number) {

      return [this.get(ai, aj), this.get(bi, bj)];

   }

   private get(i: number, j: number): string {

      if (i < 0) i = this.\_grid.length - 1;

      else if (i === this.\_grid.length) i = 0;

      if (j < 0) j = this.\_width - 1;

      else if (j === this.\_width) j = 0;

      return this.\_grid[i][j];

   }

   private createGridMap(grid: string[][]): GridMap {

      return grid.reduce((map, row, i) => {

         row.forEach((ch, j) => map.set(ch, { i, j }));

         return map;

      }, new Map());

   }

   private prepareAlphabet(keyword: string, alphabet: string[]): string[] {

      keyword = keyword.toLowerCase();

      const codeSet = new Set(keyword);

      return [...keyword.split(''), ...alphabet.filter(ch => !codeSet.has(ch))]

   }

}

Index.ts

const playfairGrid = PlayfairGrid.createEnGrid(KEYWORD);

const users: User[] = ('users' in localStorage) ?

   JSON.parse(localStorage.users) : []

const userTemplate = template($userTmpl.innerHTML);

renderUsers();

initEvents();

function initEvents() {

   $auth.addEventListener('click', () => {

      auth();

   });

   $decode.addEventListener('click', () => {

      decodeUser();

   });

   $addForm.addEventListener('submit', (event) => {

      event.preventDefault();

      addUser();

   });

}

function auth() {

   const login = prompt('Login', '');

   const pwd = prompt('Password', '');

   const user = users.find(({ login: cLogin }) => login === cLogin);

   if (!user) {

      alert('User not found');

      return;

   }

   if (encode(pwd, playfairGrid) !== user.pwd) {

      alert('Wrong password!');

      return;

   }

   alert(`Success!\nLogin: ${login}\nAbout: ${user.about}`);

}

function decodeUser() {

   const login = prompt('Login', '');

   const user = users.find(({ login: cLogin }) => login === cLogin);

   if (!user) {

      alert('User not found');

      return;

   }

   const pwd = decode(user.pwd, playfairGrid);

   alert(`Login: ${login}\nEncoded password: ${user.pwd}\n` +

      `Decoded password: ${pwd}`);

}

function addUser() {

   const login = $login.value.trim();

   const pwd = encode($pwd.value.trim(), playfairGrid);

   const about = $about.value.trim();

   if (users.find(({ login: l }) => login === l)) {

      alert('User already exist');

      return;

   }

   saveUser({ login, pwd, about });

   renderUsers();

   $addForm.reset();

   $login.focus();

}

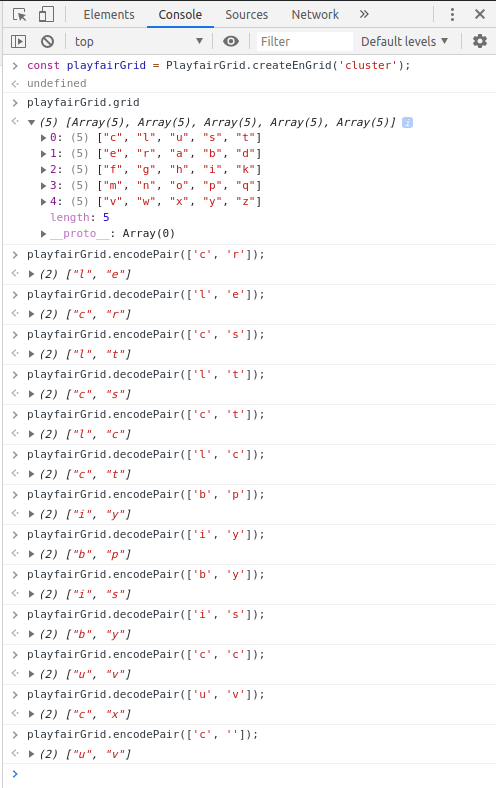
function saveUser(user: User) {

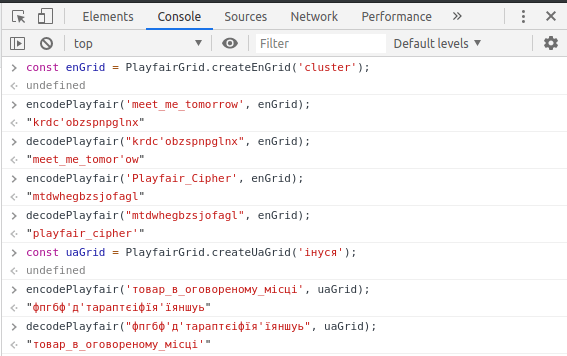
   users.push(user);

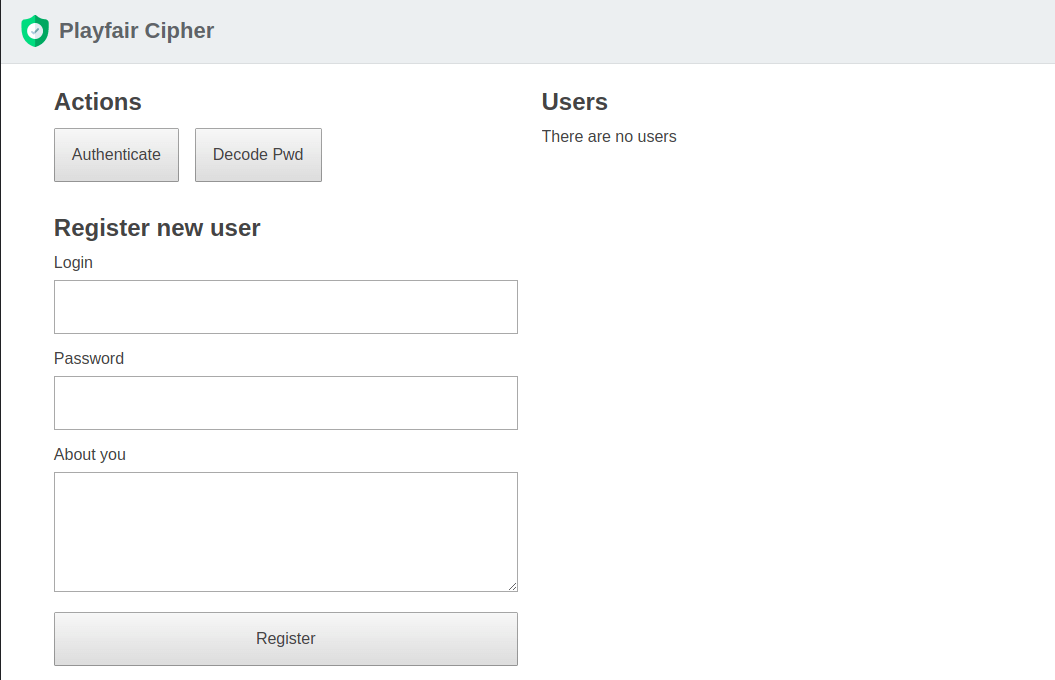
   localStorage.users = JSON.stringify(users);

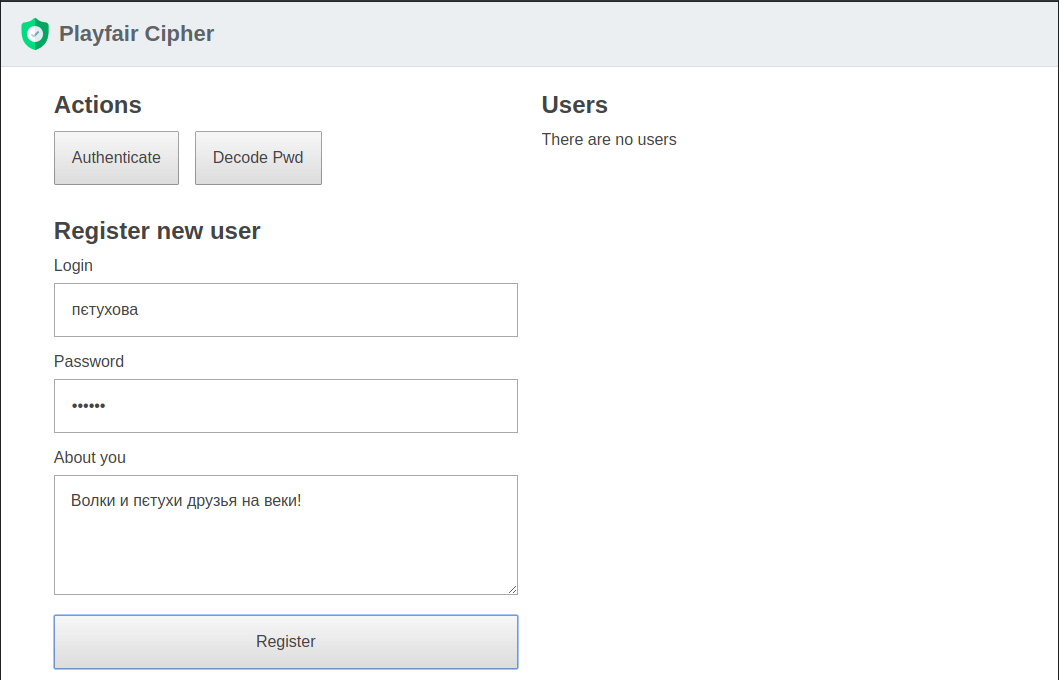
}

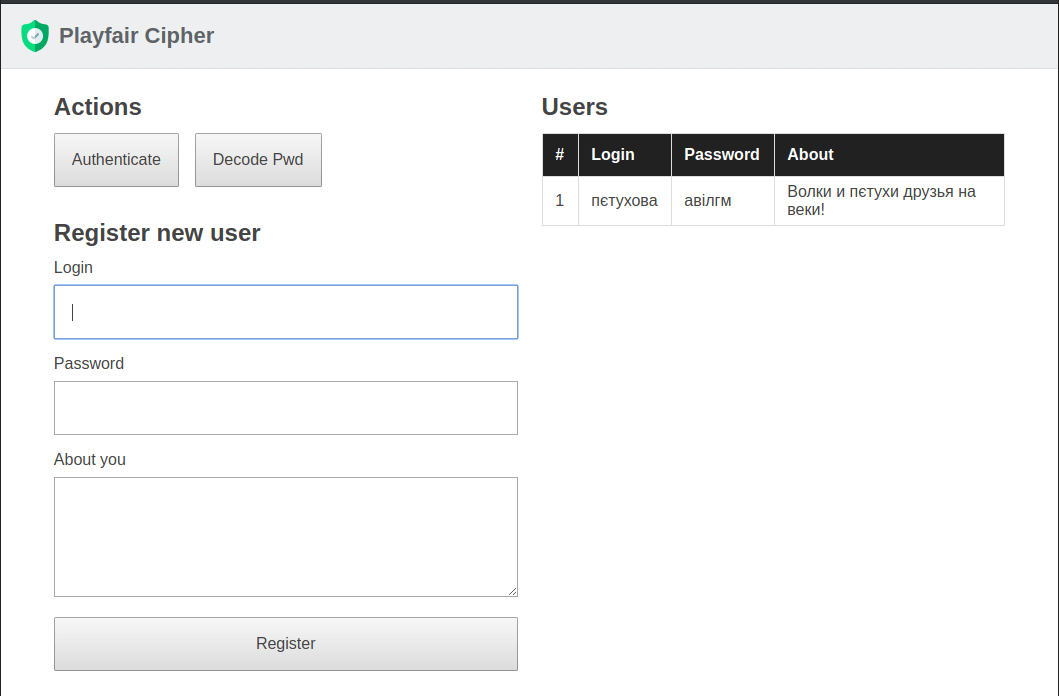
1. Результат виконання програми

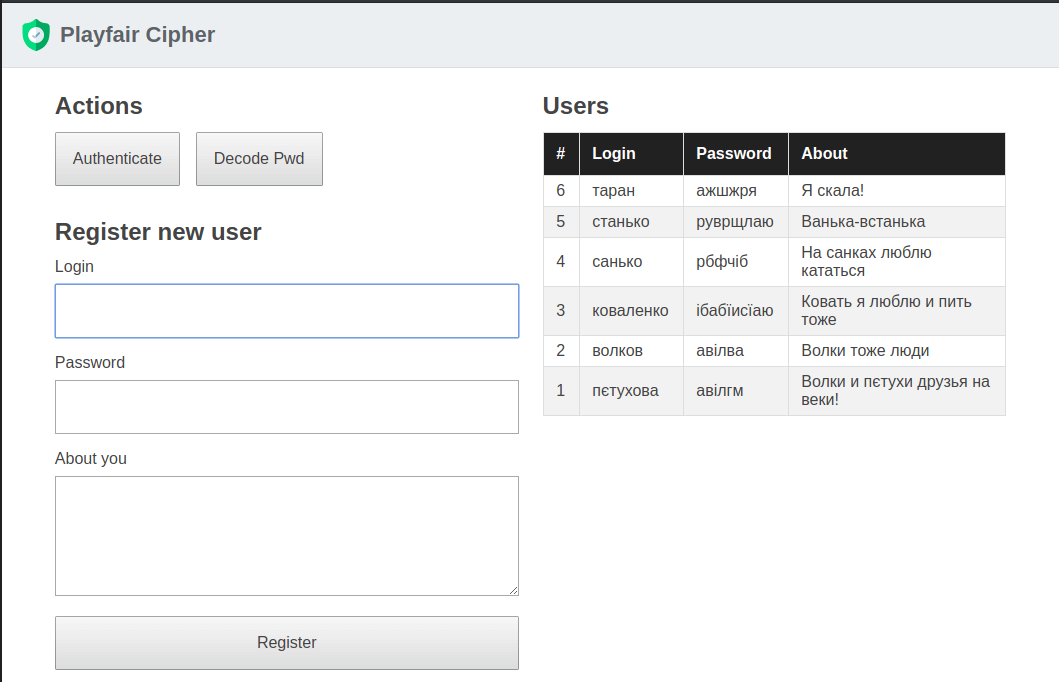
****

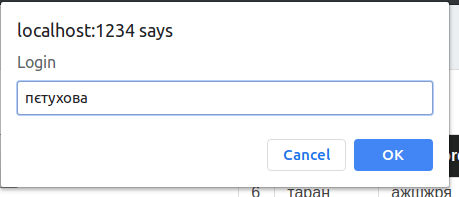
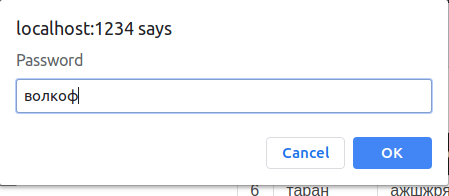
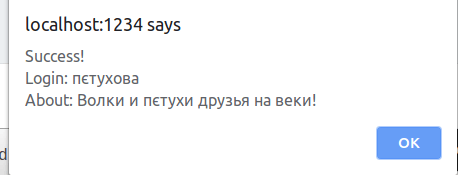
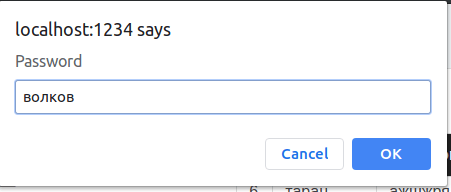
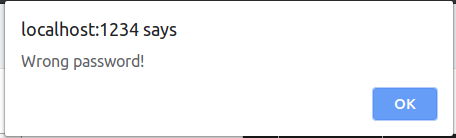
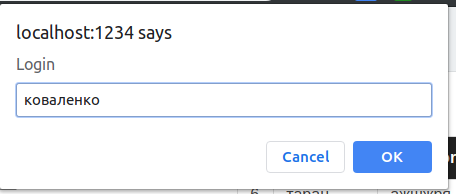
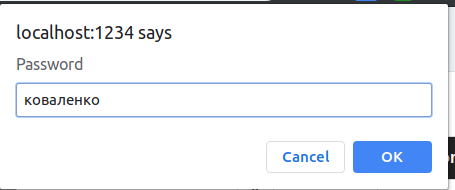
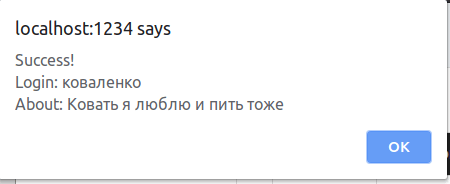
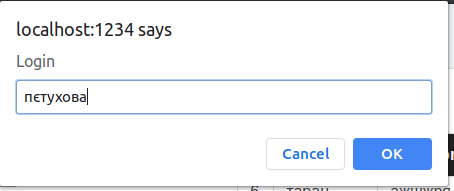
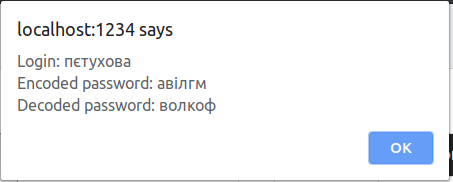
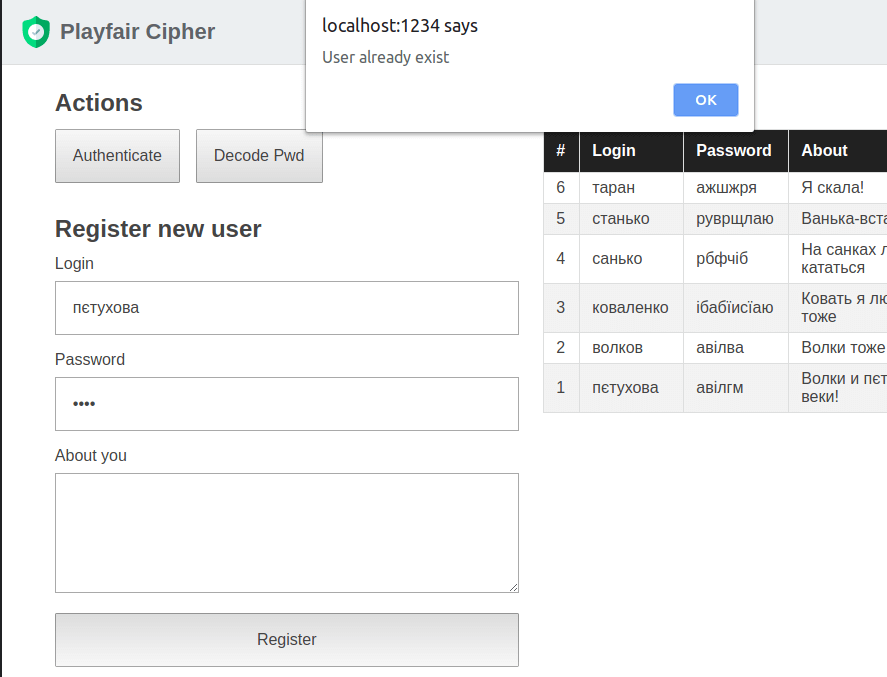
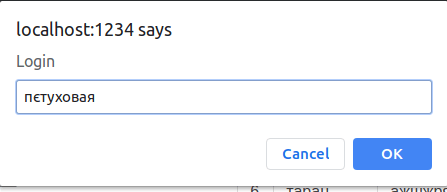
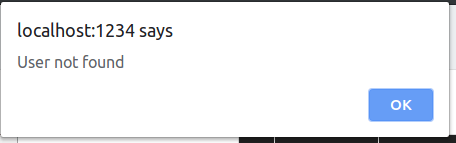










**Висновок:** на цій лабораторній роботі я отримав представлення про типові схеми ідентифікації та аутентифікації, набув практичних навичок створення програм для ідентифікації та аутентифікації з використанням Playfair Cipher.