НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

Дипломний проект

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: “Веб-сервіс шифрування на базі алгоритму Енігма”

Виконав: студент 4-го курсу, групи ІП-74 Мещеряков Олександр Андрійович
(шифр групи)
(прізвище, ім’я, по батькові) (підпис)

Керівник ас. Алещенко О. В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль)
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)
Освітньо-професійна програма
“Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Мещеряков Олександр Андрійович

1. Тема проєкту “Веб-сервіс шифрування на базі алгоритму Енігма”
керівник проєкту Алещенко Олексій Вадимович, асистент, затверджені наказом
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
по університету від 11.05.2021 року № 1139-с
2. Термін здачі студентом закінченого проєкту 01.06.2021 .
3. Вихідні дані до проєкту *технічна документація. Клієнт-серверна система. Express.js, React.js*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис предметної області, аналіз існуючих аналогів, дослідження методики побудови клієнт-серверних систем фреймворками React та Express, дослідження необхідних технологій, розробка веб-сервісу шифрування на базі алгоритму Енігма.
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень): структурна схема, схема взаємодії, функціональна схема.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	проф. Сімоненко В. П.		

7. Дата видачі завдання_____

Календарний план

№	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*28.12.2020*	
2.	*Вивчення та аналіз завдання*	*15.01.2021*	
3.	*Розробка архітектури та загальної структури системи*	*11.03.2021*	
4.	*Розробка структур окремих підсистем*	*17.04.21*	
5.	*Програмна реалізація системи*	*22.05.2021*	
6.	*Оформлення пояснювальної записки*	*29.05.2021*	
7.	*Передзахист*	*02.06.2021*	
8.	*Захист*	*16.06.2021*	

Студент-дипломник Олександр Мещеряков

(підпис)

Керівник проєкту Олексій АЛЕЩЕНКО

(підпис)

АНОТАЦІЯ

В дипломній роботі були проаналізовані вже створені веб версії машини шифрування Енігма та створено веб-сервіс шифрування на базі алгоритму Енігма. В процесі аналізу були розглянуті принципи роботи сервісу шифрування та графічний інтерфейс веб-додатків.

Розроблена система має простий для користування інтерфейс та швидку обробку даних. Програмну частину було створено з використання фреймворків мови програмування JavaScript React.js для клієнтської частини та Express.js у середовищі Node.js для серверної частини.

ANNOTATION

In this Bachelor’s Degree work, already created web versions of the Enigma encryption machine were analyzed and a web encryption service based on the Enigma algorithm was created. In the process of analysis, the principles of the encryption service and the graphical interface of web applications were considered.

The developed system has an easy-to-use interface and fast data processing. The software part was created using the JavaScript language frameworks React.js for the frontend and Express.js in the Node.js environment for the backend.

Опис альбому

бакалаврського дипломного проєкту
на тему:
“ Веб-сервіс шифрування на базі алгоритму Енігма”

Справки	Формат	Значення	Найменування	Кількість листів	№ екземпляра	Додаток
	А4		*Завдання на дипломний проєкт*	*2*		
	А4	*ІАЛЦ.467100.001 ОА*	*Опис альбому дипломного проєкту*	*1*		
	А4	*ІАЛЦ.467100.002 ТЗ*	*Технічне завдання проєкту*	*3*		
	А4	*ІАЛЦ.467100.003 ПЗ*	Веб-сервіс шифрування на базі алгоритму Енігма *Пояснювальна записка*	*61*		
	А4	*ІАЛЦ.467100.004 Д1*	Структурна схема. Блок-схема алгоритму	*1*		
	А4	*ІАЛЦ.467100.005 Д2*	Схема взаємодії	*1*		
	А4	*ІАЛЦ.467100.006 Д3*	Функціональна схема. Взаємодія модулів програми	*1*		
	А4	*ІАЛЦ.467100.007 Д4*	*Ключові елементи коду програми*	*12*		

					ІАЛЦ.467200.001 ОА		
Змн	*Арк.*	*ПІБ*	*Підпис*	*Дата*			
Розробив		*Мещеряков О*			Веб-сервіс шифрування на базі алгоритму Енігма	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перевірив		*Алещенко О.В.*					
						КПІ ім. Ігоря Сікорського ФІОТ ІП-74	
Н/Контр		*Сімоненко В.П*					
Зав.Каф		*Стіренко С.Г.*					

ТЕХНІЧНЕ ЗАВДАННЯ

бакалаврського дипломного проєкту

на тему:

“Веб-сервіс шифрування на базі алгоритму Енігма”

Зміст

1.НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування технічного завдання це “ Веб-сервіс шифрування на базі алгоритму Енігма”. Область застосування сервісу: шифрування даних для їх зберігання або листування.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки веб-сервісу є завдання на виконання дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут» імені Ігоря Сікорського.

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою проекту є створення веб-сервісу шифрування на базі алгоритму шифрувальної машини Енігма.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки є статті, науково-технічна література, публікації в спеціалізованих виданнях, та публікації в мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розроблюваного продукту.

- Шифрування даних наданих користувачем
- Простий для розуміння інтерфейс
- Надавання користувачу можливості налаштовувати машину.

5.2 Вимоги до програмного забезпечення

- Операційна система Linux, macOS, Windows
- Доступ до Інтернету

- Браузер Google chrome, Microsoft Edge або Firefox

5.3 Вимоги до апаратної частини

- Не менше 1 Гбайт оперативної пам'яті
- Процесор Intel Core i3 та вище
- Жорсткий диск об’ємом не менше ніж 1 Гбайт простору.

6. ЕТАПИ РОЗРОБКИ

Етап	Дата
Вивчення та аналіз завдання	28.12.2020
Аналіз існуючих рішень	17.01.2021
Вибір засобів розробки програмного продукту	12.03.2021
Розробка алгоритму машини Енігма	15.04.2021
Розробка клієнт-серверної архітектури	20.05.2021
Оформлення пояснювальної записки	29.05.2021

Пояснювальна записка

до дипломного проєкту

на тему:

”Веб-сервіс шифрування на базі алгоритму Енігма”

Київ – 2021

ЗМІСТ

Оглавление

ВСТУП

У сучасному світі постає проблема недоторканності особистого листування. На зараду цьому приходить шифрування повідомлень що ускладнює можливість зрозуміти(розшифрувати дані) одним з способів шифрування є машина Енігма. Її першу версію розробили у 1940х роках.

Енігма — група електромеханічних роторних машин, що використовувалися з 1920 для шифрування повідомлень. Енігма використовувалася в комерційних цілях та у військових і державних службах у багатьох країнах світу, але найчастіше з'являлася у Німеччині під час Другої світової війни. Найчастіше згадується військова модель — Wehrmacht Enigma.

У 1932 польські криптографи вперше зламали німецький військовий шифр «Енігми». Машина прославилася, тим що змогла розшифрувати значні обсяги зашифрованої інформації за допомогою криптоаналітиків. Спеціально для цих цілей створили машину з кодовою назвою «Бомба», яка забезпечила значну перевагу антигітлерівському союзу у війні. Вся інформація, отримана методом криптоаналізу, називалася ULTRA. [1]

Що таке криптографія?

Криптографія - це процес збору набору даних та забезпечення їх захисту та секретності шляхом кодування або скремблювання. Секретні ключі, про які мова піде далі, генеруються криптографічним алгоритмом - шифром, і ці ключі поєднуються з силою алгоритму та типом мережі, який він використовує для визначення міцності шифру. Існує багато різних типів криптографічних алгоритмів. Найпростішим є симетричне шифрування ключа. Симетричний шифр ключа вимагає лише використання одного ключа для шифрування та дешифрування даних. Тобто, єдиний ключ скремблює дані і розміщується в даних таким чином, що дозволяє дешифрувати дані тим самим ключем, як правило, використовуючи зворотний XORed якимось чином . Інший тип шифру - це

двошифровий шифр, який називається шифруванням із відкритим ключем. Коли виконуються операції двоклавішних асиметричних шифрів, вони значно відрізняються від операцій симетричного шифрування ключів. Вони можуть бути більш безпечними, забезпечуючи конфіденційність, а також пропонуючи перевірку цілісності та перевірку автора на відсутність відмови. Однак деякі шифри, такі як AES Рійндаеля, є симетричними шифрами, які виконують ряд окремих операцій, щоб зробити їх безпечними. [1]

Опис Enigma та її внутрішня робота

Ротори, зубці та дзвінок.

Три впорядковані зліва направо ротори, які підключили 26 входів до 26 вихідних точок контакту на альтернативних гранях диска. Двадцять шість зубців навколо зовнішньої частини роторів, що дозволило оператору вказати початкове положення обертання для зазначених роторів.

Кільцеве випромінювання, установка кільця на кожному роторі контролювала обертальну поведінку ротора за допомогою насічки. Ротори мали діаметр близько 10 см і мав ряд підпружинених штифтів на одній грані, розташованих по колу. На протилежному боці цього було відповідне число круглих електричних контактів. Шпильки та контакти представляють алфавіт, як правило, 26 букв англійського алфавіту. Коли пліч-о-пліч штирі одного ротора опираються на контакти сусіднього ротора, утворюючи електричне з'єднання. Набір з 26 проводів з'єднує контакти з одного боку за складною схемою з контактами з іншого боку, відрізняючись для кожного ротора. В основному ротор сам по собі виконує просте замінне шифрування.

Прикладом цього може бути S, підключений до C з одного боку на інший. Міцність відбувається завдяки використанню трьох або більше роторів послідовно при регулярних рухах роторів, що додають більшої міцності.

Ротор може повертати оператор вручну в одне з 26 положень. Для операторів кожен ротор має буквене кільце, прикріплене до зовнішньої сторони диска, яке можна побачити через вікно, що вказує положення ротора оператору. Можливість регулювання алфавітного кільця до основної проводки була додана пізніше до версії M3. Позиція кільця відома як Ringstellung. [1]

Кожен ротор має виїмку або кілька насічок, які використовуються для управління їх кроком. Коли Enigma була вперше випущена, було доступно лише три ротори. У 1938 р. Це змінилося на п’ять, проте з цих п’яти було обрано лише трьох, які будуть поміщені в справжню загадку. Вони мали римські цифри I, II, III, IV і V, щоб їх розрізнити але це врешті-решт дозволило протидіяти двом окремим методам атаки. Військово-морська версія додала VI, VII і VIII, в той час як вона також дозволяла мати четвертий ротор у власне машині, проте вона не замінила відбивач. Проблема четвертого полягала в тому, що він не ступав, його можна було встановити лише вручну в одне з 26 положень.

Відбивач(Reflector).

Остання частина ланцюга, відбивач, напівротор, щоб відправити вхід назад через ротори: Останній "ротор" у лінії насправді не обертати. Його називали відбивачем (Umkerwhalze, реверсивний ротор). Це була особливість, характерна виключно для Enigma у епоху машин шифрування ротора. Відбивач з'єднав останній ротор попарно, відбиваючи струм назад іншим шляхом. Відображення було там, щоб виконати інволюцію. Інволюція - це математичний термін функції, яка є власною оберненою. такий, що:

f (f (x)) = x для всіх x в області f

Це означало, що шифрування було таким самим, як і дешифрування. Проблема, створена відбивачем, полягала в тому, що жоден лист ніколи не буде зашифрований сам до себе. Пізніше це буде використано кодувальниками в парку Блетчлі. В деяких варіантах загадки відбивач вступав би під час шифрування, в інших - ні. У будь-якому випадку відбивач мав двадцять шість положень.

Відбивач - На кінці шпинделя ротора був нерухомий відбивач з 26 буквеними клемами на одній стороні, що з'єднуються з останнім (третім) ротором. Клеми відбивача були з'єднані між собою парами тринадцятьма поперечними з'єднаннями, які направляли сигнал назад через інший термінал останнього ротора, змінюючи сигнал через інший шлях назад через ротори. Таким чином сигнал був подвійно зашифрований роторним механізмом.[1]

Математичний опис Enigma.

Перетворення кожної букви в Enigma можна вказати як породження перестановок. Припускаючи модель з трьома роторами:

E: Шифрування

ρ: циклічна перестановка

P: перетворення плагіна

U: відбивач

L, M, R: лівий, середній, правий ротори

Період моделі з трьома роторами буде таким: 26 x 25 x 26 = 16 900. Причина, що це не 26 x 26 x 26, пов’язана з подвійним кроком другого ротора. [1]

E = P * R * M * L * U * (L− 1) (M− 1) (R− 1) (P− 1) Після натискання кожної клавіші- ротори повертаються. Це змінює трансформацію. Звідси випливає Формула 1.1 Математичний опис Енігма:

E = P (ρiRρ − i) (ρjMρ − j) (ρkLρ − k) U (ρkL− 1ρ − k) (ρjM− 1ρ − j) (ρiR− 1ρ − i) (P−1)

Плата розширення

Плата реалізує додатковий шифр підстановки, який просто міняє місцями пари букв. Хоча проводка роторів і відбивача була зумовлена способом їх виготовлення, проводку комутаційної панелі оператор міг легко змінити, підключивши кабелі до портів.

У комутаційну панель можна було вставити до 13 кабелів. Кабель, вставлений між двома буквами, змусив ці букви помінятися місцями. Незважаючи на те, що він здається простіше, ніж ротор, комутаційна панель була основним джерелом криптографічного стійкості шифру Enigma.[1]

РОЗДІЛ 1

АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ

1.1Опис завдання

Завдання даної роботи полягає в створенні клієнт-серверної архітектури системи, що дає змогу користувачам за потреби шифрувати та розшифровувати потрібний їм текст. Система повинна надавати можливість надійного шифрування та користувацьких налагоджувати системи шифрування.

Система шифрування, що виконується в даній бакалаврській роботі, повинна реалізувати зручний у користуванні веб-додаток що складається з клієнтської частини, серверної частини, що взаємодіють між собою для забезпечення попиту клієнта.

Клієнтська частина додатку це набір сторінок, які реалізують користувальницький інтерфейс який клієнт бачить при використанні сервісу відкривши його за гіперпосиланням або ввівши адресу в зручному для клієнта пошуковій системі та взаємодіяти з елементами графічного інтерфейсу користувача, в залежності від реалізованої логіки, використовуючи графічні елементи користувацького інтерфейсу, такі як: кнопки, поля введення тексту, спливаючі підказки, графічні додатки тощо.

Клієнтська частина допомагає користувачеві зрозуміти на інтуітивному рівні як взаємодіяти з серверною частиною додатку у простому та зрозумілому для нього виді. Частина взаємодії з клієнтом створюється за допомогою технологій, які можуть відтворюватись в сучасних браузерах. Це може бути як сторінки що написані мовою розмічання веб-сторінок html (HyperText Markup Language – “мова розмітки гіпертексту”), з використанням стилів CSS (Cascading Style Sheets – “Каскадні таблиці стилів”) для відображення зовнішнього виду веб-сторінки та її елементів, а також мови програмування JavaScript для

динамічної зміни контенту на HTML-сторінках, так і фреймворки для створення сучасних та інтуїтивно зрозумілих WEB-додатків.

Фреймворк - це програмний продукт, який спрощує створення і підтримку технічно складних або навантажених проектів. Фреймворк, як правило, містить тільки базові програмні модулі, а все специфічні для проекту компоненти реалізуються розробником на їх основі. Тим самим досягається не тільки висока швидкість розробки, а й велика продуктивність і надійність рішень.

Серверна частина додатку це набір реквестів, що відправляють на запит клієнтської частини заданою логікою та реалізують основну частину роботи наприклад додання видалення або зміна даних в базі даних.

1.2 Аналіз існуючих рішень

1.2.1 Enigma симулятор розроблений Матеусом портела

Рисунок 1.1 - Титульна сторінка, що демонструє початкову сторінку симулятору шифрувальної машини Енігма [2]

У даному аналогу користувач вводить потрібний йому текст в блок вводу тексту “input” та одразу отримує його у зашифрованому виді в блоці виводу

тексту “output”, слід зазначити що текст можливо вводити лише послідовно без відступів та англійськими буквами. Також за потреби користувач має можливість відкрити налаштування шифрувальної машини та ввести його власні налаштування:

Користувач має можливість вибрати потрібний йому ротор для трьох роторів з п’яти можливих позицій кожного з роторів (позиції “I”, “II”, ”III”, ”IV”, ”V”). Кожен ротор представляє собою статичний набір з 26 латинських літер яке в справжній машині представлене алфавітною шиною.

Кожен з роторів можна встановити в одне з 26 можливих вихідних положень окремо для кожного з вибраних роторів.

Також користувач може вибрати один з доступних рефлекторів що з’єднує парами виходи останнього ротора перенаправляючи шлях символу який ми шифруемо. В реальній машині рефлектор гарантував що Enigma буде само взаємною це робило можливим те що за допомогою двох однакових за налаштуванням машин (роторів, початкових положень, рефлектору) повідомлення може бути зашифровано на одній машині а розшифровано на іншій без необхідності громіздкого механізму перемикання між режимами шифрування та дешифрування. Можливі позиції рефлектору “B”, “C”.

Користувач може налаштувати пари вводу виводу символів панелі розширення шифрувальної машини. Наприклад, E і Q можуть бути складеною парою. Ефект полягав у обміні цих букв до і після основного блоку ротора. Наприклад, коли оператор натискав клавішу E, сигнал переходив на Q перед входом у ротори. Одночасно може бути використано до 13 пар у складі, хоча зазвичай використовуються лише 10.

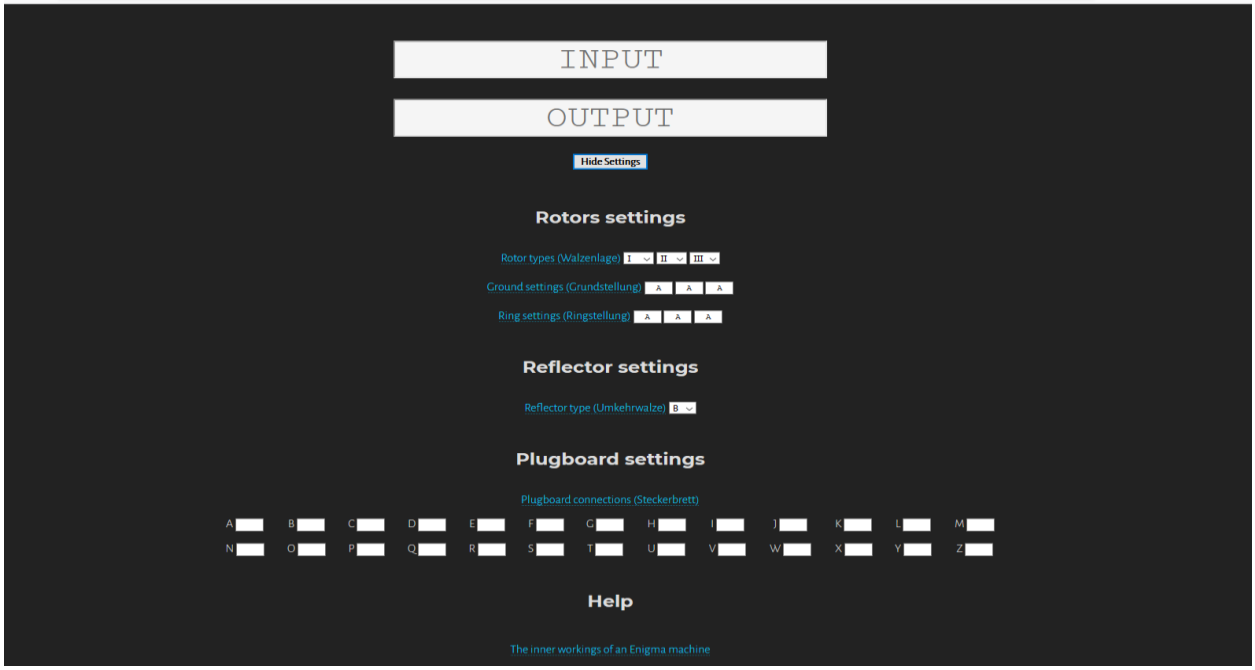


Рисунок 1.2 - Сторінка з користувацькими налаштуваннями симулятору шифрувальної машини Enigma [2]

1.2.2 Універсальна шифрувальна машина Енігма розроблена Даніель Палоком

Рисунок 1.3 - Інтерфейс універсальної шифрувальної машини Енігма.[4]

У даному аналогу користувач вводить потрібний йому текст в блок вводу тексту “input” та отримує його у зашифрованому виді при кожному натисканні в блоці виводу тексту “output”.

Користувач має можливість змінювати модель машини на одну з доступних з властивостями що належать вибраній шифрувальній машині.

Рисунок 1.4 - Блок вибору версії шифрувальної машини Енігма.[4]

Користувач має можливість за потреби змінити рефлектор на B, C, D* та вибрати ротор(I, II, III, IV, V, IV, IIV, IIIV) для кожної з трьох можливих позицій та налаштувати початковий символ для них.

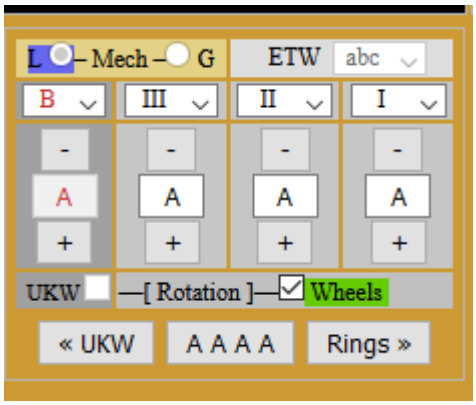


Рисунок 1.5 - Інтерфейс користувацьких налаштувань[4]

При натисканні кнопки “Rings” відкривається меню Ring setting де користувач може налаштувати початкову кількість кроків кожного з роторів та рефлектору та за потреби має кнопну “Home pos” яка повертає кроки кожного з кілець у початкове становище.

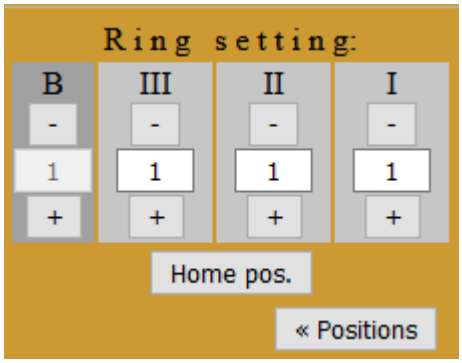


Рисунок 1.6 - інтерфейс вибору поточного кроку[4]

При натисканні кнопки “UKW” відкривається меню з поточним налаштуванням зв’яків символів на ввод/вивід плати розширення наприклад введемо що A замінюється з Q після чого потрібно підтвердити це натисканням кнопми “Activate”. Також для зручно є кнопка “Clear” що повністю зтирає вибрані набори пар.

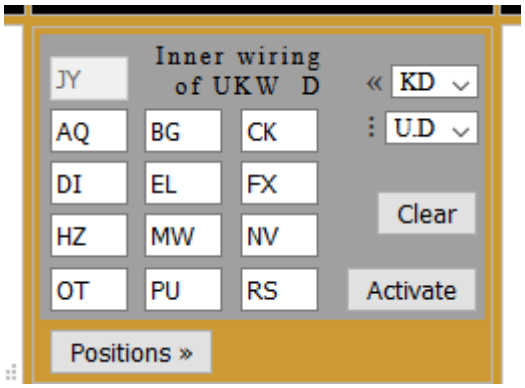


Рисунок 1.7 - Інтерфейс зв’язків символів плати розширення на ввід/вивід[4]

При натисканні кнопки “QWERTZU keys” на основній панелі в вікні вводу input відкривається панель яка підсвічує літеру повідомлення в зашифрованому виді також в режимі цього вікна ввід тексту можна робите не лише з клавіатури а також і натисканням курсором на потрібну літеру на панелі.

Рисунок 1.8 - Панель вводу повідомлення[4]

При натисканні “Show plugbord” відкривається панель для вводу пар символів що будуть взаємозаміняти один одного ця операція також потребує натискання кнопки “Activate” для підтвердження. Э кнопка “Clear all” для швидкого очищення кожної з полів вводу.

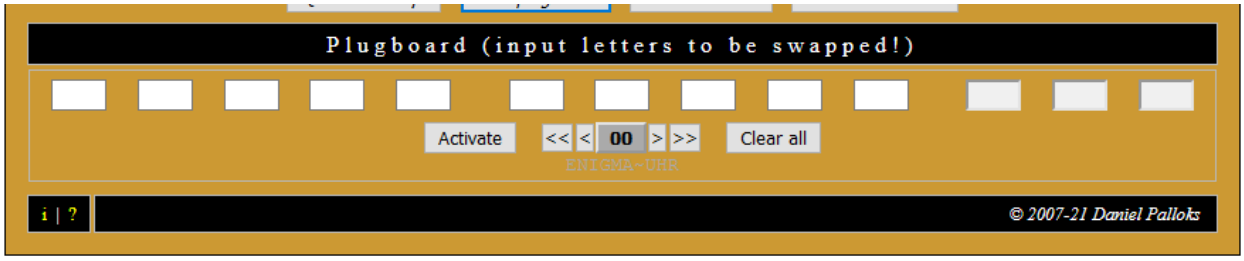


Рисунок 1.9 - Панель вводу пар, які авто замінюють один одного[4]

Кнопка “Show monitor” відкриває панель що демонструє налаштування на даний момент такі як кількість літер що були введені останній символ який ми ввели ротори які використовуються та їх поточне становище та крок а також ротори та рефлектор у виді послідовності символів відносно алфавіту.

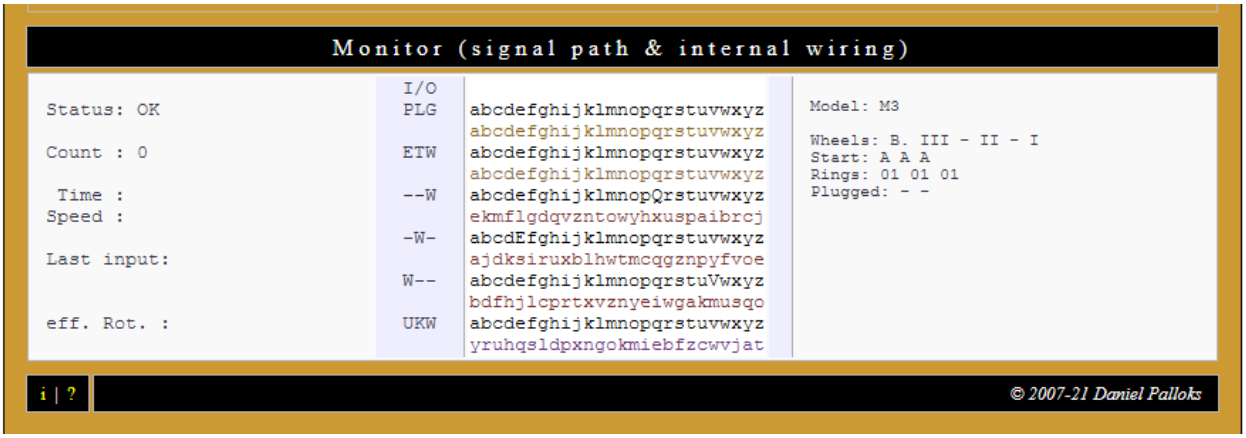


Рисунок 1.10 - Панель виводу налаштувань[4]

1.2.3 Шифрувальна машина Енігма створена trys

У даній версії шифрувальної машини Енігма фігурує простий зрозумілий дизайн що відтворює вигляд справжньої машини Енігма. Під час використання даного аналогу клієнт на зручному інтерфейсі бачить відображення поточної кількості кроків кожного з ротерів (кількість кроків збільшується після кожного кодування символу) та має можливість змінити їх кількість за бажання за допомогою натискання на блок відображення чисел та їх подальшого змінення шляхом вводу з клавіатури потрібного числа і підтвердити натисканням кнопки “Set”.

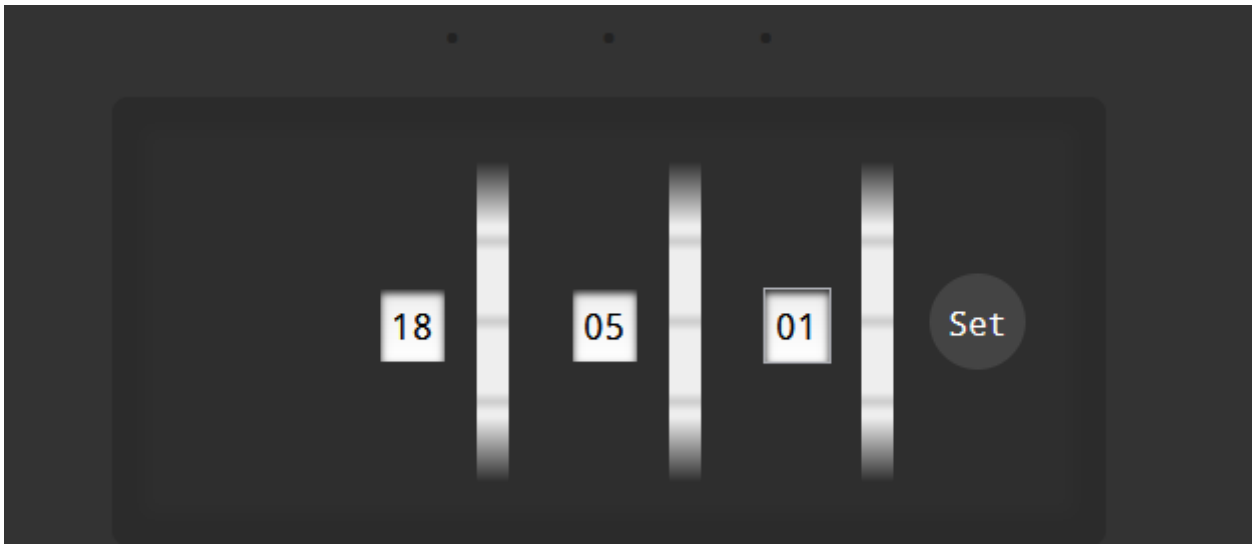


Рисунок 1.11 - Інтерфейс поточного кроку роторів [5]

Ввід даних відбувається за допомогою поля вводу по одному символу лише англійськими літерами без пробілів використовуючи клавіатуру. При кожному натисканні клавіші користувач одразу бачить символ підсвіченим в закодованому виді на інтерфейсі.

Рисунок 1.12 - Інтерфейс вводу посилання [5]

Для того щоб переглянути весь зашифрований текст потрібно нажати перемикач справа від інтерфейсу виводу. Для перемикання назад потрібно нажати перемикач повторно.

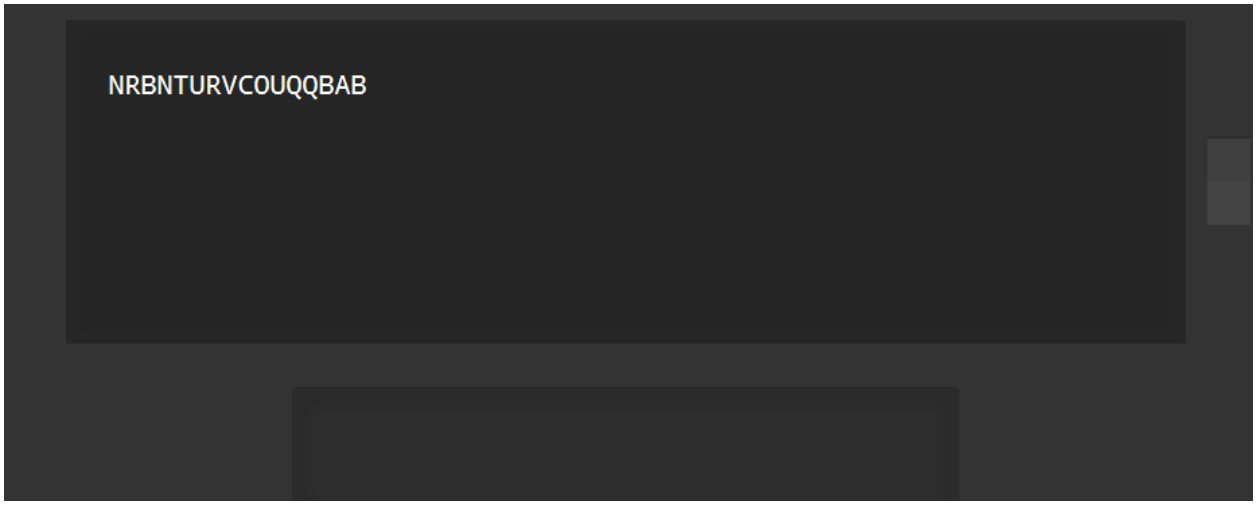


Рисунок 1.13 - Інтерфейс виводу зашифрованого тексту [5]

Для налаштування пар ключів рефлектору використовується інтерфейс в нижній частині сторінки шляхом натискання на один з символів і подальшого вводу коректних пар-ключів (максимальна кількість пар – 10) після кожної зміни пар в внутрішній частині інтерфейсу змінюється графічна частина відповідно до зроблених кроків.

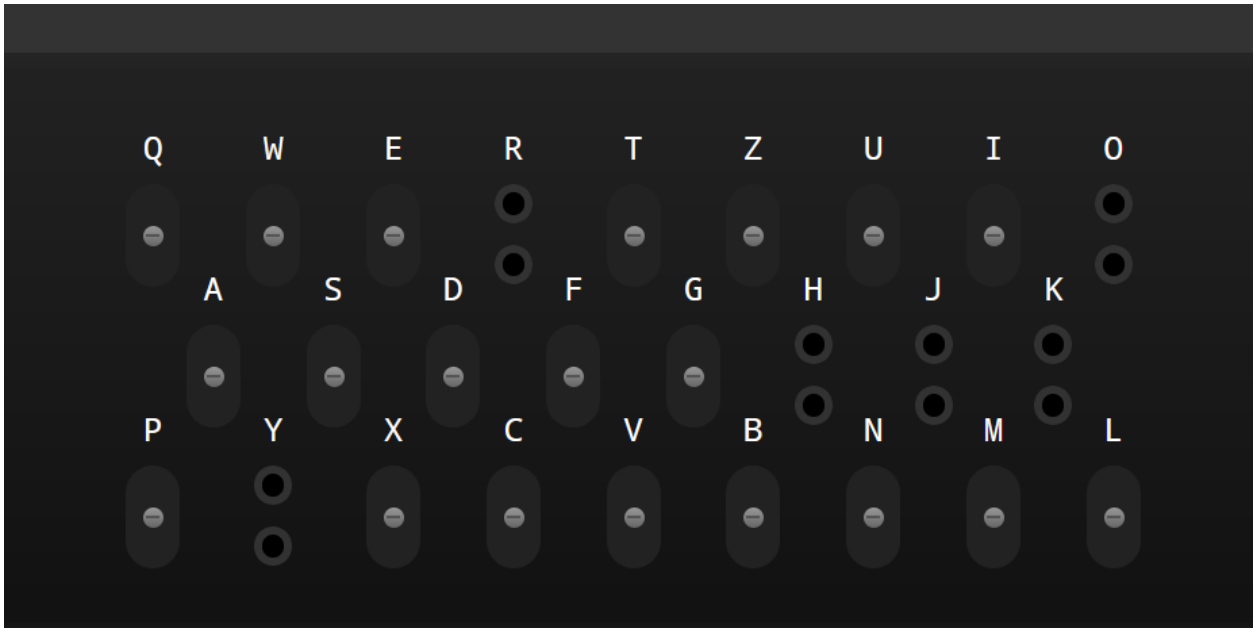


Рисунок 1.14 - Інтерфейс ключів пар для взаємозаміни [5]

1.2.4 Шифрувальна машина Енігма від dcode

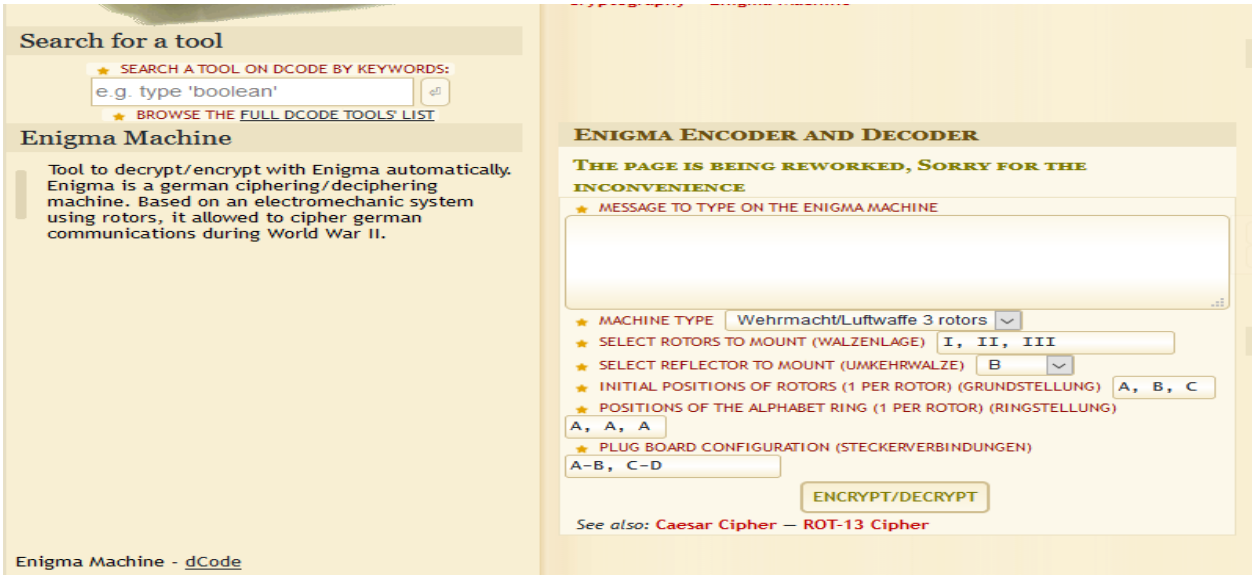


Рисунок 1.15 - Основний інтерфейс веб-додатку.[6]

У даному аналогу користувач вводить потрібний йому текст в блок вводу тексту “input” та при натисканні кнопки Encrypt/Decrypt отримує його у зашифрованому виді виводом тексту у полі results, слід зазначити що текст можливо зашифрувати використовуючи відступи та букви будь-якої мови але в зашифрованому виді вони отримуються користувачем лише послідовно без відступів ігноруючи шифрування усіх літер окрім літер англійської мови. Також

за потреби користувач має можливість використати налаштування шифрувальної машини та ввести його власні налаштування:

Користувач може вибрати один з трьох доступних типів машини Енігма.

Користувач має можливість вибрати потрібний йому ротор для трьох роторів з п’яти можливих позицій кожного з роторів (позиції “I”, “II”, ”III”, ”IV”, ”V”). Для вводу ротерів потрібно послідовно через кому вписати їх в рядок. Кожен ротор представляє собою статичний набір з 26 латинських літер яке в справжній машині представлене алфавітною шиною.

Кожен з роторів можна встановити в одне з 26 можливих вихідних положень окремо для кожного з вибраних роторів.

Користувач може вибрати початковий крок для кожного з ротерів.

Також користувач може вибрати один з доступних рефлекторів що з’єднує парами виходи останнього ротора перенаправляючи шлях символу який ми шифруемо. В реальній машині рефлектор гарантував що Enigma буде само взаємною це робило можливим те що за допомогою двох однакових за налаштуванням машин (роторів, початкових положень, рефлектору) повідомлення може бути зашифровано на одній машині а розшифровано на іншій без необхідності громіздкого механізму перемикання між режимами шифрування та дешифрування. Можливі позиції рефлектору “B”, “C”,”BThin”,”CThin”.

1.2.5 Шифрувальна машина Енігма від 101computing.net

У наступному прикладі розробники використали дизайн приближений до справжньої машини Енігма. Складається з роторів, клавіатури вводу, інтерфейсу виводу символу в зашифрованому виді та інтерфейсу пар ключів заміни для рефлектору.

У даній машині фігурує три ротори кожен з яких складається з 26 символів англійської мови послідовність розташування яких залежить від вибраних користувачем налаштувань. Для вводу символів користувач повинен натискати на потрібний йому текст на інтерфейсі вводу після чого отримає результат

шифрування на інтерфейсі виводу. Наприклад, при шифруванні повідомлення, що складається з ABC, користувач спочатку натискає кнопку A після прорахунку системою результату користувач отримує спалахування елементу інтерфейсу Z, отже Z є першою буквою криптограми і потім користувач таким самим чином шифрує B і так далі.

Рисунок 1.16 - Інтерфейс вводу посилання [7]

Для зручності даний аналог має інтерфейс для віддалення, приближення, відкриття в повно екранному режимі та відкриття короткої карточки-інструкції яким чином користуватися даною машиною шифрування.

Рисунок 1.17 - Інтерфейс допоміжних засобів [7]

Для зміни налаштувань ротора та рефлектору користувач повинен повинен натиснути на інтерфейс роторів після чого відкривається меню в якому користувачеві доступні такі дії:

Зміна рефлектору на одну з версій UKW-B або UKW-C.

Зміна ротору на один з доступних (I, II, III, IV, V)

Також користувач може вибрати початкові налаштування кожного з ротору а саме він може вибрати з якої позиції починається крок.

Користувач може вибрати з якої літери починається крок ротору.

Для підтвердження налаштувань потрібно натиснути “Apply Settings”.

Для відміни налаштувань потрібно натиснути “Cancel”.

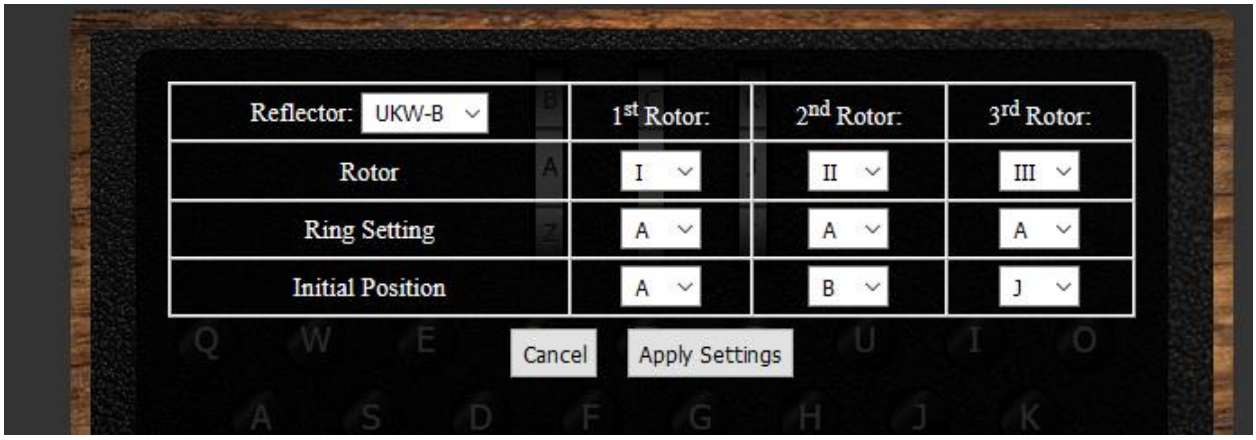


Рисунок 1.18 - Інтерфейс користувацьких налаштувань [7]

Для налаштувань ключів пар заміни для рефлектору використовується інтуїтивно зрозумілий інтерфейс який працює таким чином: для встановлення ключа пари потрібно натиснути на 2 потрібні літери після чого вони підсвічуються одним кольором а наступна пара матиме інший колір. При натисканні на вже встановлену пару – пара на яку натиснули зникає .

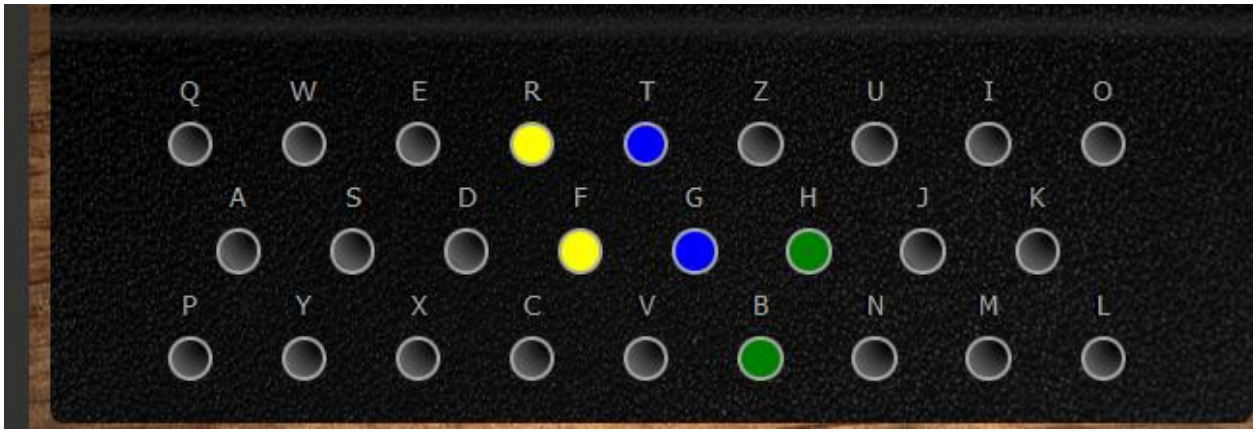


Рисунок 1.19 - Інтерфейс ключів-пар взаємозаміни [7]

Після кожного натискання клавіші вводу користувачеві виводить в блоці “Plaintext” текс який був введений з проміжком в кожні п’ять літер та текст в зашифрованому виді в блоці виводу “Ciphertext” також з проміжком в п’ять літер.

При зміни режиму в шифрування та дешифрування даних цієї машини Енігма кнопками “Encrypt” та ”Dectypt” змінюється розташування блоків вводу та виводу та повністю стирається весь текст який був записаний на даний момент.

Для стирання тексту користувач використовується елемент інтерфейсу “Clear” після натискання якого весь поточний текс стирається.

Також користувачеві доступна кнопка “Show Encryption Steps” яка відкриває інтерфейс вводу кожної з символів криптограми та їх шлях по шифрувальній машині Енігма.

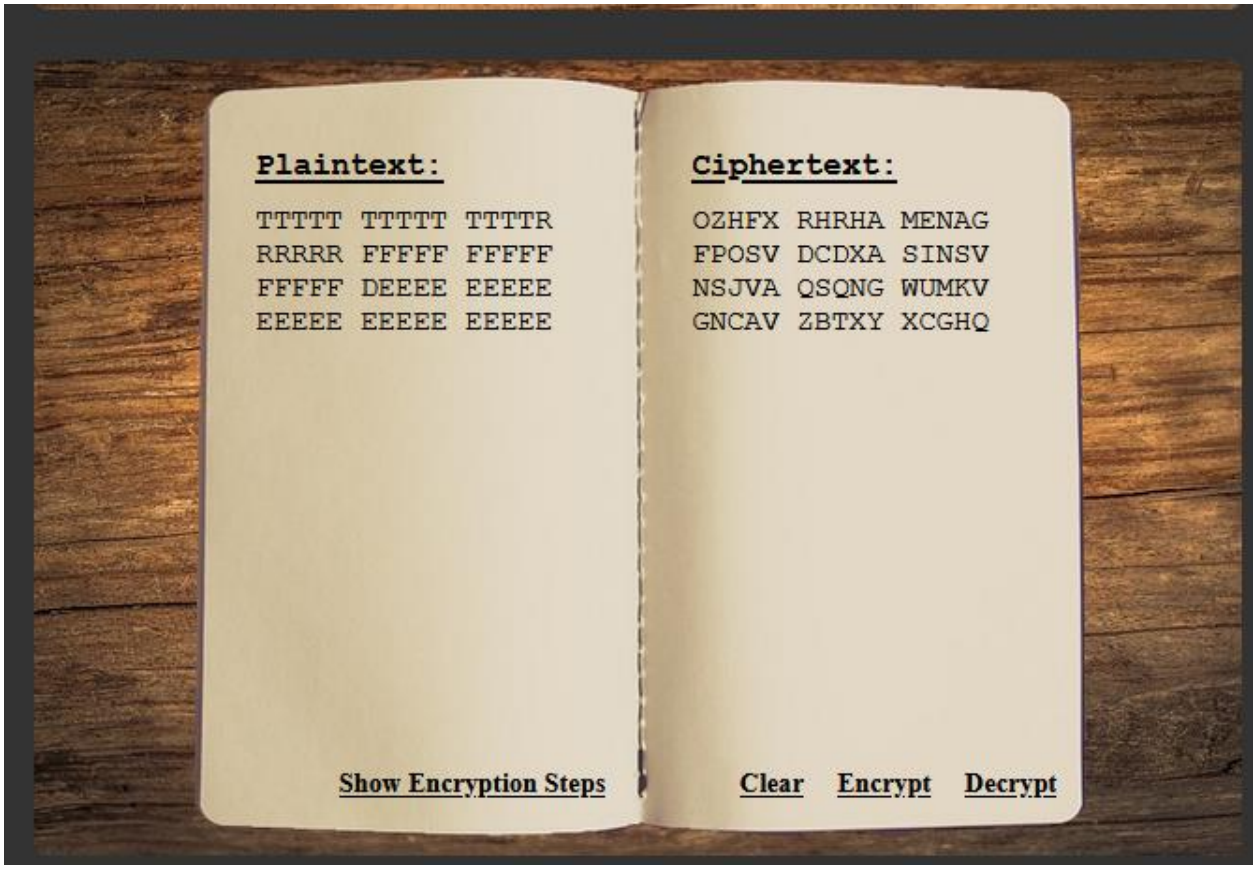


Рисунок 1.20 - інтерфейс виводу введеного посилання та зашифрованого тексту [7]

У даному інтерфейсі користувач має можливість побачити шлях кожного з символів що він шифрує в порядку від першого введеного символу до останнього зверху до низу. Поля інтерфейсу відображають такі дані зашифрованих символів:

- Символ який був зашифрований
- Позиція роторів на момент шифрування символу
- Літера на яку замінюється літера парами ключами для заміни
- Символ в зашифрованому виді в роторі 3
- Символ в зашифрованому виді в роторі 2
- Символ в зашифрованому виді в роторі 1
- Символ в зашифрованому виді в рефлекторі
- Символ в зашифрованому виді в роторі 1
- Символ в зашифрованому виді в роторі 2
- Символ в зашифрованому виді в роторі 3
- Літера на яку замінюється літера парами ключами для заміни
- Результат отриманий після проходження роторів та рефлектору

Рисунок 1.21 - Інтерфейс налаштувань на кожному кроці шифрувальної машини.[7]

ВИСНОВОК ДО ПЕРШОГО РОЗДІЛУ

В першому розділі були наведені приклади вже створених аналогів шифрувальної машини Енігма. Впродовж розділу були розглянуті елементи розробленого інтерфейсу та деталі роботи кожної з розглянутих машин.

Майже кожен з розглянутих аналогів відтворював інтерфейс справжньої машини Енігма беручи за основу модель Enigma M3, що складається з 3 роторів, кожен з яких має п’ять типів (I, II, III, IV, V) та складається з 26 літер англійської мови, рефлектору що має два типи B та C, інтерфейсу вводу, що представляє собою або поле вводу, або інтерфейс на якому потрібно було мишкою пролікувати потрібні літери. Також вона мала інтерфейс виводу які поділялися на простий блок виводу та інтерфейс клавіатури машини Енігма на якій підсвічувалася літера в зашифрованому виді. Наприклад якщо користувач шифрує літеру A, то після переходу літери через усі ротори рефлектор та плату розширення на інтерфейсі виводу підсвітиться літера Z.

Кожна з машин мала інтерфейс користувацьких налаштувань в якому давалася можливість змінювати поточні ротору на один з варіантів, змінювати крок на якому зараз знаходиться машина, та змінювати внутрішній крок ротору.

Кожна з машин має інтерфейс зміни пар-ключів рефлектору. Максимальна кількість пар 13, але частіше використовується 10 пар.

РОЗДІЛ 2.

АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ КЛІЄНТ-СЕРВЕРНОГО ВЕБ-ДОДАТКУ

2.1JavaScript

JavaScript це мова програмування, яка є об’єктно-орієнтованою прототипною мовою з частковим підтриманням функціональної парадигми програмування. JavaScript був прототипом мови ECMAScript та наразі є реалізацією стандарту мови ECMAScript. Перша версія з’явилася ще в 1995 году і з тих пір постійно покращувалася, доки не набула нинішнього вигляду.

Найчастіше JavaScript використовується в розробці вебсторінок та вебзастосунків з цілью надати їм інтерактивності та динамічності та програмування серверної частини.

Базовою особливістю цієї мови відзначається те, що на неї вплинули інші (Python, Java та ін.) мови програмування з яких JavaScript перейняла функціонал та особливості які надали мові максимального комфорту та легкості розуміння користувачем. [12]

З його допомогою доступні до використання наступні функції

- Можливість змінювати веб-сторінки браузерів;
- Додавання та стирання тегів;
- Змінення стилів сторінки;
- Можливість відслідковувати дії користувачів на сторінці;
- Запит доступу до деякої частини вихідного коду сторінки;
- Редагування коду;
- Використання cookie.

Область застосування мови досить широка і не обмежена однією областю застосування: серед програм, для яких використовують JavaScript, присутні і тестові редактори, і веб-додатки (як для комп'ютерів, так і мобільні і навіть серверні), і прикладне програмне забезпечення.

Переваги JavaScript

1. Жоден з сучаних браузерів не обходиться без підтримки JavaScript.
2. Легкість розуміння базового коду користувачем.
3. Корисні функціональні налаштування.
4. Мова постійно покращується – наприклад наразі розроблюється ECMAscript2021
5. Взаємодія з веб-додатком має можливість здійснюватися через текстові редактори наприклад Microsoft Office і Open Office.
6. Значна кількість фреймворків для різних задач.

Недоліки JavaScript

1. Знижений рівень безпеки через повсюдного і вільного доступу до вихідного коду популярних скриптів.

2. Значна кількість дрібних помилок на кожному етапі розробки. Значна кількість серед них легко виправляється, але їх наявність дозволяє вважати цю мову програмування менш професійною, порівняно з іншими мовами.

3. Своєрідним недоліком можна вважати той факт, що частина активно використовуваних програм (особливо додатків) перестануть існувати при відсутності мови, оскільки цілком базуються на ньому. [12]

2.1.1 Історія JavaScript

JavaScript було розроблено Netscape у співпраці з Sun Microsystems. До JavaScript веб-браузери були досить простим програмним забезпеченням, здатним відображати гіпертекстові документи. Пізніше було введено JavaScript, щоб додати додаткову пряність веб-сторінкам та зробити їх більш інтерактивними. Перша версія JavaScript 1.0 була дебютована в Netscape Navigator 2 у 1995 році.

На момент випуску JavaScript 1.0 Netscape Navigator домінував на ринку браузерів. Microsoft намагалася наздогнати власний браузер Internet Explorer і швидко послідувала керівництву Netscape, випустивши власну мову VBScript, а також версію JavaScript під назвою JScript з доставкою Internet Explorer 3. Як відповідь на це , Netscape та Sun разом із Європейською асоціацією виробників комп'ютерів (ECMA) взялися за стандартизацію мови. Результатом стала ECMAScript, ще одна назва тієї самої мови. Хоча назва ніколи насправді не застрягла, ми дійсно повинні посилатися на JavaScript як ECMAScript. [8]

JavaScript, ECMAScript, JScript - набирав популярності до 1996 року. Браузери версії 3 від Netscape та Microsoft підтримували в різній мірі мову JavaScript 1.1.

JavaScript спочатку називався LiveScript. Можливо, було вибрано JavaScript, щоб нова мова звучала так, ніби вона була в хорошій компанії. На жаль, вибір цієї назви призвів до змішування двох мов у свідомості людей - плутанина, яка посилювалась тим, що веб-браузери також підтримували форму Java на стороні клієнта. Однак, хоча сила Java полягає в тому, що теоретично її можна розгорнути майже в будь-якому середовищі, JavaScript завжди був призначений для обмеження веб-браузера. [13]

JavaScript - це мова сценаріїв. На відміну від програми, яка робить все сама, мова JavaScript просто говорить веб-браузеру, що робити. Веб-браузер інтерпретує сценарій і виконує всю роботу, саме тому JavaScript часто порівнюється з несприятливими компільованими мовами програмування, такими

як Java та C ++. Але відносна простота JavaScript - це і його сила.. JavaScript також пропонує розробникам можливість маніпулювати аспектами веб-браузера.

Наприклад, мову можна використовувати для регулювання властивостей вікна браузера, таких як його висота, ширина та положення. Звернення до власних властивостей браузера таким чином можна сприймати як об’єктну модель браузера. Ранні версії JavaScript також забезпечували примітивний різновид DOM(Document Object Model). [12]

2.2Веб-сценарії

Веб-браузер забезпечує середовище ECMAScript для виконання певних обчислень на стороні клієнта, включаючи, наприклад, об'єкти, що представляють вікна, меню, спливаючі вікна, діалогові вікна, текстові блоки, прив'язки, кадри, історію, файли cookie та блоки вводу / виводу. Крім того, хост-середовище надає засіб для прикріплення скриптового коду до таких подій, як зміна фокусу, завантаження сторінки та зображення, вивантаження, помилки та переривання, вибір, подання форми та дії миші або клавіатури. Код сценаріїв відображається в HTML, а відображена сторінка являє собою комбінацію елементів інтерфейсу користувача та фіксованого та обчисленого тексту та зображень. Код сценаріїв реагує на взаємодію користувача, і немає потреби в головній програмі.

Веб-сервер надає інше середовище хосту для обчислення на стороні сервера, включаючи об'єкти, що представляють запити, клієнти та файли; та механізми блокування та обміну даними. Завдяки спільному використанню сценаріїв на стороні браузера та сервера можна розподілити обчислення між клієнтом та сервером, одночасно надаючи користувацький інтерфейс для веб-додатків.

Кожен веб-браузер і сервер, що підтримують ECMAScript, постачає власне середовище хосту, доповнюючи середовище виконання ECMAScript.[11]

2.2.1 JavaScript в порівнянні з ECMAScript

JavaScript був створений в травні 1995 року Бренданом Ейхом під час роботи в Netscape, як повідомляється, лише за 10 днів. Спочатку він був названий Mocha, ім'я, обране засновником Netscape Марком Андріссеном, але через чотири місяці було перейменовано на LiveScript. На той час Sun Microsystems володіла торговою маркою JavaScript. Netscape придбав ліцензію на товарний знак і перейменував LiveScript на JavaScript у грудні 1995 року. Це було певним маркетинговим маневром, оскільки Java на той час була дійсно популярною.

Деякий час між 1996 і 1997 рр. Netscape передав JavaScript організації з питань стандартизації Ecma, щоб розробити та підтримувати специфікацію мови, щоб дозволити іншим постачальникам браузерів реалізовуватись на основі своєї роботи. Технічний комітет Ecma 39 (більш відомий як TC39) був створений для продовження розвитку мови, зрештою випустивши ECMA-262 Ed.1 в червні 1997 р. ECMAScript - це назва офіційного стандарту, причому JavaScript є найбільш відомою реалізацією стандартний. ActionScript (Macromedia) та JScript (Microsoft) є прикладами інших реалізацій.[9]

Версії JavaScript визначаються специфікацією, яка носить офіційну назву мови. Наприклад, першою стандартною версією JavaScript був ECMAScript 1. Зараз загальноприйнятою практикою є скорочення специфікації стандартів просто ESX, хоча X змінюватиметься у 2016 році, коли TC39 перейде на щорічний каденс випуску. Детальніше про це до кінця цього допису. Усі функції ECMAScript проходять офіційний процес подання пропозицій. Пропозиція щодо нової функції, яка, як правило, походить від спільноти розробників, починається з ескізу або "пропозиції солом'яника", що містить початковий опис запропонованої функції. Якщо TC39 погодиться, що ця функція цінна, вона буде підвищена до офіційного статусу "Пропозиція". Кінцевий термін подання пропозицій ES6 був ще у травні 2011 року.

Наступний статус - «В реалізації», де ця функція насправді реалізована в ідеалі 2 механізмів JavaScript. Ці реалізації, а також відгуки спільноти розробників допомагають формувати пропозицію під час її еволюції. Нарешті, якщо пропозиція продовжує виявлятися цінною, TC39 прийме її, і вона буде додана до наступної версії ECMAScript. Зараз це "Стандарт".[9]

2.3 Історія ECMAScript

2.3.1ECMAScript 1, 2 і 3

ECMAScript 1, перша версія мовного стандарту JavaScript, була випущена в червні 1997 року. Рівно через рік була випущена ECMAScript 2, яка містила лише незначні зміни, щоб синхронізуватися з паралельним стандартом ISO для JavaScript. Через 18 місяців у грудні 1999 р. Було випущено ECMAScript 3, який представив багато популярних функцій JavaScript, які ми зараз сприймаємо як належне, такі як регулярні вирази, обробка винятків try / catch та форматування для числового виводу. [9]

2.3.2 ECMAScript 4

Робота над ECMAScript 4 розпочалася на початку 2000 року як величезне оновлення до ES3 і навіть отримала кодову назву JavaScript 2. У 2003 році комітет опублікував проміжний звіт, який містив лише деякі функції, призначені для ES4. Microsoft JScript та Adobe ActionScript реалізували деякі його функції, але робота над специфікацією ES4 повністю зупинилася незабаром після виходу проміжного звіту. [9]

Приблизно в цей час з’явилася нова техніка створення динамічних веб-додатків на стороні клієнта за допомогою JavaScript, що спричинило своєрідний ренесанс JavaScript. Бібліотеки інструментарію JavaScript, такі як jQuery, Prototype, Dojo та Mootools, були випущені в цей період відродження JavaScript.

Джессі Джеймс Гаррет створив цей новий підхід до "Аякса" у доповіді, яку він написав у лютому 2005 року. Ця довідка цілком може бути причиною того, чому TC39 обрав повторне скликання та відновлення роботи над ES4 восени 2005 року. Він повинен був базуватися на теперішнього 7-річного ES3, проміжний звіт ES4 та навчальні матеріали з ActionScript та JScript.

Однак, коли комітет зібрався, вони були розділені на 2 окремі табори. В одному кутку був «табір ECMAScript 4», до складу якого входили Adobe, Mozilla, Opera та Google, які все ще хотіли працювати над масовим оновленням. В іншому куті був Microsoft & Yahoo, “ECMAScript 3.1 Camp”, який шукав невелику підмножину ES4, яка була б більше поступовим оновленням ES3, що містить деякі незначні функції та виправлення помилок.

Розкол пройшов досить глибоко, але, на щастя, у липні 2008 року на зустрічі в Осло, Норвегія, був розмитий компроміс:

По суті, те, що було ECMAScript 3.1, стане ECMAScript 5 - поступове оновлення ES3.

Тоді TC39 розробить новий великий випуск, який буде зменшеною версією амбіційного ES4, але більшим за обсягом, ніж додатковий ES5. Цей основний випуск отримав кодову назву Harmony (через обставини, в яких він був створений) і здебільшого був би тим, що ми зараз знаємо як ECMAScript 6.

Деякі інші особливості оригінального плану ES4 будуть вилучені для подальшого використання.

Результатом компромісу стало те, що ECMAScript 4 був офіційно відмовлений, але було сформовано нове рішення про те, що будь-які нові ідеї будуть розроблятися з консенсусом усього TC39, щоб запобігти потенційним можливим розколам. [10]

2.3.3 ECMAScript 5

ECMAScript 5, поступове оновлення до ES3, було випущено в грудні 2009 року, більш ніж через десять років після ES3. Це версія JavaScript, яка повністю підтримується у всіх браузерах, що використовуються сьогодні, крім Internet Explorer 8.

Він додав кілька покращень до стандартної бібліотеки, таких як:

- Підтримка розбору / серіалізації JSON
- Методи прототипу масиву (як map і forEach)
- Методи перерахування властивостей (як Object.keys).
- Додані аксесори get і set
- Багаторядкові строкові літерали;
- Введені деструктуруючі привласнення
- Було додано стрілочні функції;
- Шаблоннні рядки можна оголошувати рядки з допомогою зворотніх лапок. Шаблонні рядки можуть бути багаторядковими, також можуть інтерполюватися;
- Змінні let та const - альтернатива var.
- Додана “часова мертва зона”;
- Ітератор и протокол ітерації тепер визначаються способ перебору будь-которого об'єкту, а не тільки масівів.
- Symbol використовується для привласнення ітератора до будь-якого об'єкту;
- додані Функції-генератори. Використовують yield для створення послідовності елементів. Функції-генератори можуть використовувати yield * для делегування в іншу функцію генератора, окрім цього можуть повертати об'єкт генератора, Який реалізує обидва протоколи;

- додані проміси для полегшення розробки з використанням колбєків.

Були також синтаксичні оновлення, такі як:

Допуски на звисання коми в кінці списків або визначень об'єктів

Припуски на зарезервовані слова (наприклад, нові або для) як властивості об’єкта

Нарешті, він ввів суворий режим, який зробив JavaScript чистішою мовою, заборонивши деякі функції, виконавши додаткові перевірки часу виконання та додавши більше винятків. [10]

2.3.4 ECMAScript 6

ECMAScript 6, про який ми будемо детально дізнаватися в серії публікацій у блозі, спочатку планувалося випустити в 2013 році, але офіційно вийшов лише в червні 2015 року. ECMAScript Harmony була кодовою назвою для групи функцій ES5, але стало зрозуміло, що його цілі все ще занадто амбітні, тому TC39 вирішив розбити його на дві фази:

Найвищі пріоритетні функції будуть у наступному випуску ECMAScript. Він отримав кодову назву ECMAScript.next, щоб спробувати уникнути повного нумераційного фіаско із втраченим ECMAScript 4.

Решта функцій потраплять у наступну версію.

Коли ECMAScript.next дозрівав, кодове ім’я було скинуто, і воно почало діяти за допомогою ECMAScript 6, або просто ES6. Однак наприкінці 2014 року TC39 вирішив перейменувати ECMAScript 6 на ECMAScript 2015 у світлі нового плану випуску ECMAScript щорічно. На той час стандарт вже був добре

встановлений як ECMAScript 6, тому більшість людей досі знають його як ES6 і називають його таким.

Функції ES6 повільно впроваджуються в інтерпретатори наших сучасних браузерів та серверів. Таблиця сумісності ES6 відстежує, скільки стандартного підтримує кожен двигун. Новий браузер Microsoft від Edge насправді є лідером, підтримуючи 67% функцій ES6. Firefox 11 (випущений у березні 2012 року) був першим браузером із “значною” підтримкою (понад 10%). Транспілятори, такі як Babel і Tracer, дозволяють вам скомпілювати багато ES6 до дійсного ES5 для сумісності між браузерами. [10]

2.3.5 Строгий варіант ECMAScript

Мова ECMAScript визнає можливість того, що деякі користувачі цієї мови можуть побажати обмежити використання деяких функцій, доступних на цій мові. Вони можуть робити це в інтересах безпеки, щоб уникнути того, що вони вважають схильними до помилок, отримати посилену перевірку помилок або з інших причин, які вони обрали. На підтвердження такої можливості ECMAScript визначає суворий варіант мови.

Суворий варіант мови виключає деякі специфічні синтаксичні та семантичні особливості звичайної мови ECMAScript та модифікує детальну семантику деяких ознак. Суворий варіант також визначає додаткові умови помилок, про які потрібно повідомляти, викидаючи виключення помилок у ситуаціях, які не визначені як помилки несуворою формою мови.

Строгий режим вибору та використання строгого синтаксису режиму та семантики ECMAScript явно проводиться на рівні окремих одиниць вихідного тексту ECMAScript. Оскільки строгий режим вибирається на рівні синтаксичної одиниці текстового джерела, суворий режим лише накладає обмеження, які мають локальний ефект у межах такої одиниці вихідного тексту. Суворий режим не обмежує та не змінює жодного аспекту семантики ECMAScript, який повинен

працювати послідовно в декількох одиницях вихідного тексту. Повна програма ECMAScript може складатися як із суворого режиму, так і з нестрогого режиму вихідних текстових одиниць ECMAScript. У цьому випадку строгий режим застосовується лише тоді, коли фактично виконується код, який визначений у вихідній текстовій одиниці строгого режиму.

Для того, щоб відповідати цій специфікації, реалізація ECMAScript повинна реалізовувати як повну необмежену мову ECMAScript, так і суворий варіант мови ECMAScript, як визначено цією специфікацією. Крім того, реалізація повинна підтримувати поєднання необмежених і строгих режимів вихідних текстових одиниць в єдину складену програму. [11]

2.3.6ECMAScript 2016 - 2017

В новітніх версіях були додані нові методи

В **ECMAScript 2016 було додано:**

- Метод includes для визначення чи знаходиться той чи інший елемент в масиві, на відміну від indexOf має можливість оброблювати значення NaN
- Індексний оператор возведення в ступінь – **

В ECMAScript 2017 було додано**:**

- Метод values – функція яка повертає всі значення власних свойств об’єкта.
- Метод entries – повертає у вигляді масиву ключі та значення об’єкту.
- Об’єкти типу стрінг отримали два нових методи padStart та padEnd. З їх допомогою розробник отримує можливість додавання вказану кількість заданих символів в початок або в кінець строки.
- Конструкція Async/Await [10]

2.4 DOM

Об'єктна модель документа (DOM) - це API програмування для документів HTML і XML. Він визначає логічну структуру документів та спосіб доступу та маніпулювання ним. У специфікації DOM термін "документ" використовується в широкому розумінні - все частіше XML використовується як спосіб представлення багатьох різних видів інформації, яка може зберігатися в різних системах, і більша частина цього традиційно розглядається як дані, а не як документи. Тим не менше, XML представляє ці дані як документи, і DOM може використовуватися для управління цими даними.

За допомогою об'єктної моделі документа програмісти можуть створювати та будувати документи, орієнтуватися в їх структурі, а також додавати, змінювати або видаляти елементи та вміст. До будь-чого, що міститься в документі HTML або XML, можна отримати доступ, змінити, видалити або додати за допомогою об’єктної моделі документа, за деякими винятками - зокрема, інтерфейси DOM для внутрішньої та зовнішньої підмножини ще не вказані.[14]

Об'єктна модель документу виникла як специфікація, щоб дозволити сценаріям JavaScript та програмам Java мати смогу виконуватися в різних браузерах. Динамічний HTML був безпосереднім предком об’єктної моделі документа, і спочатку він розглядався переважно з точки зору браузерів. Однак, коли була сформована Робоча група об’єктної моделі документа, до неї також приєдналися постачальники в інших доменах, включаючи редактори HTML або XML та сховища документів. Деякі з цих постачальників працювали з SGML до розробки XML; в результаті на об’єктну модель документа вплинули SGML Groves та стандарт HyTime. Деякі з цих постачальників також розробили власні об'єктні моделі для документів, щоб надати API програмування для редакторів SGML / XML або сховищ документів, і ці об'єктні моделі також вплинули на об'єктну модель документа.[14]

2.5 V8

V8 здатен на:

- Компілює та виконує код JS
- Обробка стеку викликів - запуск функцій JS в певному порядку
- Управління розподілом пам’яті для об’єктів - купа пам’яті
- Прибирання сміття - об’єктів, які вже не використовуються
- Надає всі типи даних, оператори, об’єкти та функції
- Надає цикл подій, але це іноді реалізує і браузер[16]

V8 - це одно поточний механізм виконання. Він побудований для запуску рівно одного потоку в контексті виконання JavaScript. Ви можете запустити два двигуни V8 за один і той же процес , але вони не будуть ділитися жодними змінними чи контекстом, як справжні потоки. Це не означає, що V8 працює на одному потоці, але означає, що забезпечує потік JavaScript одного потоку.

Під час виконання V8 в основному керує розподілом кучі пам'яті та однопотоковим стеком викликів. Стек викликів - це, в основному, список функцій, які потрібно виконати за порядком виклику. Кожна функція, яка викликає іншу функцію, буде вставлена одна за одною безпосередньо, а зворотні дзвінки будуть надіслані до кінця. Ось чому насправді виклик функції з setTimeout в нуль мілісекунд надсилає її в кінець поточного рядка і не викликає її відразу (0 мілісекунд).[16]

Інші ключові компоненти:

JS Interpreter - Компілятор запалювання та оптимізації - TurboFan & Crankshaft

V8 отримує свою швидкість завдяки своєчасному (JIT) компілюванню JavaScript до власного машинного коду, безпосередньо перед його виконанням. Перш за все, код компілюється базовим компілятором, який швидко генерує не

оптимізований машинний код. Під час виконання скомпільований код аналізується і може бути перекомпільований для оптимальної роботи. Запалювання забезпечує перше, а TruboFan & Crankshaft - друге.

Машинний код результату компіляції JIT може зайняти великий обсяг пам'яті, хоча може виконуватися один раз. Це вирішується системою запалювання, яка виконує код із меншими витратами пам'яті.

Проект TurboFan розпочався у 2013 році для покращення слабкості колінчастого валу, який не оптимізований для певної частини функціональних можливостей JavaScript, наприклад обробка помилок. Він був розроблений для оптимізації як існуючих, так і майбутніх запланованих функцій на той час.

V8 Engine, є продуктом компанії Google, є відкритим кодом і написаний мовою C++. Цей механізм використовується всередині веб-браузеру Google Chrome. Однак, на відміну від решти двигунів, V8 також використовується для популярного середовища Node.js.

V8 вперше був розроблений для покращення ефективності роботи JavaScript у веб-браузерах. Для пришвидшення, V8 кодує JavaScript у більш ефективний машинний код замість використання інтерпретатора. Він компілює код JavaScript у байтовий код під час виконання, впроваджуючи компілятор JIT (Just-In-Time), як і багато сучасних движків JavaScript, таких як SpiderMonkey або Rhino (Mozilla). Основна відмінність тут полягає в тому, що V8 не виробляє байт-код або будь-який проміжний код.

Раніше у V8 було два компілятори

Перш ніж вийшла версія 5.9 V8, движок використовував два компілятори:

full-codegen - простий і дуже швидкий компілятор, який виробляє простий і відносно повільний машинний код.

Crankshaft - більш складний (точно вчасно) оптимізуючий компілятор, який створив високооптимізований код.[15]

Двигун V8 також використовує кілька потоків внутрішньо:

Основний потік робить те, що ви очікуєте: завантажте свій код, скомпілюйте його, а потім виконайте

Існує також окремий потік для компіляції, так що основний потік може продовжувати виконуватися, поки перший оптимізує код

Потік Profiler, який повідомляє про час виконання, на які методи ми витрачаємо багато часу, щоб колінчастий вал міг їх оптимізувати

Кілька ниток для обробки сміттєвих збирачів

При першому запуску коду JavaScript V8 використовує повний кодеген, який безпосередньо перетворює проаналізований JavaScript у машинний код без будь-яких перетворень. Це дозволяє йому дуже швидко почати виконувати машинний код. Зверніть увагу, що V8 не використовує проміжне представлення байт-коду таким чином, усуваючи потребу в інтерпретаторі.

Коли ваш код працює деякий час, потік профілі збирає достатньо даних, щоб визначити, який метод слід оптимізувати.

Далі, оптимізація колінчастого валу починається з іншого потоку. Він перекладає дерево абстрактного синтаксису JavaScript у подання високого рівня статичного одинарного призначення (SSA) під назвою Гідроген і намагається оптимізувати цей графік Гідрогену. Більшість оптимізацій проводиться на цьому рівні.[15]

2.6 Node.js

Перша версія Node.js яку написав Райан Далом у 2009 р. близько 30 р. з моменту впровадження мови JavaScript на стороні сервера, LiveWire від компанії Netscape. Перша версія отримала підтримку операційних систем Linux та Mac OS. Розробкою версії та її технічною підтримкою керував Райан Дал, її спонсором став Joyent.

Даль надав критику щодо обмежених можливостей найпоширенішого веб-серверу в 2009 р. , Apache сервер, одночасної обробки громадної кількості

з'єднань, максимальна кількість одночасних з'єднань досягала 10000 і більше та найпопулярніший метод написання коду , а саме послідовне програмування, метод заключався у тому що система блокувала поступаючі процеси – при підключенні в один і той самий момент різних процесів до стеків виконання відбувалося їх блокування та подальше очікування звільнення одного зі стеків для обробки потоку.

Даль презентував проект в європейському АТ "Конф" 8 2009 р. Node.js з'єднав механізми роботи двигун роботи JavaScript - V8 від компанії Google, цикл подій та програмний інтерфейс введення/виведення низького рівня.

В 2010 році було введено менеджер пакетів для програмного продукту Node.js - npm. Менеджер пакетів зпрощує публікування та надання спільного доступу до вихідного коду пакетів Node.js і за допомогою командної строки терміналу розробник отримує спрощення для встановлення потрібних пакетів середовища їх оновлення до потрібної версії, видалення пакетів та створення, налаштування і тестування нового проекту.

У 2011 році Microsoft та Joyent ввели власну версію середовища під назвою Windows Node.js. Перша версія Node.js з підтримкою компаній була випущена у 2011 році.

У 2012 році Даль передав управління розробки програмного продукту Node.js творцю npm - Ісааку Шлютеру. У 2014 році Шлютр розказав на конференції - Фонтен наступним очолить проект.

У 2014 році Федір Індутний запустив io.js, з підтримкою Node.js. В зв'язку з в конфлікт в середині компанії щодо управління, io.js був створений як окрема структура з комітетом з питань технічної підтримки . На відміну від Node.js , керівники компанії поставили на меті щорічно покращувати io.js останніми випусками механізму V8 від компанії Google.

У 2015 році оголосили про бажання сформувати нейтральний фонд Node.js. До 2015 року співтовариство Node.js та io.js об'єдналися для співпраці в рамках фонду Node.js .

У 2015 року середовище Node.js та io.js були об’єднані у Node.js v4.0. Їхне злиття принесло функції N8 ES6 V8 у Node.js та довгостроковий період технічної підтримки. Станом на 2016 рік веб-сайт io.js надає рекомендацію програмістам перейти на Node.js і не планує в майбутньому випусків io.js у зв’яку з об’єнанням.

У 2016 році випущено новітню версію Node.js 0.10.42

У 2019 році фонд JavaScript та фонд Node.js об’єдналися, для формування нового фонду під назвою OpenJS.

Node.js - це платформа, побудована на виконанні JScript в браузері Chrome для легкої побудови швидких та масштабованих мережевих програмних продуктів. Node.js використовує керовану подіями модель неблокуючого вводу-виводу, що робить її легкою та ефективною, ідеально підходить для додатків у режимі реального часу, що працюють на розподілених пристроях.

Node.js - це міжплатформене середовище виконання з відкритим кодом для розробки серверних, браузерних та мережевих додатків. Додатки Node.js написані на JS і можуть запускатися в середовищі виконання Node.js в OS, Windows та Linux.

Node.js також надає багату бібліотеку різних модулів JavaScript, що значною мірою спрощує розробку веб-додатків за допомогою Node.js.

Node.js - це середовище виконання, яке перетворює JavaScript на мову програмування загального призначення, яка може запускати програми майже в будь-якій системі. Творці Node.js взяли механізм JavaScript V8 з відкритим кодом і додали API та інструменти, необхідні для обчислень настільних комп'ютерів та серверів.

Загальне середовище програмування, наприклад Node.js, потребує таких мінімальних можливостей:

- Можливість читання та запису дискових файлів (введення / виведення диска);

- Можливість читання та запису через термінал (стандартний ввід / вивід);
- Можливість надсилання та отримання повідомлень через мережу (мережевий ввід / вивід);
- Можливість взаємодії з базою даних. [17]

Node.js має API та пакети для всіх цих завдань та багато іншого. Він також забезпечує інтерактивний REPL (цикл читання-виведення-друку), де ви можете виконувати команди JavaScript і отримувати миттєві результати. Як і будь-яке корисне середовище виконання, Node.js надає інструменти для налагодження та перевірки програм під час виконання. На жаль, інструменти налагодження та перевірки дещо важко використовувати безпосередньо; натомість вам, як правило, доводиться користуватися браузером, зокрема Google Chrome.

Інші середовища виконання JavaScript:

Окрім браузерів та Node.js, ви можете знайти JavaScript у деяких інших середовищах. Наприклад, Adobe Acrobat підтримує JavaScript для автоматизації та анімації елементів у документі. Інші середовища включають ActionScript і оболонку GNOME. [17]

2.6.1Особливості Node.js

Асинхронні та керовані події - Усі API бібліотеки Node.js є асинхронними, тобто не блокуючими. Це означає, що сервер який базується на Node.js не чекає, поки API поверне дані. Сервер переходить від одно API до іншого після його виклику, а механізм сповіщення про події Node.js допомагає серверу отримати відповідь від попереднього виклику API.

Дуже швидко - побудована на механізмі JavaScript V8 від Google Chrome, бібліотека Node.js дуже швидко виконує код.

Однопоточна, але дуже масштабована - Node.js має за основу однопоточну модель із циклічним подією. Механізм подій допомагає серверу реагувати без блокування запитів і робить сервер масштабованим на відміну від серверів

основаних на інших моделях обробки запитів, які створюють обмежені потоки для обробки запитів. Node.js використовує одну потокову програму, і ця сама програма може надати набагато більшу кількість запитів, ніж традиційні сервери, такі як Apache HTTP Server.

Без буферизації - програми Node.js ніколи не буферизують будь-які дані. Ці програми просто виводять дані шматками.

На наступній схемі зображено важливі частини Node.js,

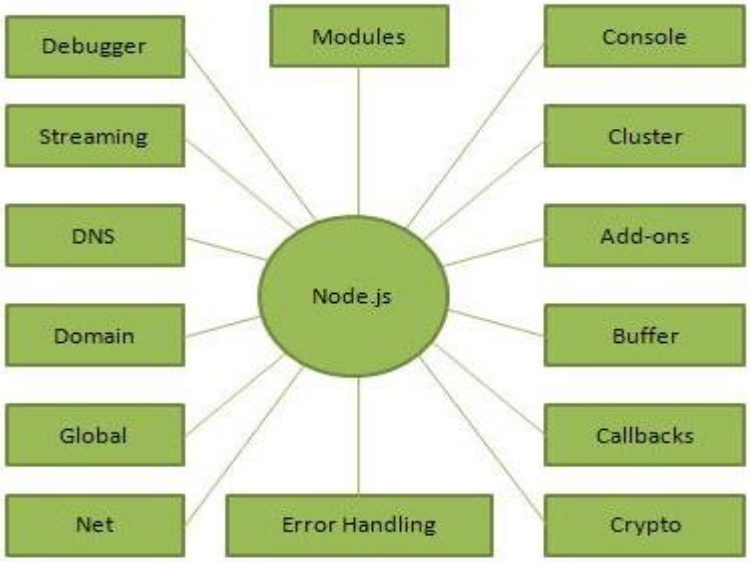


Рисунок 2.1 - схема частин Node.js

Node.js використовують для:

- Створення програмного продукту з необхідністю введення / виведення данних для подальшої обробки
- Створення сервісу з потоковим передаванням даних
- Програми що працюють в режимі реального часу (DIRT)
- Додатків що мають за основу JSON API
- Односторінкові програми [17]

2.7 React.js

React - це бібліотека JavaScript, яка спеціалізується на допомозі розробникам у створенні користувальницьких інтерфейсів або інтерфейсів. Що стосується веб-сайтів та веб-додатків, інтерфейси користувача - це набір екранних меню, рядків пошуку, кнопок та всього іншого, з чим хтось взаємодіє для використання веб-сайту чи програми.

До React JS розробники застрягли у створенні користувальницьких інтерфейсів вручну за допомогою «ванільного JavaScript» (розробник самостійно говорить за необроблену мову JavaScript) або з менш орієнтованими на UI попередниками React, такими як jQuery. Це означало довший час розробки та безліч можливостей для помилок та помилок. Так, у 2011 році інженер Facebook Джордан Уолк створив React JS

Відкривається в новій вкладці спеціально для поліпшення розробки інтерфейсу користувача.

Окрім забезпечення багаторазового коду бібліотеки React (економія часу на розробку та зменшення шансів на помилки кодування), React має дві ключові функції, які додають привабливості розробникам JavaScript:

- JSX
- Віртуальний DOM [18]

2.7.1Використання React.js

Для початку роботи з React.js потрібно створити html документ елементом з унікальним айді через нього ми будемо відображати усі наші React-компоненти.

Для створення компоненту ми використовуємо конструкцію:

```
class назва елементу extends React. Component{
constructor(){
super(props);
this.state.value = “ ”;
this.handler = this.handler.bind(this);
```

}

render(){

return()

}

}

.. у новому файлі який імпортуємо в головний файл для подальшого його відображення на сторінці. У конструкторі класу записуються усі поля змінних що використовуються в середині компоненту та методи що оброблюють значення або події всередині компоненту. Усі елементи інтерфейсу мовою html записуються в ретурні метод рендерингу наприклад для створення простого поля вводу потрібно буде створити:

<input className = “” value = “” onChange={handler}>

У className потрібно записати ім’я елементу для подальшої його обробки у CSS.

Значення value може бути як вказане завчасно так і прирівняне до значення поля класу для подальшої його обробки та зручності взаємодії з даними інтерфейсу.

В onChange – це подія зміни стану елементу на яку ми вказуємо обробник події який відслідковує даний елемент. Його дій ми зазначаємо в методах компоненту.Також кожен обробник підписують методом bind до контексту компоненту. Обробник подій виконує більшість дій на стороні клієнту робить перевірки даних, їх присвоєння та відправлення на сервер. Наприклад для присвоєння значення введеного користувачем полю нашого компонента за умови що в блоці вводу вказані такі данні <input className = “” value = “this.state.value” onChange={this.handler}> потрібно в методі прописати що дані які приходять на подію зміни блоку присвоюються методом setState полю нашого компоненту. Це виглядатиме таким чином:

handler(event){

this.setState({value: event.target.value});

}

А для того щоб відправити дані з клієнту на сервер наприклад на натискання кнопки нам потрібно створити ще один обробник події. Він виглядатиме таким чином:

Потрібно створите новий обробник подій нехай його назва буде handlerRequest та підключити його до події на яку він буде реагувати

handlerRequest(event){

Прописати метод інтерфейсу для створення HTTP запитів та подальшої обробки запитів що відправляє та отримує клієнт.

fetch(

В методі потрібно вказати шлях на який відправляється запит

‘шлях’,

{

Далі потрібно вказати тіло запросу в якому вказується

Метод запросу POST GET PUT DELETE та інші доступні методи. Наприклад

method: “POST”;

Оглавлення запросу в якому вказується тип даних який відправляється. Це виглядатиме таким чином:

Header: {

Accept:'Content-Type': 'application/json'

}

Та основну частину в якій відправляємо дані на сервер. Наприклад відправимо поле “value” з нашого компоненту

Body: JSON.stringify({

```
value: this.state.value,
})
})
```

За допомогою чейнінгу викликаємо очікування та обробку даних з серверу та обробку помилок. Для прикладу отримані дані запишемо в поле “value” метод запису setState

```
.then((res) => res.json())
.then((data) => this.setState(value: data))
.catch((err) => console.log(err));
 }
```

2.8 JSON

Що таке JSON?

JSON або JavaScript Object Notation - це безсхемове представлення структурованих даних у вигляді тексту в читабельному форматі для даних, яке представляє собою об’єкт, що вміщує в себе пари вигляду “ключ: значення”. Використання JSON полягає у передачі даних між серверною частиною та клієнтом, як альтернатива XML. Squarespace використовує JSON для зберігання та організації вмісту сайту, створеного за допомогою системи управління вмістом.

Хоча JSON походить від JavaScript, він підтримується або вбудовано, або через бібліотеки на більшості основних мов програмування.

Сьогодні JSON це формат, який вибирають майже для кожного загальнодоступного веб-додатку, і він часто використовується також для приватних веб-служб.

Популярність JSON також призвела до вбудованої підтримки JSON у багатьох базах даних. Реляційні бази даних, такі як PostgreSQL та MySQL, тепер мають вбудовану підтримку для зберігання та запиту даних JSON. Бази даних

NoSQL, такі як MongoDB та Neo4j, також підтримують JSON, хоча MongoDB використовує трохи модифіковану двійкову версію JSON за кадром.[19]

2.8.1Ключі та значення

Дві основні частини, що складають JSON, - це ключі та значення. Разом вони складають пару ключ / значення.

Ключ: Ключ - це завжди рядок, укладений у лапки.

Значення: Значення може бути рядком, числом, логічним виразом, масивом або об’єктом.

Пара ключ / значення: пара значень ключ слідує певному синтаксису, за ключем слідує двокрапка і значення. Пари ключ / значення розділяються комами.

Наприклад:

"foo": "bar"

Цей приклад - пара ключ / значення. Ключ - "foo", а значення - "bar".

Типи значень:

Масив: Асоціативний масив значень.

Логічне значення: Істинно чи хибно.

Число: ціле число.

Об’єкт: Асоціативний масив пар ключ / значення.

Рядок: Кілька символів простого тексту, які зазвичай утворюють слово.

Числа, булеві значення та рядки зрозумілі самі собою. Масиви та об'єкти використовуються таким чином.

Масиви

Майже в кожному блозі є категорії та теги. У цьому прикладі ми додали ключ категорій, але значення може виглядати незнайомим. Оскільки кожна публікація в блозі може мати більше однієї категорії, повертається масив із декількох рядків.

```
"foo": {
 "baz": ["quuz", "norf"]
}
```

Об'єкти

Об’єкт позначається фігурними дужками. Все, що знаходиться всередині фігурних дужок, є частиною об’єкта. Ми вже дізналися, що цінність може бути об'єктом. Отже, це означає, що "foo" та відповідний об'єкт є парою ключ / значення.[19]

```
"foo" : {
 "bar" : "Hello" }
```

2.9 Express.js

Що таке Express.js?

Express.js – це фреймворк веб-додатків з відкритим кодом для Node.js. Він використовується для швидкого та простого проектування та створення веб-додатків. Оскільки Express.js вимагає лише javascript, програмістам та розробникам стає простіше створювати веб-програми та API.

Express.js - це фреймворк Node.js, що означає, що більша частина коду вже написана для роботи програмістів. Ви можете створити односторінкові, багатосторінкові або гібридні веб-програми за допомогою Express.js. Він є легким і допомагає організувати веб-програми на стороні сервера в більш організовану архітектуру MVC.

Express.js полегшує управління веб-додатками. Це частина технології на основі JavaScript, яка називається MEAN стек програмного забезпечення, що

розшифровується як MongoDB, ExpressJS, AngularJS та Node.js. Express.js є внутрішньою частиною MEAN і управляє маршрутизацією, сеансами, HTTP-запитами, обробкою помилок тощо. Бібліотека JavaScript Express.js допомагає програмістам створювати ефективні та швидкі веб-програми. Express.js покращує функціональність node.js. Він спростив програмування на node.js і надав безліч додаткових функцій.[22]

2.9.1 Переваги Express.js

Express.js має наступні переваги:

Значно Спрощує розробку серверних платформ для розробників, які використовують Express.js

Веб-розробник може використовувати JavaScript як єдину мову для серверу та клієнту веб-додатку.

Код JavaScript інтерпретується через Google V8 JavaScript Engine від Node.js. Отже, код реалізується швидко і легко, ефективно.

Фреймворк Express.js дуже простий у налаштуванні та використанні.

Express.js надає гнучкий модуль проміжного програмного забезпечення. Це в основному корисно для виконання додаткових завдань за відповіддю та запитом.[22]

Маршрутизація - це процес який очікує запит за певною URL-адресою від клієнту сервером з певним методом запросу.

Структурна схема запросу на маршрутизатор виглядає таким чином:

App.method(path, handler) в якому:

- App – це об’єкт додатку фреймворку Express
- Method – метод запросу який очікую сервер він поділяється на (GET,PUT,POST,DELETE та інші)

- Path – це шлях на який очікується запит він виглядає таким чином: “/шлях”
- Handler – це функція обробки запиту який очікує наш маршрутизатор та подальших дій серверу. В більшості випадків вона має два аргументи перший request – це дані які відправив клієнт на сервер та другий аргумент response – дані які відправляє сервер на клієнт. [21]

Також в Express.js використовується система раутингу вона полягає у спрощені написання коду та підвищення його зрозумілості.

Спрощення полягає у прив’язці запросів до певної адреси кожного раутингу. Наприклад розробник створив раутинг sheepRouter на адресу http://localhost:3000/sheep/ тоді він підписує засобами express. Js раутер до цієї адреси таким чином:

```
app.use(‘/sheep’, sheepRouter);
```

і усі запити з /sheep відслідковуються цим раутингом. Для обробки потрібно лише дописати до раутингу потрібний метод запиту та вказати адресу(у випадку вказаної вище адреси в раутингу вона виглядатиме як “/”) і обробник запиту. Це виглядатиме таким чином:

```
router.get(‘/’, function(req, res){
    res.statusCode = 200;
    res.send(‘eating’);
})
```

Для додавання переходу по цій адресі далі наприклад: http://localhost:3000/sheep/grass потрібно лише змінити шлях в раутингу на /grass і додати необхідний метод та обробник. В коді це виглядатиме таким чином:

```
router.get(‘/grass’, function(req, res){
```

```
        res.statusCode = 200;
        res.send('eating');
})
```

2.10 CSS

Що таке CSS?

CSS означає «Каскадні таблиці стилів» з акцентом на «Стиль». Хоча HTML використовується для структурування веб-сторінки (визначаючи такі речі, як заголовки та абзаци та дозволяючи вставляти зображення, відео та інші засоби масової інформації), CSS проходить і визначає стиль документа - макети сторінок, кольори та шрифти визначаються за допомогою CSS. [20]

Як працює CSS?

CSS привносить стиль у веб-сторінки, взаємодіючи з елементами HTML. Елементи - це окремі HTML-компоненти веб-сторінки - наприклад, абзац - які в HTML можуть виглядати так:

```
<p> Абзац! </p>
```

Для того щоб абзац вище виглядав рожевим і жирним для людей, які переглядають веб-сторінку через браузер, потрібно вписати поля CSS, які виглядають таким чином:

```
p { color:pink; font-weight:bold; }
```

У цьому випадку "p" (абзац) називається "селектором" - це частина коду CSS, яка вказує, на який елемент HTML вплине стилістика CSS. У CSS селектор пишеться зліва від першої фігурної дужки. Інформація між фігурними дужками називається декларацією, і вона містить властивості та значення, які застосовуються до селектора. Властивості - це речі, такі як розмір шрифту, колір та поля, тоді як значення - це налаштування для цих властивостей. У наведеному вище прикладі властивості мають значення "колір" та "шрифт-шрифт", а значення "рожевий" та "жирний" Повний набір в дужки[20]

{ color:pink; font-weight:bold; }

- це декларація, і знову ж “p” (мається на увазі абзац HTML) - селектор. Ці самі основні принципи можна застосовувати для зміни розміру шрифту, кольорів тла, відступів полів тощо. Наприклад. . .

body {background-color: lightblue; }

Фон сторінки зробиться блакитним, або

p { font-size:20px; color:red; }

створить абзац із 20 крапковими шрифтами з червоними літерами.[20]

2.10 .1Зовнішній, внутрішній або вбудований CSS

Подібно до HTML, CSS пишеться простим текстом через текстовий редактор або текстовий процесор на вашому комп'ютері, і є три основні способи додати цей CSS-код на HTML-сторінки. Код CSS (або Таблиці стилів) може бути зовнішнім, внутрішнім або вбудованим. Зовнішні таблиці стилів зберігаються у форматі .css-файлів і можуть використовуватися для визначення зовнішнього вигляду цілого веб-сайту за допомогою одного файлу (замість того, щоб додавати

окремі екземпляри коду CSS до кожного елемента HTML, який потрібно налаштувати). Для того, щоб використовувати зовнішню таблицю стилів, ваші файли .html повинні містити розділ заголовка, який посилається на зовнішню таблицю стилів і виглядає приблизно так:

```
<head>
<link rel=”stylesheet”  type=”text/css”  href=mysitestyle.css”>
</head>
```

Це прив’яже файл .html до зовнішньої таблиці стилів (у цьому випадку mysitestyle.css), і всі інструкції CSS у цьому файлі будуть застосовуватися до ваших пов’язаних сторінок .html.[20]

Внутрішні таблиці стилів - це інструкції CSS, записані безпосередньо в заголовок певної сторінки .html. (Це особливо корисно, якщо у вас є одна сторінка на сайті, яка має унікальний вигляд.) Внутрішня таблиця стилів виглядає приблизно так. . .

```
<head>
<style>
Body  {  background-color:thistle;  }
P  {  font-size:20px;  color:mediumblue;  }
</style>
</head>
```

тепер до цієї єдиної сторінки .html буде застосовано колір тла чертополоху та абзаци із 20-крапковим, середньо-синім шрифтом.

Нарешті, вбудовані стилі - це фрагменти CSS, записані безпосередньо в HTML-код, і застосовуються лише до одного екземпляра кодування. Наприклад:

```
<h1  style=”font-size:40px;color:violet;”>Check out this headline!</h1>
```

призведе до того, що один конкретний заголовок на одній сторінці .html з’явиться фіолетовим шрифтом із 40 крапками.

Взагалі кажучи, зовнішні таблиці стилів є найефективнішим методом реалізації CSS на веб-сайті (простіше відстежувати та реалізовувати стиль сайту із виділеного CSS-файлу), тоді як внутрішні таблиці стилів та вбудований стиль можуть бути використані у випадку випадок, коли потрібно внести індивідуальні зміни стилю.

Отже, якщо HTML є основою, рамами, стінами та прогонами, що підтримують ваш веб-сайт, слід розглядати CSS як колір фарби, стилі вікон та озеленення, що з’являється згодом. [20]

ВИСНОВОК ДО ДРУГОГО РОЗДІЛУ

В другому розділі були розглянуті засоби, які були обрані для розробки серверної та клієнтської частини шифрувальної машини Енігма. Обраними засобами для створення серверної частини є середовище Node.js та програмний каркас розробки серверної частин Express.js .

Для створення клієнтської частини було обрано бібліотеку для створення користувацького інтерфейс React.js та мова стилізації веб-застосунків CSS.

Передача даних з клієнту на сервер відбувається шляхом відправки JSON файлу з усіма необхідними полями для обробки сервером.

Основними плюсами обраних засобів були полегшення розробки та значна кількість документації для проекту.

Засоби програмування були обрані для реалізації поставленої цілі, а саме створення шифрувальної машини Енігма з простим в розумінні для користувача інтерфейсом.

РОЗДІЛ 3

3.1 Інструкція користувача

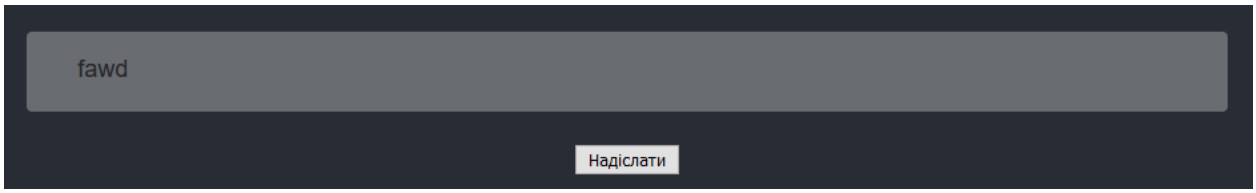


Рисунок 3.1 - Інтерфейс вводу повідомлення

Для початку користувачу потрібно ввести повідомлення, яке він хоче зашифрувати у відповідний блок запису та нитиснути кнопну відправки повідомлення “Надіслати”.

Оброблений текст записується у відповідне вікно яке до наведення на блок зливається з сайтом і виглядає таким чином:

Рисунок 3.2 - Інтерфейс виводу зашифрованого повідомлення без наведення на блок.

При наведенні на блок для зручності він змінює колір і виглядає таким чином:

Рисунок 3.3 – Інтерфейс виводу зашифрованого повідомлення з наведенням на блок.

Також користувач може встановити власні налаштування кроку ротору, типи рефлектору(B,C)

Кожен рефлектор відповідає даному набору символів:

Reflector B: YRUHQSLDPXNGOKMIEBFZCWVJAT

Reflector C: FVPJIAOYEDRZXWGCTKUQSBNMHL

та кожного з доступних роторів(I,II,III).

Кожен ротор відповідає даному набору символів:

I: EKMFLGDQVZNTOWYHXUSPAIBRCJ

II: AJDKSIRUXBLHWTMCQGZNPYFVOE

III: BDFHJLCPRTXVZNYEIWGAKMUSQO

Також крок ротора збільшується при кожному введеному символу.

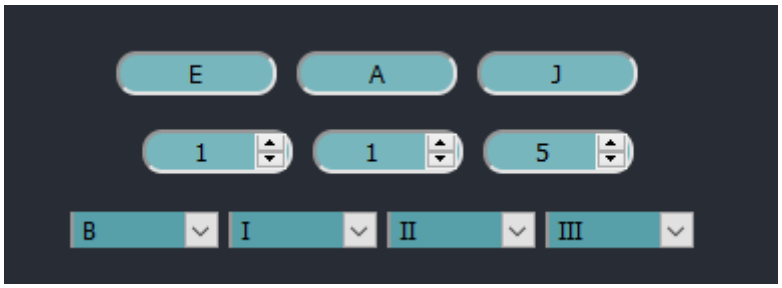


Рисунок 3.4 – Інтерфейс зміни кроку ротору і типу ротора та рефлектору

За потреби користувач може змінювати налаштування плати розширення встановлюючи потрібні пари взаємозамінних ключів.

Рисунок 3.5 – Інтерфейс вводу пар ключів плати розширення

ВИСНОВОК ДО ТРЕТЬОГО РОЗДІЛУ

В третьому розділі була розглянута інструкція користувача. Створений програмний продукт має простий в використанні та інтуїтивно зрозумілий інтерфейс, який надає можливість користувачу просто і швидко зашифрувати потрібні йому данні.

Також були розглянуті можливі позиції користувацьких налаштувань їх відображення у виді тексту з 26 літер відповідно до вибраного поля та деякі особливості дизайну.

ВИСНОВОК

Результатом бакалаврської роботи є веб-додаток для шифрування даних алгоритмом шифрування машини Енігма. Створена програма легка у використанні, не потребує додаткових роз'яснень в використанні та швидко оброблює дані користувача.

Створена веб версія машини Енігма надає користувачу можливість зашифрувати або розшифрувати дані також за потреби користувач має можливість налаштування роботи машини, а саме налаштування типу роторів та рефлектору, кроку ротора та встановлення потрібних пар взаємозаміни на платі розширення. Дані які користувач вводить для шифрування або розшифрування відправляються на сервер де обробляються і відправляється відповідь серверу з зашифрованими даними на інтерфейс користувача.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Re-engineering the Enigma cipher. - Re-engineering the Enigma cipher..pdf [Електронний ресурс] Режим доступу : https://ir.library.louisville.edu/cgi/viewcontent.cgi?article=2388&context=etd
2. Enigma Simulator · Matheus Portela[Електронний ресурс] Режим доступу: https://matheusportela.com/enigma/
3. Practical Cryptography [Електронний ресурс] Режим доступу: http://practicalcryptography.com/ciphers/enigma-cipher/
4. Universal Enigma - Simulator by dp [Електронний ресурс] Режим доступу: https://people.physik.hu-berlin.de/~palloks/js/enigma/enigma-u_v25_en.html
5. JS Enigma Machine [Електронний ресурс] Режим доступу: http://trys.github.io/Enigma/
6. Enigma Machine Cipher - Decoder, Encoder, Solver, Translator [Електронний ресурс] Режим доступу : https://www.dcode.fr/enigma-machine-cipher
7. Enigma Machine by 101Computing.net [Електронний ресурс] Режим доступу: https://www.101computing.net/enigma/enigma-M3.html
8. A Brief History of JavaScript | SpringerLink [Електронний ресурс] Режим доступу: https://link.springer.com/chapter/10.1007%2F978-1-4302-3390-9_1
9. History of ECMAScript | Ben Ilegbodu [Електронний ресурс] Режим доступу : https://www.benmvp.com/blog/learning-es6-history-of-ecmascript/
10. Обзор новшеств ECMAScript 2016, 2017, и 2018 с примерами / Блог компании RUVDS.com / Хабр [Електронний ресурс] Режим доступу: https://habr.com/ru/company/ruvds/blog/353174/
11. ECMAScript® 2020 Language Specification [Електронний ресурс] Режим доступу: https://262.ecma-international.org/11.0/#sec-intro
12. JavaScript: A History for Beginners | Course Report [Електронний ресурс] Режим доступу: https://www.coursereport.com/blog/history-of-javascript
13. An introduction to JavaScript Programming and the history of JavaScript. [Електронний ресурс] Режим доступу: https://launchschool.com/books/javascript/read/introduction#briefhistory
14. Введение - Интерфейсы веб API | MDN [Електронний ресурс] Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/Document_Object_Model/Introduction

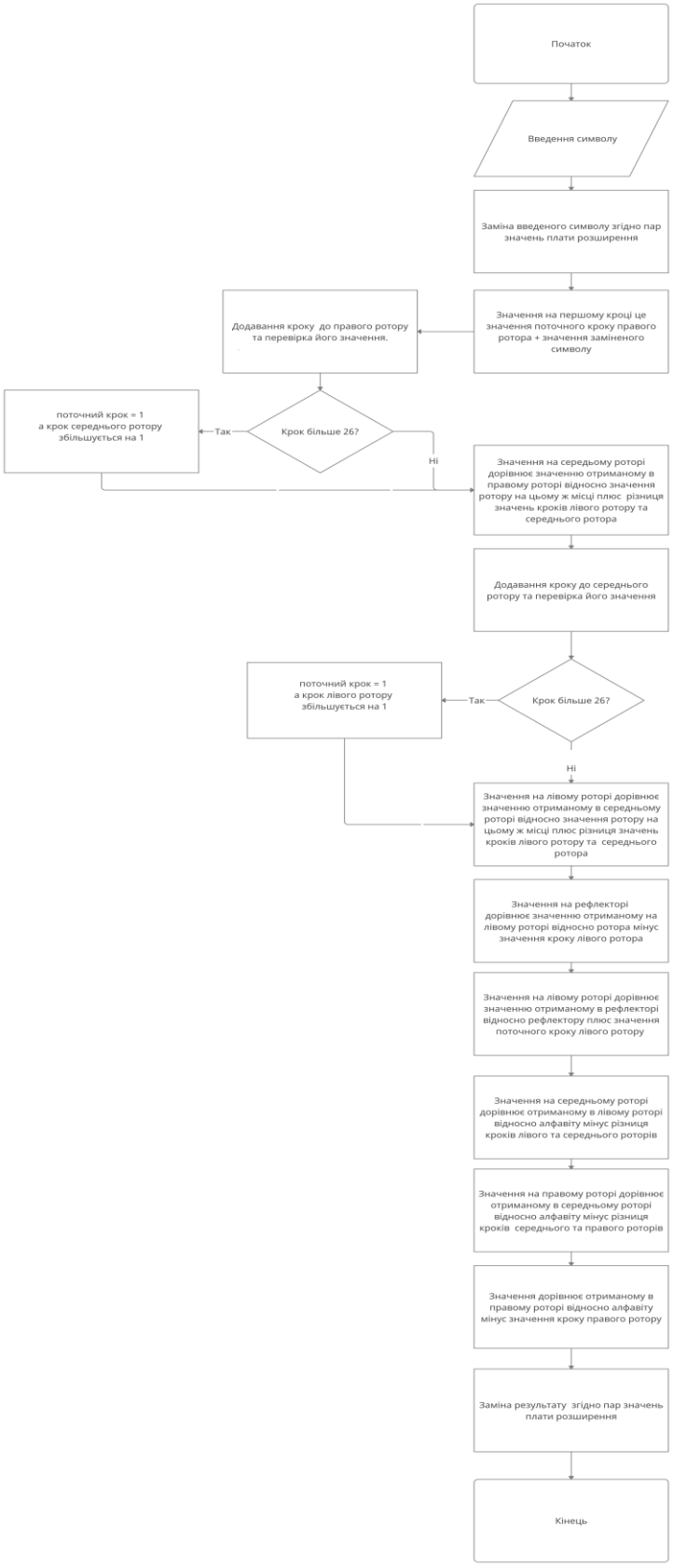
15. JavaScript V8 Engine Explained | Hacker Noon [Електронний ресурс] Режим доступу: https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef
16. How JavaScript works: inside the V8 engine + 5 tips on how to write optimized code | by Alexander Zlatkov | SessionStack Blog [Електронний ресурс] Режим доступу: https://blog.sessionstack.com/how-javascript-works-inside-the-v8-engine-5-tips-on-how-to-write-optimized-code-ac089e62b12e
17. Node.JS Use Case: When & How Node.JS Should be Used | Simform [Електронний ресурс] Режим доступу: https://www.simform.com/nodejs-use-case/
18. Початок роботи – React [Електронний ресурс] Режим доступу: https://uk.reactjs.org/docs/getting-started.html
19. What is JSON? A better format for data exchange | InfoWorld [Електронний ресурс] Режим доступу: https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html
20. What is CSS, How Does It Work and What is It Used For? - Skillcrush [Електронний ресурс] Режим доступу: https://skillcrush.com/blog/css/
21. Маршрутизация в Express [Електронний ресурс] Режим доступу: https://expressjs.com/ru/guide/routing.html
22. What is Express.js? | Why should use Express.js? | Features of Express.js [Електронний ресурс] Режим доступу: https://www.besanttechnologies.com/what-is-expressjs

ДОДАТОК 1
Веб-сервіс шифрування на базі алгоритму Енігма

Структурна схема. Блок-схема алгоритму
ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2020р



					ІАЛЦ.467200.004 Д1
Зм	*Лист*	***№ докум.***	***Підпис***	***Дата***	
Розробив		Мещеряков О.			Веб-сервіс шифрування на базі алгоритму Енігма
Перевірив		*АлещенкоО.В.*			
Н. Контр		Сімоненко В. П.			
Затверд.		Стіренко С.Г.			

Літ.	*Арк*	*Аркушів*
	1	1

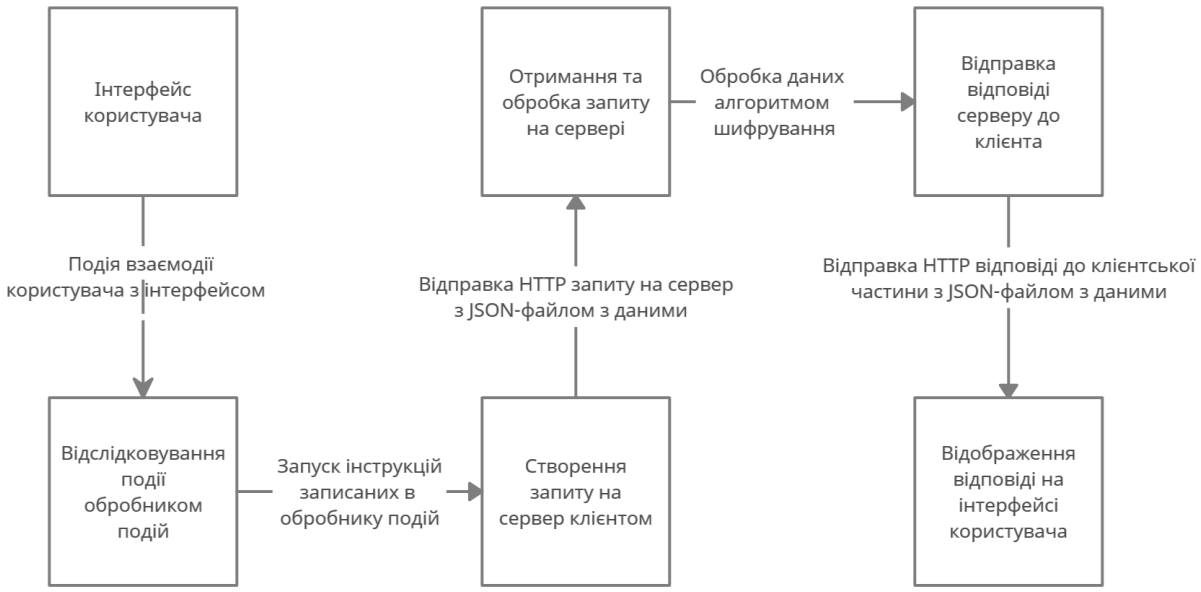
КПІ ім. Ігоря Сікорського
ФІОТ ІП-74

ДОДАТОК 2
Веб-сервіс шифрування на базі алгоритму Енігма

Схема Взаємодії
ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2021р.



					ІАЛЦ.467200.005 Д2	
Зм	*Лист*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Мещеряков О.			Веб-сервіс шифрування на базі алгоритму Енігма	*Літ.* / *Арк* / *Аркушів*: 1 / 1
Перевірив		*АлещенкоО.В.*				*КПІ ім. Ігоря Сікорського ФІОТ ІП-74*
Н. Контр		Сімоненко В. П.				
Затверд.		Стіренко С.Г.				

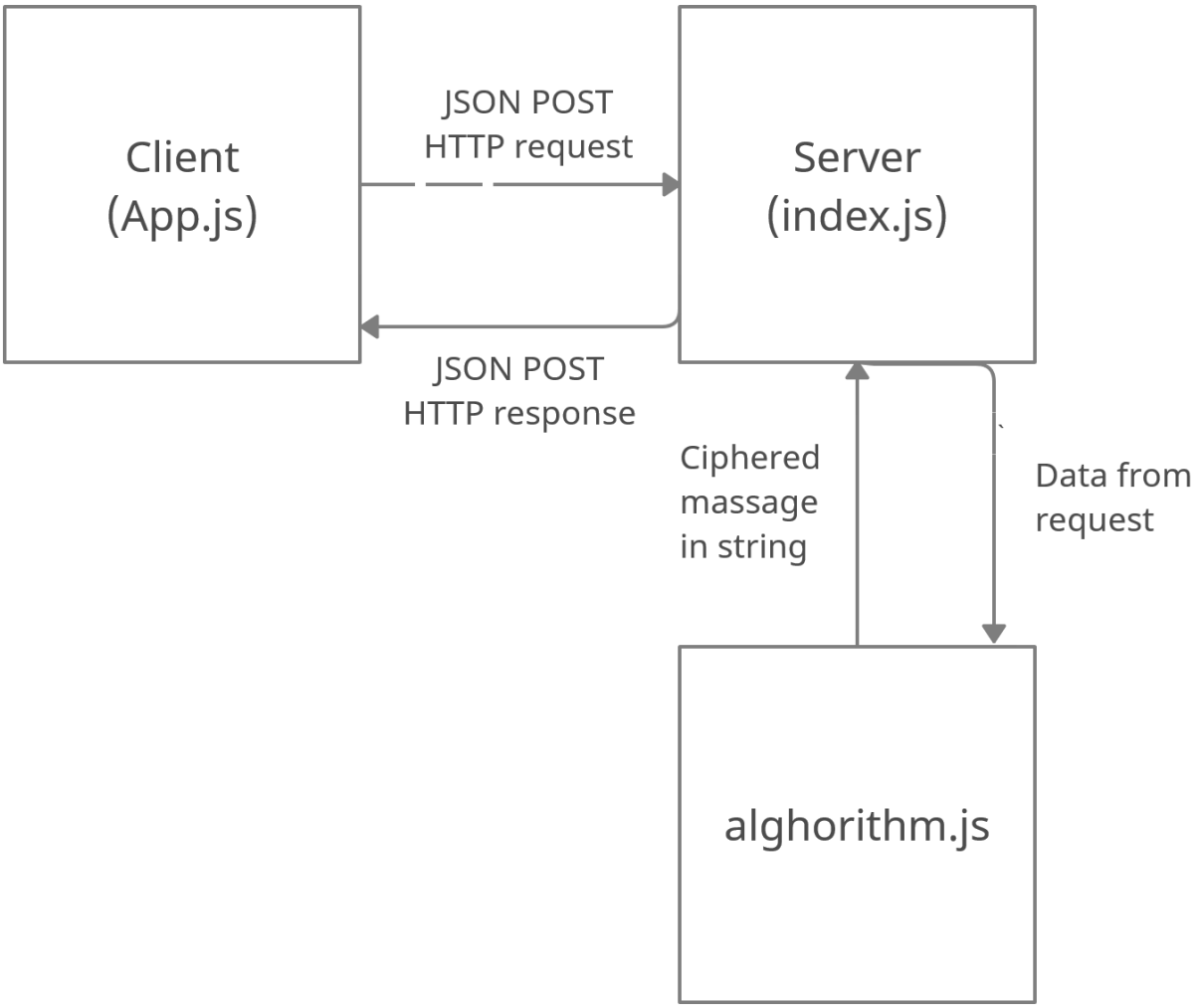
ДОДАТОК 3
Веб-сервіс шифрування на базі алгоритму Енігма

Функціональна схема. Взаємодія модулів програми

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2021р.



					ІАЛЦ.467200.006 Д3	
Зм	*Лист*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Мещеряков О.			Веб-сервіс шифрування на базі алгоритму Енігма	*Літ.* / *Арк* / *Аркушів*: 1 / 1
Перевірив		*АлещенкоО.В.*				
Н. Контр		Сімоненко В. П.				*КПІ ім. Ігоря Сікорського ФІОТ ІП-74*
Затверд.		Стіренко С.Г.				

ДОДАТОК 4

Веб-сервіс шифрування на базі алгоритму Енігма

Ключові елементи коду програми

Аркушів 12

Київ 2021 р.

fieldInput.js

```
import React from 'react';
import './fieldInput.css';
import './output.css';
import './alphabet.css';
import './num.css';
import './rotor.css';
const option = [];
const first = '9EKMFLGDQVZNTOWYHXUSPAIBRCJ'.split('');
const second = '9AJDKSIRUXBLHWTMCQGZNPYFVOE'.split('');
const third = '9BDFHJLCPRTXVZNYEIWGAKMUSQO'.split('');
let optionsLeftRotor = first;
let optionsMiddleRotor = second;
let optionsRightRotor = third;
export default class Input extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: '',
      out: '',
      stepLR: '1',
      stepMR: '1',
      stepRR: '1',
      stepLeftA: 'E',
      stepMiddleA: 'A',
      stepRightA: 'B',
      plugboard_1: 'AW',
      plugboard_2: 'BZ',
      plugboard_3: 'CY',
      plugboard_4: 'DX',
      plugboard_5: 'EV',
      plugboard_6: 'FU',
      plugboard_7: 'GT',
      plugboard_8: 'HS',
      plugboard_9: 'IR',
      plugboard_10: 'JQ',
      plugboard_11: 'KP',
      plugboard_12: 'LO',
      plugboard_13: 'MN',
      reflector: 'B',
      rotorLeft: 'I',
      rotorMiddle: 'II',
      rotorRight: 'III',
    };
    this.handleChange1 = this.handleChange1.bind(this);
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.handleRequest = this.handleRequest.bind(this);
    this.handleChangeStepL = this.handleChangeStepL.bind(this);
    this.handleChangeStepM = this.handleChangeStepM.bind(this);
    this.handleChangeStepR = this.handleChangeStepR.bind(this);
    this.handlerChangeLettersStep = this.handlerChangeLettersStep.bind(this);
    this.handleChangePlug1 = this.handleChangePlug1.bind(this);
    this.handleChangePlug2 = this.handleChangePlug2.bind(this);
    this.handleChangePlug3 = this.handleChangePlug3.bind(this);
    this.handleChangePlug4 = this.handleChangePlug4.bind(this);
    this.handleChangePlug5 = this.handleChangePlug5.bind(this);
    this.handleChangePlug6 = this.handleChangePlug6.bind(this);
    this.handleChangePlug7 = this.handleChangePlug7.bind(this);
```

```
  this.handleChangePlug8 = this.handleChangePlug8.bind(this);
  this.handleChangePlug9 = this.handleChangePlug9.bind(this);
  this.handleChangePlug10 = this.handleChangePlug10.bind(this);
  this.handleChangePlug11 = this.handleChangePlug11.bind(this);
  this.handleChangePlug12 = this.handleChangePlug12.bind(this);
  this.handleChangePlug13 = this.handleChangePlug13.bind(this);
  this.handleReflector = this.handleReflector.bind(this);
  this.handleRotorLeft = this.handleRotorLeft.bind(this);
  this.handleRotorMiddle = this.handleRotorMiddle.bind(this);
  this.handleRotorRight = this.handleRotorRight.bind(this);
}

handlerChangeLettersStep(event) {
  this.setState({step: event.target.value});
}
handleChange1(event) {
  this.setState({out: event.target.value});
}
handleChange(event) {
  this.setState({value: event.target.value});
  let next = 1 + Number(this.state.stepRR);
  this.setState({stepRR: next});
  this.setState({
    stepRightA: optionsRightRotor[Number(this.state.stepRR) + 1],
  });
  if (Number(this.state.stepRR) >= 27) {
    this.setState({stepRR: 1});
    this.setState({stepMR: Number(this.state.stepMR) + 1});
    this.setState({
      stepMiddleA: optionsMiddleRotor[Number(this.state.stepMR) + 1],
    });
  }
  if (Number(this.state.stepMR) >= 27) {
    this.setState({stepMR: 1});
    this.setState({stepLR: Number(this.state.stepLR) + 1});
    this.setState({
      stepLeftA: optionsLeftRotor[Number(this.state.stepLR) + 1],
    });
  }
}

handleSubmit(event) {
  event.preventDefault();
}

handleChangeStepL(event) {
  if (event.target.value === '27') {
    alert('Достигнут лимит');
  }
  this.setState({stepLR: event.target.value});
  this.setState({stepLeftA: optionsLeftRotor[event.target.value]});
}

handleChangeStepM(event) {
  if (event.target.value === '27') {
    event.target.value = '1';
    let next = Number(this.state.stepLR) + 1;
    this.setState({stepLR: next});
    this.setState({
      stepLeftA: optionsLeftRotor[Number(this.state.stepLR) + 1],
    });
  }
  this.setState({stepMR: event.target.value});
```

```
  this.setState({stepMiddleA: optionsMiddleRotor[event.target.value]});
}

handleChangeStepR(event) {
  if (event.target.value === '27') {
    event.target.value = '1';
    let next = 1 + Number(this.state.stepMR);
    this.setState({stepMR: next});
    this.setState({
      stepMiddleA: optionsMiddleRotor[Number(this.state.stepMR) + 1],
    });
  }
  this.setState({stepRR: event.target.value});
  this.setState({stepRightA: optionsRightRotor[event.target.value]});
}

handleReflector(event) {
  this.setState({reflector: event.target.value});
}

handleRotorLeft(event) {
  if (event.target.value === 'I') {
    optionsLeftRotor = first;
    this.setState({stepLeftA: optionsLeftRotor[this.state.stepLR]});
  } else if (event.target.value === 'II') {
    optionsLeftRotor = second;
    this.setState({stepLeftA: optionsLeftRotor[this.state.stepLR]});
  } else if (event.target.value === 'III') {
    optionsLeftRotor = third;
    this.setState({stepLeftA: optionsLeftRotor[this.state.stepLR]});
  }
  this.setState({rotorLeft: event.target.value});
}
handleRotorMiddle(event) {
  if (event.target.value === 'I') {
    optionsMiddleRotor = first;
    this.setState({stepMiddleA: optionsMiddleRotor[this.state.stepMR]});
  } else if (event.target.value === 'II') {
    optionsMiddleRotor = second;
    this.setState({stepMiddleA: optionsMiddleRotor[this.state.stepMR]});
  } else if (event.target.value === 'III') {
    optionsMiddleRotor = third;
    this.setState({stepMiddleA: optionsMiddleRotor[this.state.stepMR]});
  }
  this.setState({rotorMiddle: event.target.value});
}
handleRotorRight(event) {
  if (event.target.value === 'I') {
    optionsRightRotor = first;
    this.setState({stepRightA: optionsRightRotor[this.state.stepRR]});
  } else if (event.target.value === 'II') {
    optionsRightRotor = second;
    this.setState({stepRightA: optionsRightRotor[this.state.stepRR]});
  } else if (event.target.value === 'III') {
    optionsRightRotor = third;
    this.setState({stepRightA: optionsRightRotor[this.state.stepRR]});
  }
  this.setState({rotorRight: event.target.value});
}

handleChangePlug1(event) {
  this.setState({plugboard_1: event.target.value});
}
```

```
handleChangePlug2(event) {
  this.setState({plugboard_2: event.target.value});
}
handleChangePlug3(event) {
  this.setState({plugboard_3: event.target.value});
}
handleChangePlug4(event) {
  this.setState({plugboard_4: event.target.value});
}
handleChangePlug5(event) {
  this.setState({plugboard_5: event.target.value});
}
handleChangePlug6(event) {
  this.setState({plugboard_6: event.target.value});
}
handleChangePlug7(event) {
  this.setState({plugboard_7: event.target.value});
}
handleChangePlug8(event) {
  this.setState({plugboard_8: event.target.value});
}
handleChangePlug9(event) {
  this.setState({plugboard_9: event.target.value});
}
handleChangePlug10(event) {
  this.setState({plugboard_10: event.target.value});
}
handleChangePlug11(event) {
  this.setState({plugboard_11: event.target.value});
}
handleChangePlug12(event) {
  this.setState({plugboard_12: event.target.value});
}
handleChangePlug13(event) {
  this.setState({plugboard_13: event.target.value});
}

handleRequest(event) {
  event.preventDefault();
  fetch(
    'http://localhost:3006/',
    {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        value: this.state.value,
        stepLR: this.state.stepLR,
        stepMR: this.state.stepMR,
        stepRR: this.state.stepRR,
        plugboard_1: this.state.plugboard_1,
        plugboard_2: this.state.plugboard_2,
        plugboard_3: this.state.plugboard_3,
        plugboard_4: this.state.plugboard_4,
        plugboard_5: this.state.plugboard_5,
        plugboard_6: this.state.plugboard_6,
        plugboard_7: this.state.plugboard_7,
        plugboard_8: this.state.plugboard_8,
        plugboard_9: this.state.plugboard_9,
        plugboard_10: this.state.plugboard_10,
        plugboard_11: this.state.plugboard_11,
```

```
        plugboard_12: this.state.plugboard_12,
        plugboard_13: this.state.plugboard_13,
        reflector: this.state.reflector,
        rotorLeft: this.state.rotorLeft,
        rotorMiddle: this.state.rotorMiddle,
        rotorRight: this.state.rotorRight,
      }),
    }
  )
    .then((res) => res.json())
    .then((data) => this.setState({out: data}))
    .catch((err) => console.log(err));
}

render() {
  return (
    <form className="form" onSubmit={this.handleRequest}>
      <div className="output">
        <input
          className=""
          value={this.state.out}
          onChange={this.handleChange1}
        />
      </div>

      <div className="steps">
        <input
          className="alphabet"
          maxLength="1"
          type="text"
          value={this.state.stepLeftA}
          onChange={this.handlerChangeLettersStep}
        />
        <input
          className="alphabet"
          maxLength="1"
          type="text"
          value={this.state.stepMiddleA}
          onChange={this.handlerChangeLettersStep}
        />
        <input
          className="alphabet"
          maxLength="1"
          type="text"
          value={this.state.stepRightA}
          onChange={this.handlerChangeLettersStep}
        />
      </div>

      <div className="details">
        <input
          className="num"
          min="1"
          max="27"
          maxLength="2"
          type="number"
          value={this.state.stepLR}
          onChange={this.handleChangeStepL}
        />

        <input
          className="num"
          min="1"
```

```
    max="27"
    maxLength="2"
    type="number"
    value={this.state.stepMR}
    onChange={this.handleChangeStepM}
   />

   <input
    className="num"
    min="1"
    max="27"
    maxLength="2"
    type="number"
    value={this.state.stepRR}
    onChange={this.handleChangeStepR}
   />
  </div>

  <select className="r" onChange={this.handleReflector}>
   <option className="reflector" value="B">
    B
   </option>
   <option className="reflector" value="C">
    C
   </option>
  </select>

  <select className="rot" onChange={this.handleRotorLeft}>
   <option className="rotor" value="I">
    I
   </option>
   <option className="rotor" value="II">
    II
   </option>
   <option className="rotor" value="III">
    III
   </option>
  </select>

  <select className="rot" onChange={this.handleRotorMiddle}>
   <option className="rotor" value="I">
    I
   </option>
   <option className="rotor" value="II">
    II
   </option>
   <option className="rotor" value="III">
    III
   </option>
  </select>

  <select className="rot" onChange={this.handleRotorRight}>
   <option className="rotor" value="I">
    I
   </option>
   <option className="rotor" value="II">
    II
   </option>
   <option className="rotor" value="III">
    III
   </option>
  </select>
```

```
<div className="field">
  <input
    type="text"
    value={this.state.value}
    onChange={this.handleChange}
  />
</div>
<input name="" type="submit" value="Надіслати" />

<div className="plugboard">
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_1}
    onChange={this.handleChangePlug1}
    maxLength="2"
  />
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_2}
    onChange={this.handleChangePlug2}
    maxLength="2"
  />
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_3}
    onChange={this.handleChangePlug3}
    maxLength="2"
  />
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_4}
    onChange={this.handleChangePlug4}
    maxLength="2"
  />
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_5}
    onChange={this.handleChangePlug5}
    maxLength="2"
  />
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_6}
    onChange={this.handleChangePlug6}
    maxLength="2"
  />
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_7}
    onChange={this.handleChangePlug7}
    maxLength="2"
  />
  <input
    className="opt"
    type="text"
    value={this.state.plugboard_8}
```

```
            onChange={this.handleChangePlug8}
            maxLength="2"
          />
          <input
            className="opt"
            type="text"
            value={this.state.plugboard_9}
            onChange={this.handleChangePlug9}
            maxLength="2"
          />
          <input
            className="opt"
            type="text"
            value={this.state.plugboard_10}
            onChange={this.handleChangePlug10}
            maxLength="2"
          />
          <input
            className="opt"
            type="text"
            value={this.state.plugboard_11}
            onChange={this.handleChangePlug11}
            maxLength="2"
          />
          <input
            className="opt"
            type="text"
            value={this.state.plugboard_12}
            onChange={this.handleChangePlug12}
            maxLength="2"
          />
          <input
            className="opt"
            type="text"
            value={this.state.plugboard_13}
            onChange={this.handleChangePlug13}
            maxLength="2"
          />
        </div>
      </form>
    );
  }
}
```

Index.js

```
var express = require('express');
var router = express.Router();
const algo = require('./algorithm');

router
  .get('/', function (req, res, next) {
    if (!req.body) return res.sendStatus(400);
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
  })
  .post('/', function (req, res, next) {
    if (!req.body) return res.sendStatus(400);
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
    let plug = [
      req.body.plugboard_1,
      req.body.plugboard_2,
```

```
      req.body.plugboard_3,
      req.body.plugboard_4,
      req.body.plugboard_5,
      req.body.plugboard_6,
      req.body.plugboard_7,
      req.body.plugboard_8,
      req.body.plugboard_9,
      req.body.plugboard_10,
      req.body.plugboard_11,
      req.body.plugboard_12,
      req.body.plugboard_13,
    ];
    res.json(
      algo.f(
        req.body.value,
        req.body.stepLR,
        req.body.stepMR,
        req.body.stepRR,
        plug
      )
    );
  });

module.exports = router;
```

algorithm.js

```
const {o} = require('./data');

exports.f = function algorithm(
  t,
  rotorStepLeft,
  rotorStepMiddle,
  rotorStepRight,
  arr
) {
  let cipherText = '',
    rL = rotorStepLeft > 1 ? rotorStepLeft : 1,
    rM = rotorStepMiddle > 1 ? rotorStepMiddle : 1,
    rR = rotorStepRight > 1 ? rotorStepRight - t.length : 1,
    r,
    r1,
    r2,
    r3,
    r4,
    r5,
    r6,
    res,
    fin = '',
    pr,
    result;

  for (let i = 0; i < t.length; i++) {
    t[i] = plugSwap(t[i], arr);
    //ring 1 step 1
    r1 = (rR + indexN(t[i])) % 26;
    pr = o.rotorRight[r1];
    rR < 26 ? rR++ : (rM++, (rR = 0));
    rM < 26 ? rM++ : (rL++, (rM = 0));
    //console.log(rR, rM);

    //ring 2 step 2
    r2 = indexR(pr) + rM - rR;
```

```
    r2 = check(r2);
    pr = o.rotorMiddle[r2];
    //console.log(pr);
    // console.log(r2);

    // ring 3 step 3
    r3 = indexL(pr) + (rL - rM);
    r3 = check(r3);
    pr = o.rotorLeft[r3];
    // console.log(pr);

    // reflector step 4
    r = indexReflector(pr) - rL;
    r = check(r);
    pr = o.reflector[r];
    // console.log(pr);

    // ring 3 step 5
    r4 = (indexReflector(pr) + rL) % 26;
    pr = o.reflector[r4];
    // console.log(pr);

    // ring 2 step 6
    r5 = indexN(pr) - (rL - rM);
    r5 = check(r5);
    pr = o.normal[r5];

    // ring 1 step 7
    r6 = indexN(pr) - (rM - rR);
    r6 = check(r6);
    pr = o.normal[r6];

    // result step 8
    res = indexN(pr) - rR;
    res = check(res);
    pr = o.normal[res];

    pr = plugSwap(pr, arr);

    fin += pr;
  }
  result = fin;
  fin = '';
  rL = 1;
  rM = 1;
  rR = 1;
  return result;
};

function step1(text) {}

function check(text) {
  if (text > 0) return (text %= 26);
  return 26 + text;
}

function indexer(text) {
  return o.normal[text];
}

function indexN(text) {
  return o.normal.indexOf(text);
}
```

```
function indexL(text) {
  return o.rotorLeft.indexOf(text);
}

function indexM(text) {
  return o.rotorMiddle.indexOf(text);
}

function indexR(text) {
  return o.rotorRight.indexOf(text);
}

function indexReflector(text) {
  return o.reflector.indexOf(text);
}

function plugSwap(s, arr) {
  let check;
  let left = [];
  let right = [];
  let symb = s.toUpprrCase();
  for (let i = 0; i < arr.length; i++) {
    left = arr[i].split('')[0]. toUpperCase();
    right = arr[i].split('')[1]. toUpperCase();
  if (symb == left) {
      symb = right;
    } else if (symb == right) {
      symb = left;
    }
  }
  //  console.log(symb);
  return symb;
}
```