**ЛАБОРАТОРНА РОБОТА №1**

**НАЛАШТУВАННЯ СЕРЕДОВИЩА РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ ОС ANDROID**

**1.1 Мета роботи**

Налаштувати середовище розробки додатків для операційної системи Android. Ознайомитись з типовою структурою Android-проекту. Ознайомитись з системою автоматичного збирання проектів Gradle. Навчитись розробляти прості Android-додатки з використанням стандартної бібліотеки візуальних компонентів.

**1.2 Короткі теоретичні відомості**

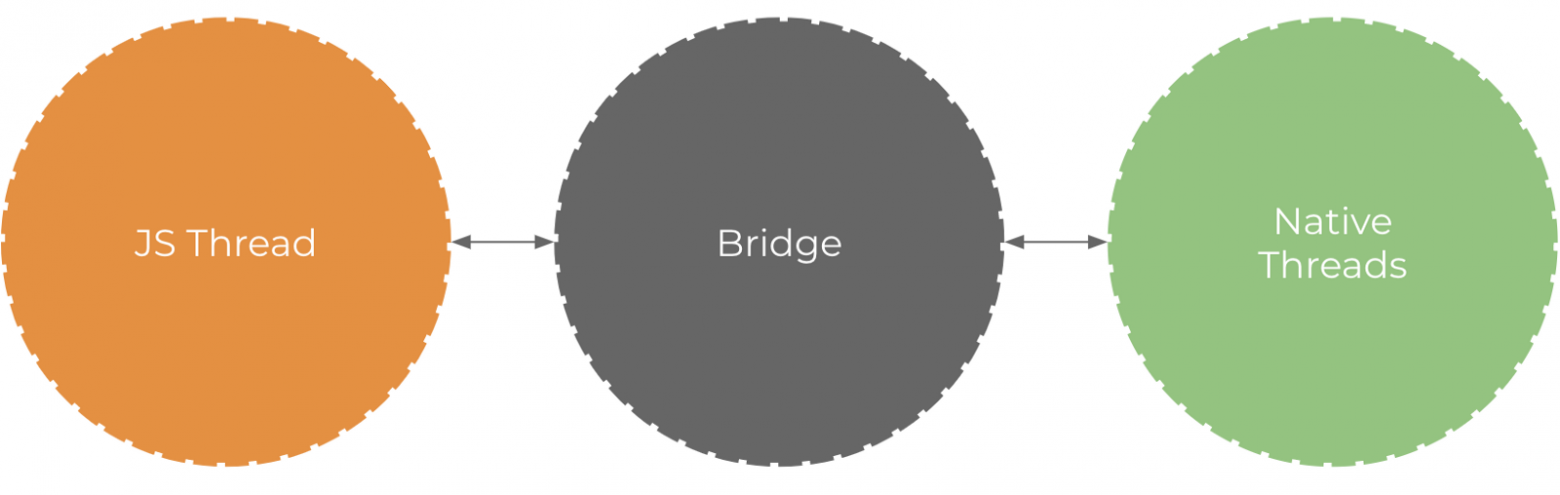
**React Native з Expo**

**React Native** – це кросплатформений фреймворк з відкритим вихідним кодом, написаний на JavaScript. React Native використовується для створення додатків, як для мобільних систем Android та iOS, так і для десктоп, серед яких Windows та macOS.

React Native був вперше випущений Facebook як проект з відкритим вихідним кодом у 2015 році. Усього за пару років він став одним із найкращих рішень, що використовуються для мобільної розробки. Розробка React Native використовується для підтримки деяких провідних мобільних програм у світі, включаючи Instagram, Facebook і Skype.

React Native написано з використанням суміші JavaScript та JXL, спеціального коду розмітки, схожого на XML. Платформа має можливість взаємодіяти як із областями – потоками на основі JavaScript, так і з існуючими потоками нативних програм. Як працює це спілкування? React Native використовує так званий міст. У той час як JavaScript і нативні потоки написані зовсім різними мовами, це функція мосту, яка робить можливим двонаправлений зв'язок.

Відмінна візуалізація концепції мосту:



Це означає, що - якщо ви вже маєте власний додаток для iOS або Android - ви все одно можете використовувати його компоненти або перейти до розробки на React Native.

Різниця між React Native та іншими кросплатформовими рішеннями для розробки полягає в тому, що React Native не відображає WebView у своєму коді. Він працює на реальних, нативних уявленнях та компонентах. Це одна з причин вражаючого успіху React Native.

**Expo** - це набір інструментів, за допомогою якого ви можете написати програму React Native за лічені хвилини. Він включає готові інструменти, такі як конфігурації Android Studio / XCode, управління сертифікатами в Apple & Google і push-сповіщення, і це лише деякі з них.

**1.3 Хід роботи**

Завдання для реалізації лабораторної роботи зображено в таблиці 1.1

Таблиця 1.1 - Завдання для реалізації лабораторної роботи

|  |  |
| --- | --- |
| № | Опис завдання |
| 18 | Створити Android-додаток «Віднови слово». Додаток складається з екранів головного меню, налаштувань та екрану гри. - Головне меню містить інформацію про розробника, кнопки «Налаштування», «Розпочати гру». - Екран налаштувань дозволяє задати мінімальний (4,5,6) та максимальний (7,8,9) розмір слів у символах та час зворотного відліку (1 хв, 2 хв, 5 хв). - Екран гри показує час, який залишивсь до закінчення гри, кількість вгаданих слів. Гравцеві показується слово з перемішаними випадковим чином буквами. Гравець може натискати на букви, складаючи з них слово. Наприклад, програма показує букви «ідроа». Натискаючи послідовно на букви «р», «а», «д», «і», «о» гравець складає слово «радіо»  У випадку помилки, гравець може вернути букву назад вгору, натиснувши на неї. Слово заповнюється послідовно, зліва направо. Якщо гравець вгадав слово, програма показує Toast-повідомлення «Вгадано» та генерує наступне слово. Гра закінчується, коли час вичерпано. Гравцеві 116 показується діалог з результатами: кількість вгаданих слів. Навігація має бути побудована на базі фрагментів (Fragments). |

Для створення екранів гри було використано бібліотеки:

react-navigation/core, react-navigation/native, react-navigation/stack.

Функціонал навагації між екранами було описано в файлі navigation.js, код файлу зображено в лістингу 1.1  
Лістинг 1.1

import 'react-native-gesture-handler'

import React from 'react'

import Main from './Main'

import Game from './Game'

import Settings from './Settings'

import { createStackNavigator } from '@react-navigation/stack'

import { NavigationContainer } from '@react-navigation/native'

const Stack = createStackNavigator();

export default function Navigate() {

    return <NavigationContainer>

        <Stack.Navigator>

            <Stack.Screen

                name="Main"

                component={Main}

                options={{ headerShown: false }}

            />

            <Stack.Screen

                name="Game"

                component={Game}

                options={{ headerShown: false }}

            />

            <Stack.Screen

                name="Settings"

                component={Settings}

                options={{ headerShown: false }}

            />

        </Stack.Navigator>

    </NavigationContainer>;

}

Перший екран – «Main» він містить інформацію про розробника, логотип гри, кнопки «Грати» та «Налаштування».

Вихідний код екрану зображено в лістингу 1.2

Лістинг 1.2

import { Text, View, StatusBar, SafeAreaView, TouchableOpacity, Image } from 'react-native'

import { useNavigation } from '@react-navigation/core'

import { styles } from '../styles/main'

export default function Main() {

    const navigation = useNavigation()

    return (

        <SafeAreaView style={styles.container}>

            <View style={styles.container2}>

                <Image source={require("../assets/icon.png")} style={{ height: 150, width: 150 }} />

                <View style={styles.box}>

                    <TouchableOpacity style={styles.textbutton}

                        onPress={() => navigation.replace('Game')}>

                        <Text style={styles.textmain}>

                            Грати

                        </Text>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.textbutton}

                        onPress={() => navigation.replace('Settings')}>

                        <Text style={styles.textmain}>

                            Налаштування

                        </Text>

                    </TouchableOpacity>

                </View>

            </View>

            <Text style={styles.developer}>

                Розробник Оленченко Ілля (2022)

            </Text>

            <StatusBar

                animated={true}

                barStyle={'light-content'}

                backgroundColor="#17153C"

            />

        </SafeAreaView>

    );

}

Другий екран гри – «Game» показує час, який залишивсь до закінчення гри, кількість вгаданих слів. Гравцеві показується слово з перемішаними випадковим чином буквами. Гравець може натискати на букви, складаючи з них слово. У випадку помилки, гравець може вернути букву назад вгору, натиснувши на неї. Слово заповнюється послідовно, зліва направо. Якщо гравець вгадав слово, програма генерує наступне слово. Вихідний код зображено в лістингу 1.3.

Лістинг 1.3

import React, { useState, useEffect } from 'react';

import { Text, View, StatusBar, SafeAreaView, TouchableOpacity, Modal, FlatList, Pressable } from 'react-native'

import AsyncStorage from '@react-native-async-storage/async-storage'

import { AntDesign } from '@expo/vector-icons';

import { useNavigation } from '@react-navigation/core'

import { styles } from '../styles/game'

let number = 0

let counter = 0

const WORDS0 = [

    {

        id: 0,

        word: 'радіо',

        leng: 5

    },

    {

        id: 2,

        word: 'слово',

        leng: 5

    },

    {

        id: 3,

        word: 'буда',

        leng: 4

    },

    {

        id: 4,

        word: 'аміак',

        leng: 5

    },

    {

        id: 5,

        word: 'білка',

        leng: 5

    },

    {

        id: 6,

        word: 'горщик',

        leng: 6

    },

    {

        id: 7,

        word: 'десерт',

        leng: 6

    },

    {

        id: 8,

        word: 'князь',

        leng: 5

    },

    {

        id: 9,

        word: 'гієна',

        leng: 5

    },

    {

        id: 10,

        word: 'серп',

        leng: 4

    }

]

const WORDS1 = [

    {

        id: 1,

        word: 'лиходійка',

        leng: 9

    },

    {

        id: 2,

        word: 'струмок',

        leng: 7

    },

    {

        id: 3,

        word: 'балаган',

        leng: 7

    },

    {

        id: 4,

        word: 'вермішель',

        leng: 9

    },

    {

        id: 5,

        word: 'сорочка',

        leng: 7

    },

    {

        id: 6,

        word: 'альпініст',

        leng: 9

    },

    {

        id: 7,

        word: 'андероль',

        leng: 8

    },

    {

        id: 8,

        word: 'лавочник',

        leng: 8

    },

    {

        id: 9,

        word: 'портфель',

        leng: 8

    },

    {

        id: 10,

        word: 'агроном',

        leng: 7

    }

]

export default function Game() {

    const navigation = useNavigation()

    const [modalVisible, setModalVisible] = useState(false);

    const [level, setLevel] = useState('');

    const [time, setTime] = useState('');

    const [counterTextMM, setcounterTextMM] = useState('00');

    const [counterTextSS, setcounterTextSS] = useState('00');

    const [words, setWords] = useState([]);

    const [randomwords, setRandomWords] = useState([]);

    const [buttonstart, setButtonStart] = useState('flex');

    const [safeareaview, setSafeAreaView] = useState('flex');

    const load = async () => {

        try {

            let level = await AsyncStorage.getItem('key\_level');

            let time = await AsyncStorage.getItem('key\_time');

            if (level !== null && time !== null) {

                setLevel(parseInt(level, 10));

                setTime(parseInt(time, 10));

                if (time == 120) {

                    setcounterTextMM('02')

                    setcounterTextSS('00')

                }

                else if (time == 60) {

                    setcounterTextMM('01')

                    setcounterTextSS('00')

                }

            }

            else {

                setLevel(0);

                setTime(120);

                setcounterTextMM('02')

                setcounterTextSS('00')

            }

            console.log(level, time)

        } catch (err) {

            alert(err);

        }

    }

    function shuffle(array) {

        let currentIndex = array.length, randomIndex;

        while (currentIndex != 0) {

            randomIndex = Math.floor(Math.random() \* currentIndex);

            currentIndex--;

            [array[currentIndex], array[randomIndex]] = [

                array[randomIndex], array[currentIndex]];

        }

        return array;

    }

    function getRandWords() {

        if (level === 0) {

            const randArr = WORDS0[number].word.split('')

            shuffle(randArr)

            return randArr

        }

        if (level === 1) {

            const randArr = WORDS1[number].word.split('')

            shuffle(randArr)

            return randArr

        }

    }

    function checking(words) {

        let chek = 0

        let i = 0

        if (level === 0) {

            while (WORDS0[number].leng > i) {

                if (words[i] === WORDS0[number].word[i]) {

                    chek += 1

                }

                i++

            }

            if (chek === WORDS0[number].leng) {

                number++

                if (number === 10) {

                    counter = 0

                }

                else

                    putNullWords()

            }

        }

        if (level === 1) {

            while (WORDS1[number].leng > i) {

                if (words[i] === WORDS1[number].word[i]) {

                    chek += 1

                }

                i++

            }

            if (chek === WORDS1[number].leng) {

                number++

                if (number === 10) {

                    counter = 0

                }

                else

                    putNullWords()

            }

        }

    }

    function putNullWords() {

        setButtonStart('none')

        //shuffle(WORDS0)

        //shuffle(WORDS1)

        const confusedArr = getRandWords()

        setRandomWords(confusedArr)

        words.length = 0

        var i = 0

        if (level === 0) {

            while (WORDS0[number].leng > i) {

                words[i] = '-'

                i++

            }

        }

        if (level === 1) {

            while (WORDS1[number].leng > i) {

                words[i] = '-'

                i++

            }

        }

        setWords([...words])

        return 0

    }

    const EditElementFromRandomwords = (index, item) => {

        if (item !== '-') {

            let i = 0

            if (level === 0) {

                while (WORDS0[number].leng > i) {

                    if (words[i] === '-') {

                        words[i] = item

                        randomwords[index] = '-'

                        setRandomWords([...randomwords])

                        setWords([...words])

                        break

                    }

                    i++

                }

                checking(words)

            }

            if (level === 1) {

                while (WORDS1[number].leng > i) {

                    if (words[i] === '-') {

                        words[i] = item

                        randomwords[index] = '-'

                        setRandomWords([...randomwords])

                        setWords([...words])

                        break

                    }

                    i++

                }

                checking(words)

            }

        }

    }

    const DeleteElementFromWords = (index, item) => {

        if (item !== '-') {

            let i = 0

            if (level === 0) {

                while (WORDS0[number].leng > i) {

                    if (randomwords[i] === '-') {

                        words[index] = '-'

                        randomwords[i] = item

                        setWords([...words])

                        setRandomWords([...randomwords])

                        break

                    }

                    i++

                }

            }

            if (level === 1) {

                while (WORDS1[number].leng > i) {

                    if (randomwords[i] === '-') {

                        words[index] = '-'

                        randomwords[i] = item

                        setWords([...words])

                        setRandomWords([...randomwords])

                        break

                    }

                    i++

                }

            }

        }

    }

    function timer() {

        counter = time

        const intervalId = setInterval(async () => {

            if (level === 0) {

                counter -= 1

                console.log(counter)

                if (counter <= 0) {

                    clearInterval(intervalId)

                    setModalVisible(true)

                    setcounterTextMM('00')

                    setcounterTextSS('00')

                }

                if (counter >= 60) {

                    setcounterTextMM('01')

                    if ((counter - 60) < 10)

                        setcounterTextSS('0' + (counter - 60))

                    else

                        setcounterTextSS(counter - 60)

                }

                else if (counter <= 60) {

                    setcounterTextMM('00')

                    if (counter < 10)

                        setcounterTextSS('0' + counter)

                    else

                        setcounterTextSS(counter)

                }

            }

            if (level === 1) {

                counter -= 1

                console.log(counter)

                if (counter <= 0) {

                    clearInterval(intervalId)

                    setModalVisible(true)

                    setcounterTextMM('00')

                    setcounterTextSS('00')

                }

                if (counter < 60) {

                    setcounterTextMM('00')

                    if (counter < 10)

                        setcounterTextSS('0' + counter)

                    else

                        setcounterTextSS(counter)

                }

            }

        }, 1000)

    }

    useEffect(() => {

        load();

    }, []);

    safeareaview

    return (

        <SafeAreaView style={styles.container}>

            <View style={styles.container1}>

                <View style={styles.backwordnumber}>

                    <Text style={{ fontFamily: 'robot-regular' }}>Вгадано слів: {number} з 10</Text>

                </View>

                <View>

                    <Text style={{ fontFamily: 'robot-regular' }}>{counterTextMM} : {counterTextSS}</Text>

                </View>

            </View>

            <View style={styles.container2}>

                <View style={{ display: safeareaview }}>

                    <TouchableOpacity style={[{ display: buttonstart }, styles.startButton]}

                        onPress={() => {

                            timer()

                            putNullWords()

                        }}>

                        <Text style={styles.backTextstart}>Старт</Text>

                    </TouchableOpacity>

                    <View style={styles.box1}>

                        <FlatList horizontal={true} data={randomwords} renderItem={({ index, item }) => (

                            <TouchableOpacity key={index} style={styles.button1}

                                onPress={() => EditElementFromRandomwords(index, item)}>

                                <Text style={styles.word1}>{item} </Text>

                            </TouchableOpacity>

                        )} />

                    </View>

                    <View style={styles.box1}>

                        <FlatList horizontal={true} data={words} renderItem={({ index, item }) => (

                            <Pressable key={index} style={styles.button1}

                                onPress={() => DeleteElementFromWords(index, item)}>

                                <Text style={styles.word1}>{item} </Text>

                            </Pressable>

                        )} />

                    </View>

                </View>

            </View>

            <View style={styles.backView}>

                <TouchableOpacity style={styles.backButton}

                    onPress={() => {

                        setModalVisible(true),

                            counter = 0

                    }}>

                    <AntDesign name="leftsquare" size={45} color="#4ebfa7" />

                </TouchableOpacity>

            </View>

            <Modal

                animationType="slide"

                transparent={true}

                visible={modalVisible}

                onRequestClose={() => {

                    setModalVisible(!modalVisible);

                }}

            >

                <View style={styles.centeredView}>

                    <View style={styles.modalView}>

                        <Text style={styles.modalText}>Кількість вгаданих слів: {number}</Text>

                        <TouchableOpacity

                            style={styles.buttonClose}

                            onPress={() => {

                                setModalVisible(!modalVisible),

                                    number = 0,

                                    navigation.replace('Main')

                            }}>

                            <Text style={styles.textStyle}>В меню</Text>

                        </TouchableOpacity>

                    </View>

                </View>

            </Modal>

            <StatusBar

                animated={true}

                barStyle={'light-content'}

                backgroundColor="#17153C"

            />

        </SafeAreaView >

    );

}

Третій екрна – «Settings» це екран налаштувань де можливо вибрати складність гри, легку або важку. Якщо обрати легку складніть, кількість букв в слові не буде перевищувати 6 і часу буде доступно 2хв. Якщо обрати складну то букв в слові буде до 9 включно час буде доступно 1хв. . Вихідний код зображено в лістингу 1.4.

Лістинг 1.4

import { Text, View, StatusBar, SafeAreaView, TouchableOpacity } from 'react-native'

import AsyncStorage from '@react-native-async-storage/async-storage'

import { AntDesign } from '@expo/vector-icons';

import { styles } from '../styles/settings'

import { useNavigation } from '@react-navigation/core'

export default function Settings() {

    const navigation = useNavigation()

    const save = async (level, time) => {

        try {

            await AsyncStorage.setItem('key\_level', level);

            await AsyncStorage.setItem('key\_time', time);

            navigation.navigate('Main')

        } catch (err) {

            alert(err);

        }

    }

    return (

        <SafeAreaView style={styles.container}>

            <View style={styles.header}>

                <Text style={styles.wordSettings}>Налаштування</Text>

            </View>

            <View style={styles.backView}>

                <TouchableOpacity style={styles.backButton}

                    onPress={() => navigation.replace('Main')}>

                    <AntDesign name="leftsquare" size={45} color="#4ebfa7" />

                </TouchableOpacity>

            </View>

            <View style={styles.box2}>

                <TouchableOpacity style={styles.button} onPress={() => save('0', '120')}>

                    <Text style={styles.word}>Легко</Text>

                </TouchableOpacity>

                <TouchableOpacity style={styles.button} onPress={() => save('1', '60')}>

                    <Text style={styles.word}>Важко</Text>

                </TouchableOpacity>

            </View>

            <StatusBar

                animated={true}

                barStyle={'light-content'}

                backgroundColor="#17153C"

            />

        </SafeAreaView>

    );

}

На рисунку 1.1. зображено головне меню. Вибір складності зображено на рисунку 1.2. Екран з запущеною грою зображено на рисунку 1.3.

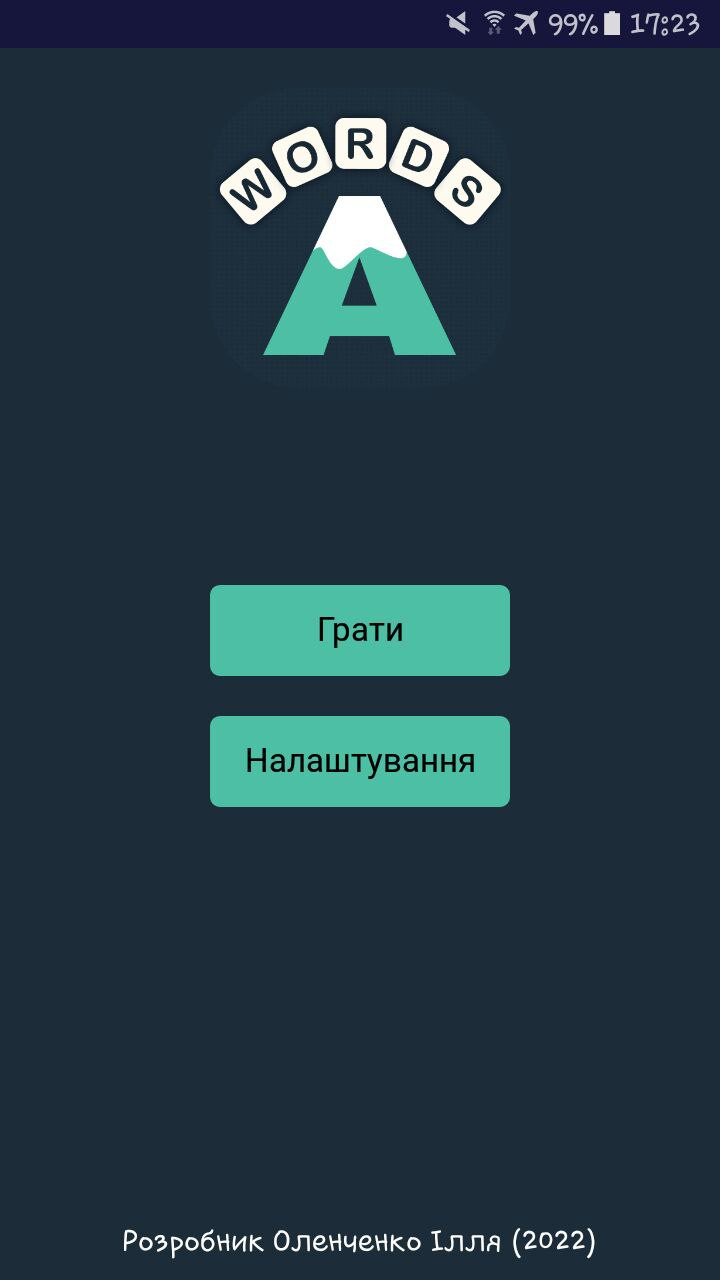


Рисунок 1.1 – Головне меню



Рисунок 1.2 – Екран налаштуваннь

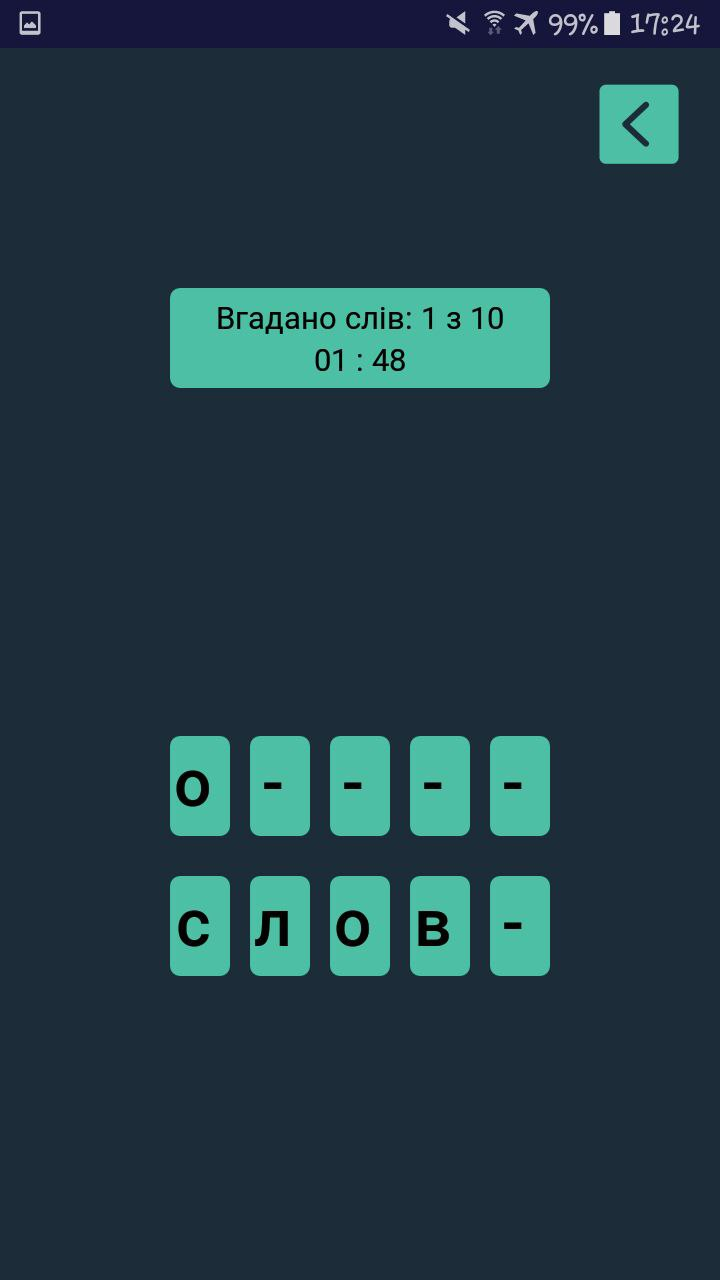


Рисунок 1.3 - Екран з запущеною грою

Висновок: на даній лабораторній роботі я створив мобільний застосунок Restore Word.