การพัฒนาเว็ปแอปพลิเคชั่น

เรื่อง การพัฒนาโมบายแอปพลิเคชันการจัดการเวลา Chronobreak

จัดทำโดย

นางสาว เอสเธอร์ เหล่าสันติพลวุฒิ 003

นาย ปฏิวัฒน์ กฤษฎิ์สุภารัตน์ 031

นาย อังคสิทธิ์ การุณชาติ 035

นาย ทรรศนัย สวยล้ำ 037

เสนอ

ผศ.ดร. เสถียร จันทร์ปลา

รหัสวิชา CSD3201

ภาคเรียนที่ 2

คณะวิทยาศาสตร์และเทคโนโลยี

มหาวิทยาลัยราชภัฏสวนสุนันทา 2567

บทนำ

ในยุคดิจิทัลปัจจุบัน สมาร์ทโฟนกลายเป็นอุปกรณ์ที่ขาดไม่ได้ในชีวิตประจำวันของผู้คนการจัดการเวลาอย่างมีประสิทธิภาพจึงเป็นสิ่งสำคัญ แอปพลิเคชันมือถือที่ช่วยในการจัดการเวลาได้รับความนิยมอย่างแพร่หลาย เนื่องจากสามารถช่วยให้ผู้ใช้วางแผนและติดตามกิจกรรมต่าง ๆ ได้อย่างสะดวกสบายแม้ว่าแอปพลิเคชันจัดการเวลาจะมีอยู่มากมาย แต่หลายแอปพลิเคชันยังขาดความยืดหยุ่นและความสามารถในการปรับแต่งให้ตรงกับความต้องการเฉพาะของผู้ใช้ นอกจากนี้ การซิงโครไนซ์ข้อมูลระหว่างอุปกรณ์หรือการสำรองข้อมูลยังเป็นปัญหาที่พบได้บ่อย ผู้ใช้คาดหวังว่าแอปพลิเคชันจะสามารถตอบสนองต่อความต้องการเฉพาะของตนได้ แต่ในความเป็นจริง แอปพลิเคชันที่มีอยู่ยังไม่สามารถตอบสนองได้อย่างเต็มที่

โครงงานนี้มีวัตถุประสงค์ในการพัฒนาแอปพลิเคชันมือถือที่ชื่อว่า "Chronobreak" โดยใช้ React Native ซึ่งเป็นเฟรมเวิร์กที่ช่วยในการพัฒนาแอปพลิเคชันข้ามแพลตฟอร์ม ทำให้สามารถใช้งานได้ทั้งบนระบบปฏิบัติการ iOS และ Android นอกจากนี้ ยังใช้ฐานข้อมูล MySQL สำหรับการจัดเก็บและจัดการข้อมูลของผู้ใช้อย่างมีประสิทธิภาพ การผสานรวมเทคโนโลยีเหล่านี้จะช่วยให้แอปพลิเคชันมีความยืดหยุ่น ปรับแต่งได้ตามความต้องการของผู้ใช้ และสามารถซิงโครไนซ์ข้อมูลระหว่างอุปกรณ์ได้อย่างราบรื่น

การพัฒนาแอปพลิเคชัน "Chronobreak" อ้างอิงจากทฤษฎีการพัฒนาซอฟต์แวร์แบบ Agile ซึ่งเน้นการพัฒนาเป็นขั้นตอน มีการทดสอบและปรับปรุงอย่างต่อเนื่อง เพื่อให้ได้ผลิตภัณฑ์ที่ตรงกับความต้องการของผู้ใช้มากที่สุด นอกจากนี้ ยังใช้แนวคิดของการออกแบบที่มุ่งเน้นผู้ใช้ (User-Centered Design) เพื่อให้แน่ใจว่าแอปพลิเคชันมีอินเทอร์เฟซที่ใช้งานง่าย และตอบสนองต่อประสบการณ์ของผู้ใช้อย่างแท้จริง

วัตถุประสงค์ของการวิจัยและความสำคัญของการวิจัย

วัตถุประสงค์หลักของโครงงานนี้คือการพัฒนาแอปพลิเคชันจัดการเวลาที่มีความยืดหยุ่น ปรับแต่งได้ตามความต้องการของผู้ใช้ และสามารถซิงโครไนซ์ข้อมูลระหว่างอุปกรณ์ได้อย่างมีประสิทธิภาพ การวิจัยนี้มีความสำคัญเนื่องจากจะช่วยเติมเต็มช่องว่างที่มีอยู่ในแอปพลิเคชันจัดการเวลาปัจจุบัน และมอบประสบการณ์การใช้งานที่ดียิ่งขึ้นให้กับผู้ใช้

อ้างอิง

- คุณากร กล้าอาษา และพัชระ กันทา. (2563). พัฒนาโมบายแอปพลิเคชันก้าวไกลทูเดย์. วารสารวิชาการ "การประยุกต์ใช้เทคโนโลยีสารสนเทศ", 1(1), 1-10 e-research.siam.edu

- บัญชา ปะสีละเตสัง. (2563). พัฒนา Application ด้วย React และ React Native. กรุงเทพฯ: ซีเอ็ดยูเคชั่น m.se-ed.com
- Hajiheydari, N., & Ashkani, M. (2018). Mobile application user behavior in the developing countries: a survey in Iran. Information Systems, 77, 22–33 ph02.tci-thaijo.org
- Nuanmeesri, S. (2019). Mobile application for the purpose of marketing, product distribution and location-based logistics for elderly farmers. Applied Computing and Informatics, 1–20 https://doi.org/10.1016/j.aci.2019.11.001. ph02.tci-thaijo.org

จุดประสงค์

- ช่วยให้ผู้ใช้งานสามารถบริหารและจัดการเวลาได้อย่างมีประสิทธิภาพ
- เพื่อพัฒนาโมบายแอปพลิเคชันการจัดการเวลา
- เพิ่มความสะดวกในการดูเวลาและเปรียบเทียบเขตเวลาโลก (Time Zone)
- ลดข้อผิดพลาดที่อาจเกิดจากการคำนวณเวลาข้ามเขต

การวิเคราะห์ความต้องการ (Requirement Analysis)

เป้าหมายของแอป

- ช่วยให้ผู้ใช้สามารถ จัดการเวลา ได้อย่างมีประสิทธิภาพ
- เพิ่ม ประสิทธิภาพในการจัดการเวลา ของผู้ใช้
- รองรับการทำงาน ข้ามแพลตฟอร์ม (iOS และ Android)
- อินเทอร์เฟซใช้งานง่ายตามหลัก User-Centered Design

ฟังก์ชันที่จำเป็นของแอปพลิเคชัน

การจัดการเวลาและกิจกรรม

- สร้าง แก้ไข และลบกิจกรรม
- ดูเวลาโลก
- แปลงเวลาของเขตเวลาโลก
- นับเวลาถอยหลัง
- กำหนดเวลาแจ้งเตือน

การเชื่อมต่อกับฐานข้อมูล

- บันทึกข้อมูลกิจกรรมและเวลาของผู้ใช้ลงใน MySQL

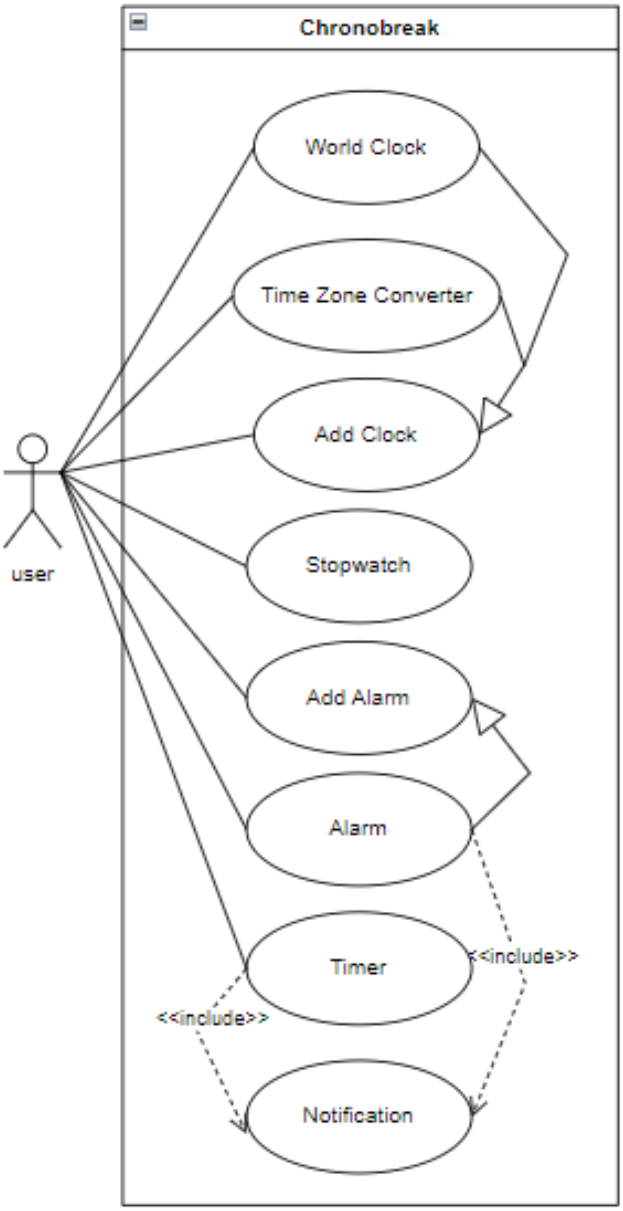
การแจ้งเตือนและเตือนความจำ

- ใช้ Push Notification เพื่อแจ้งเตือนกิจกรรม
- รองรับการตั้งค่าการแจ้งเตือนแบบปรับแต่งได้

ระบบวิเคราะห์พฤติกรรมการใช้เวลา

- สร้างรายงานการใช้เวลา
- แสดงข้อมูลสถิติการใช้เวลาในแต่ละหมวดหมู่

Use Case Diagram



คำอธิบายภาพรวมของระบบ

Frontend:

- พัฒนาโดยใช้ React Native รองรับทั้ง iOS และ Android
- ออกแบบ UI/UX ให้ใช้งานง่ายและตอบสนองได้ดี

Backend:

- ใช้ Node.js และ Express เป็นเซิร์ฟเวอร์ API
- จัดการการเชื่อมต่อกับฐานข้อมูล MySQL

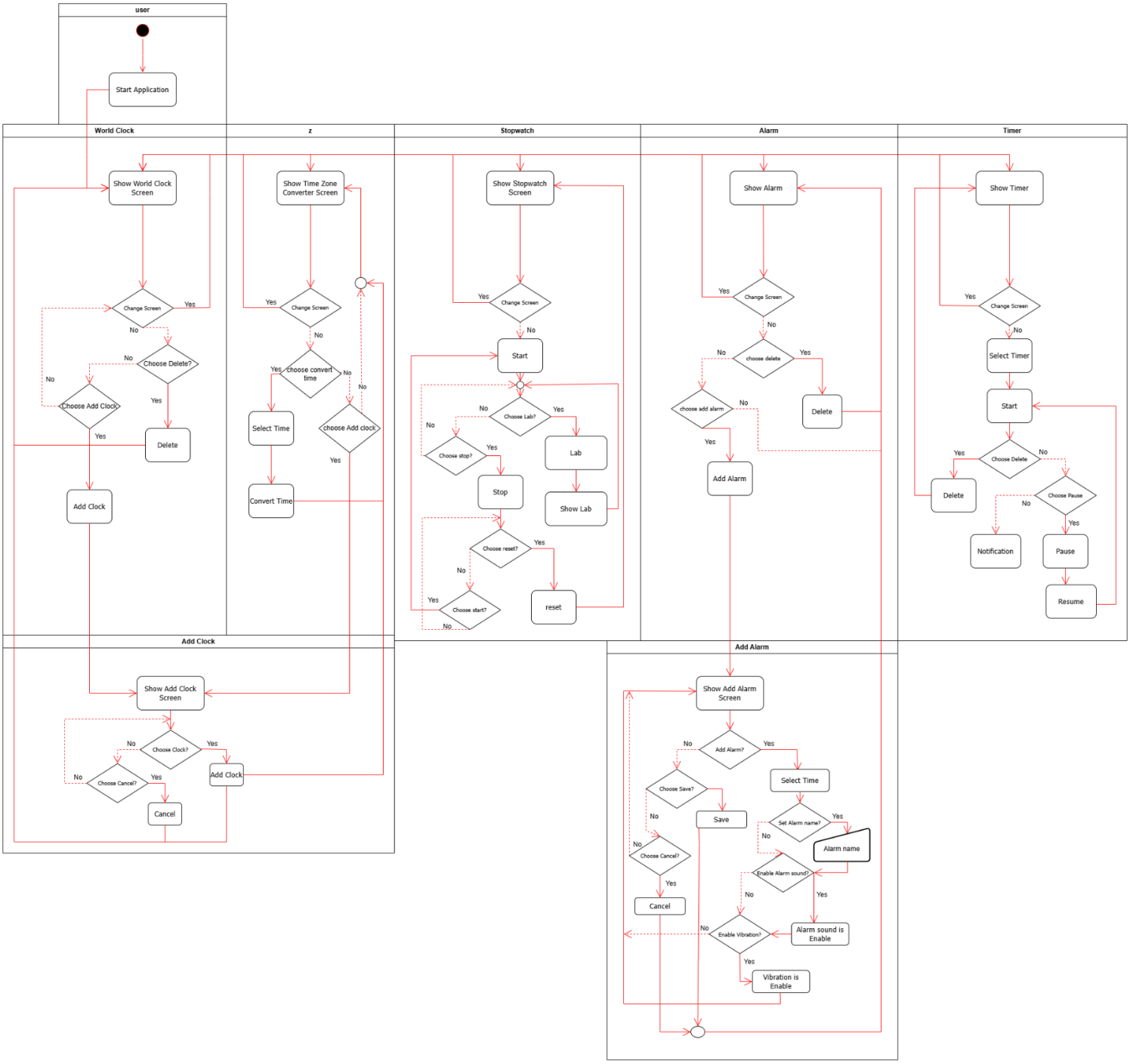
Database:

- ฐานข้อมูล MySQL ใช้จัดเก็บข้อมูลผู้ใช้และกิจกรรม
- มีโครงสร้างฐานข้อมูลที่รองรับการซิงโครไนซ์ข้อมูล

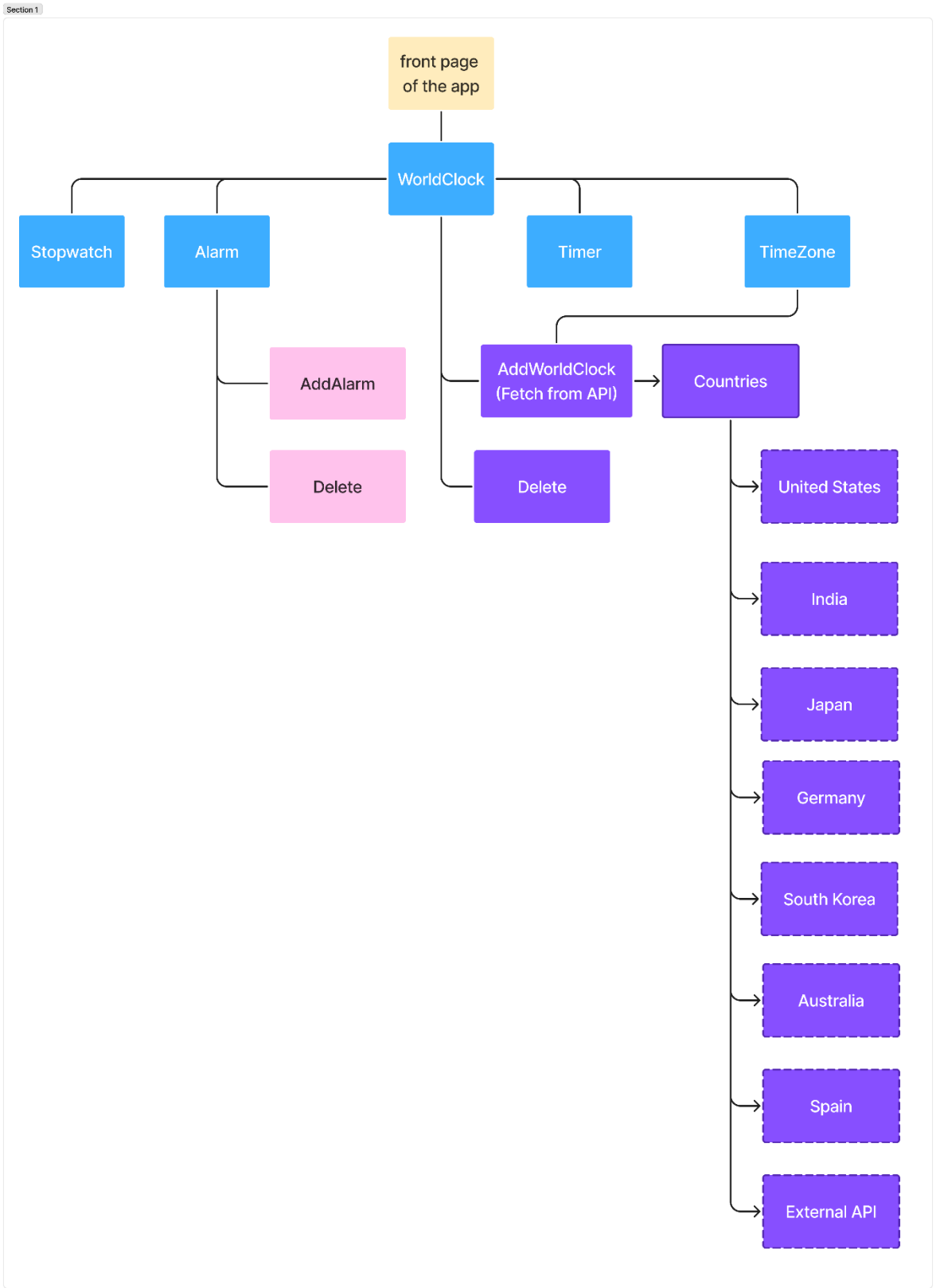
การทำงานโดยรวมของระบบ:

1. ผู้ใช้สามารถตั้งเวลานับถอยหลัง จับเวลา และตั้งนาฬิกาปลุกและการแจ้งเตือน
2. ผู้ใช้สามารถดูนาฬิกาโลก
3. ผู้ใช้สามารถเปลี่ยนเขตเวลา
4. ระบบแจ้งเตือนเตือนความจำกิจกรรมให้ผู้ใช้

รายละเอียดของแต่ละฟังก์ชัน

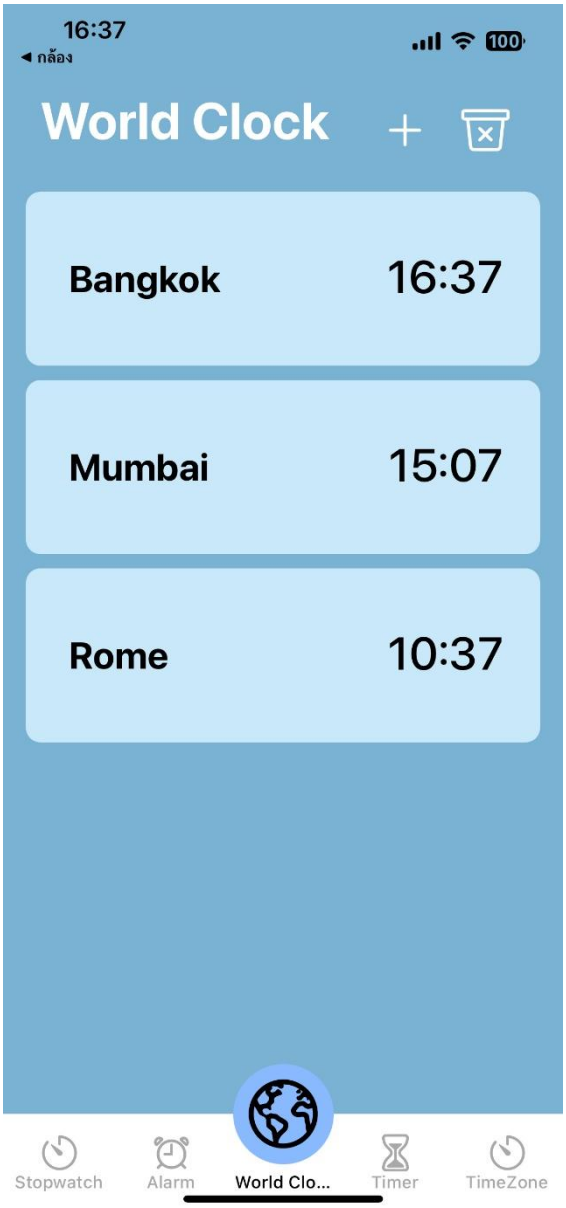


ออกแบบโครงสร้างระบบโดยรวม



ออกแบบส่วนติดต่อผู้ใช้ (UI/UX Design)

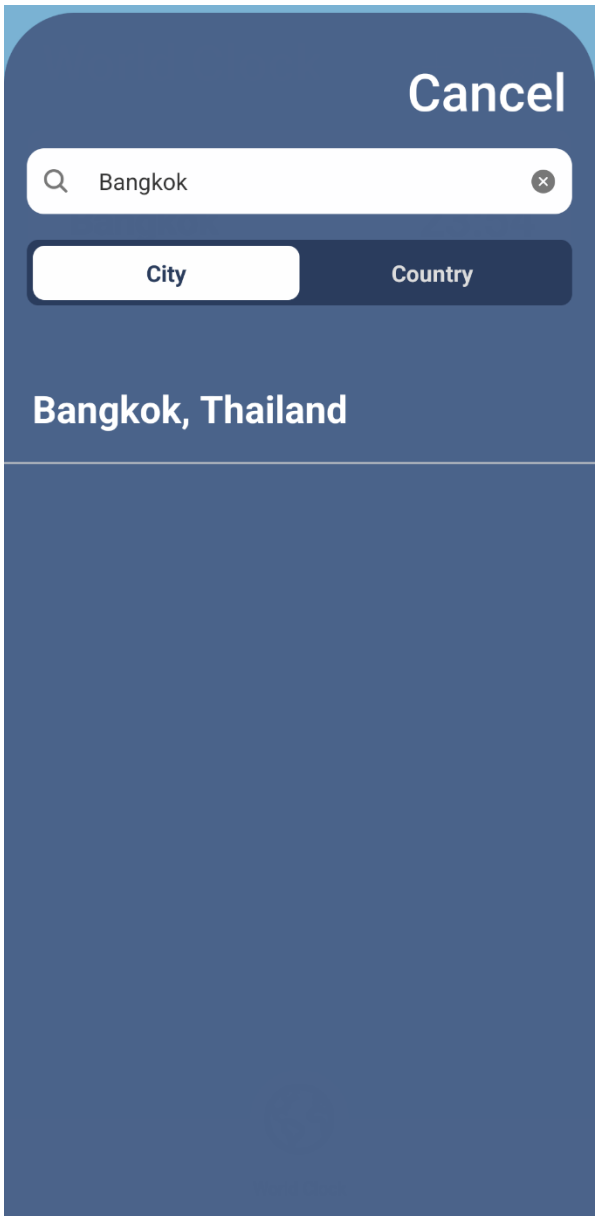
- Wireframe และ Mockup สำหรับหน้าต่าง ๆ
  - ➢ World Clock



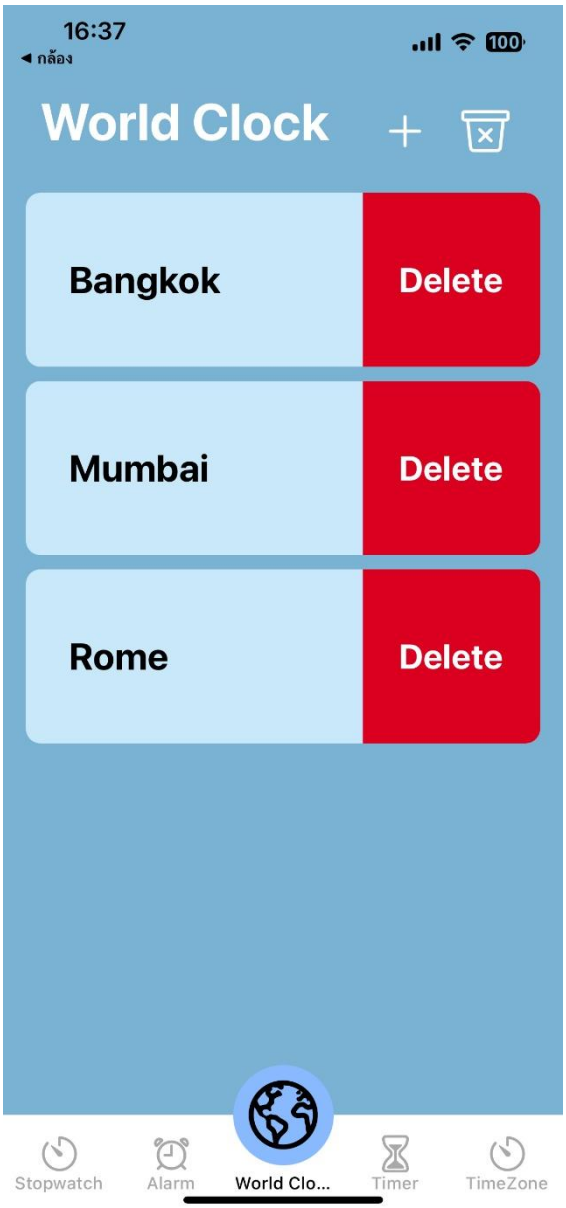
➢ Add Clock


Cancel
Search by city...
City
Country
Andorra la Vella, Andorra
Abu Dhabi, United Arab Emirates
Kabul, Afghanistan
Saint Johns, Antigua and Barbuda
The Valley, Anguilla
Tirana, Albania
Yerevan, Armenia

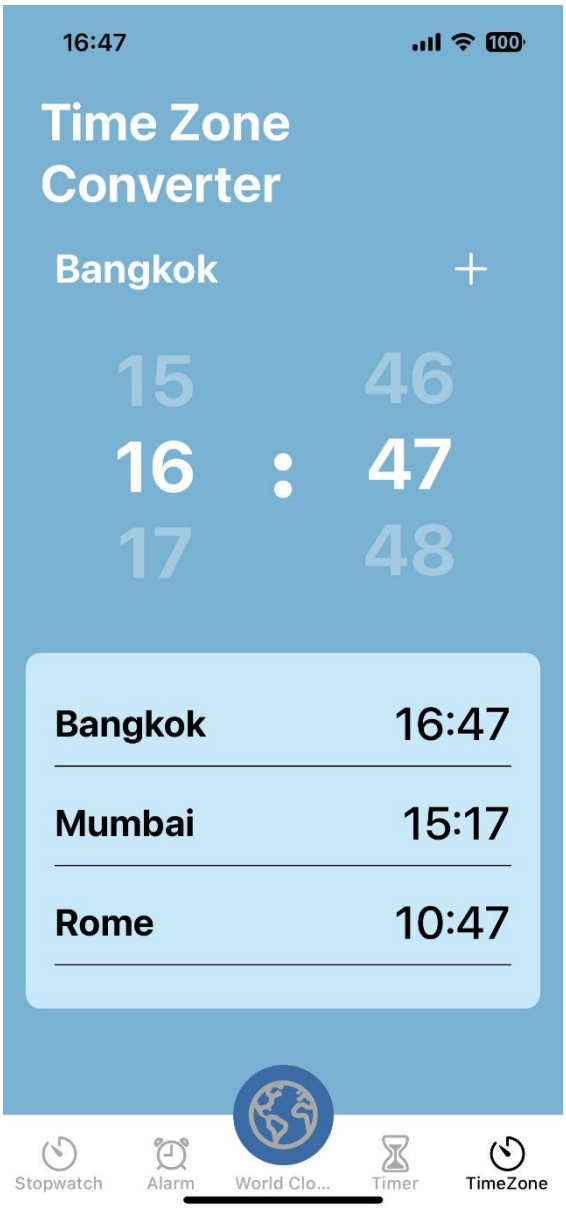
➢ Search Add Clock



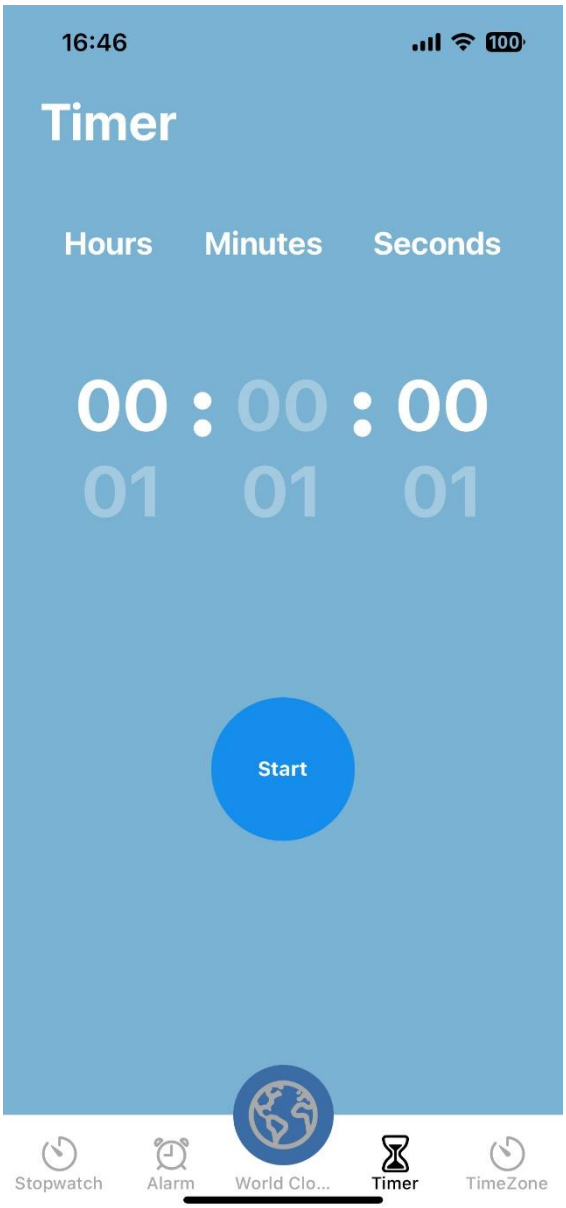
➢ Delete Clock



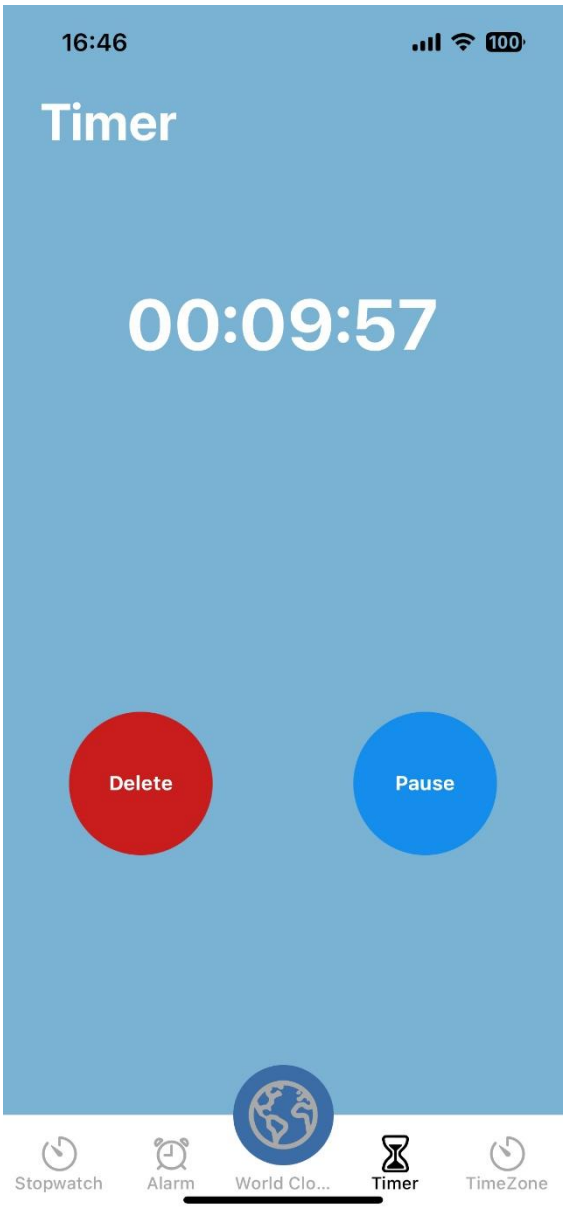
➢ Time Zone Converter



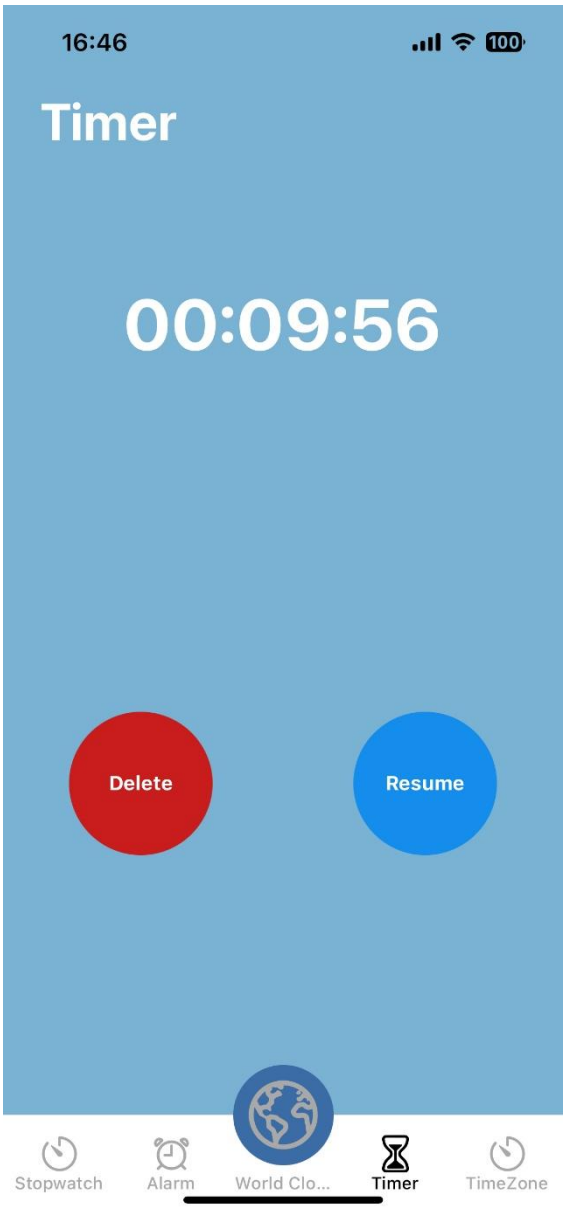
➢ Timer


16:46
100
Timer
Hours
Minutes
Seconds
00 : 00 : 00
01 01 01
Start
Stopwatch
Alarm
World Clo...
Timer
TimeZone

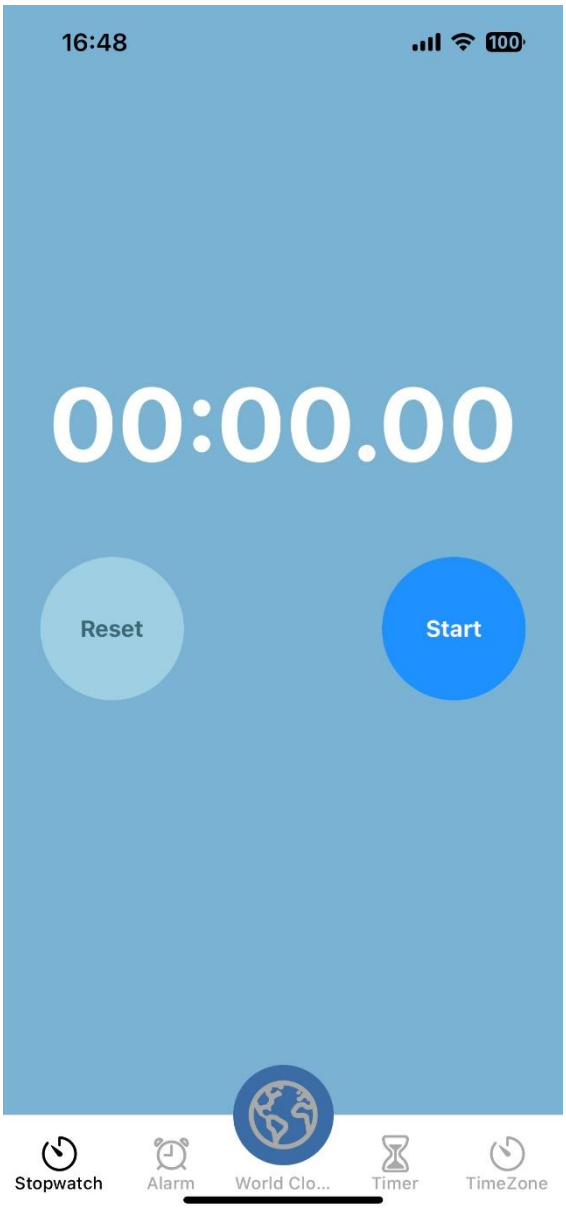
➢ Start Timer, Resume Timer



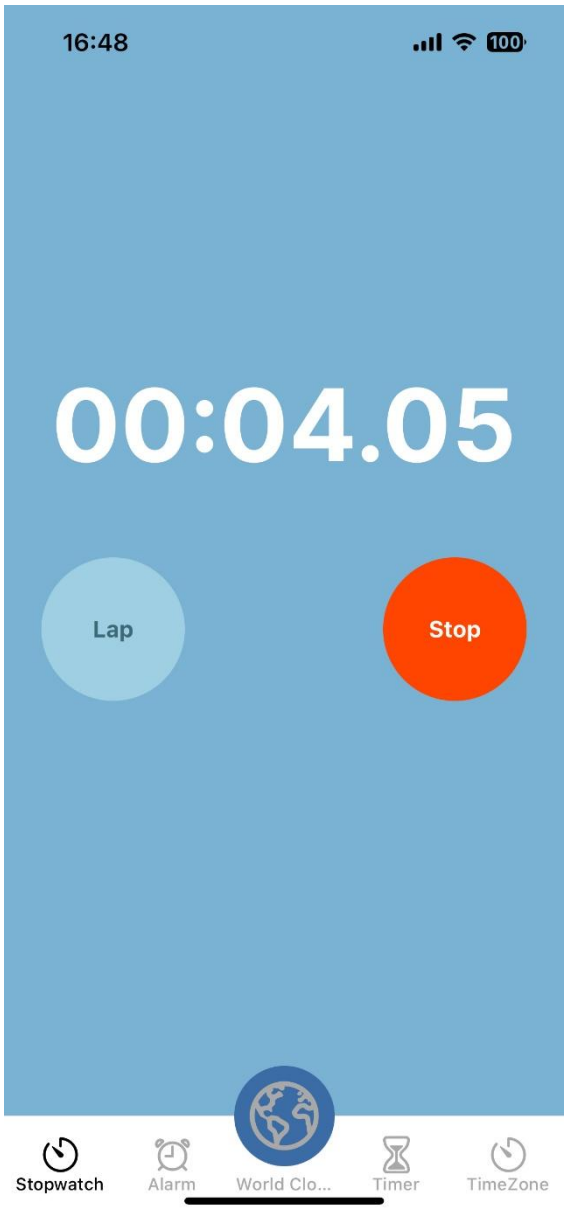
➢ Pause Timer, Delete Timer



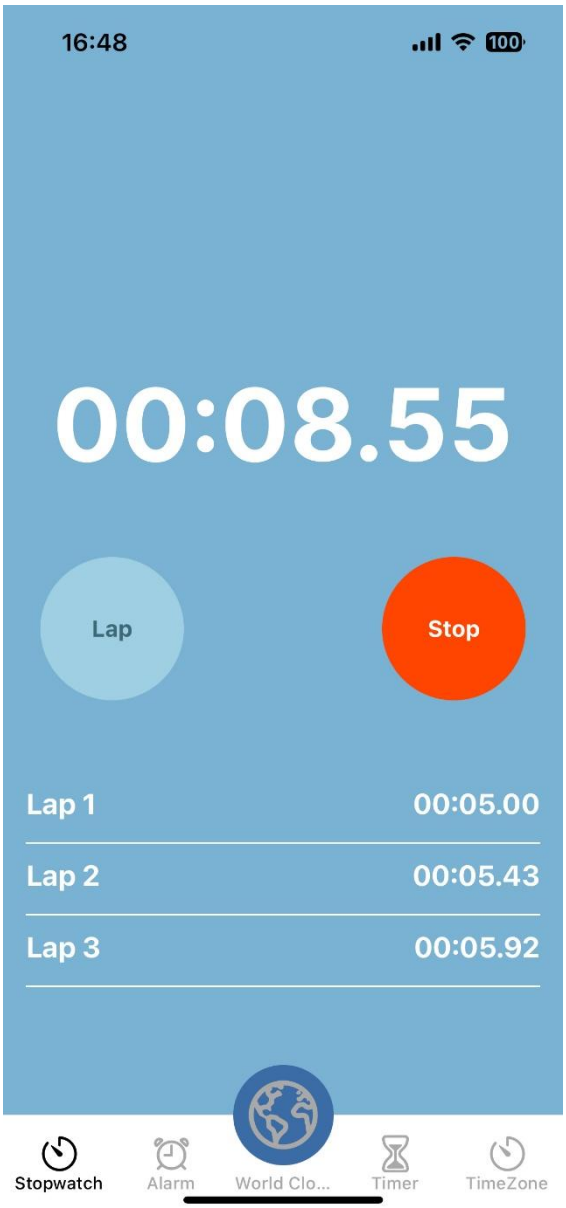
➢ Stopwatch, reset



➢ Start/Stop


16:48
100
00:04.05
Lap
Stop
Stopwatch
Alarm
World Clo...
Timer
TimeZone

➢ Record Lap

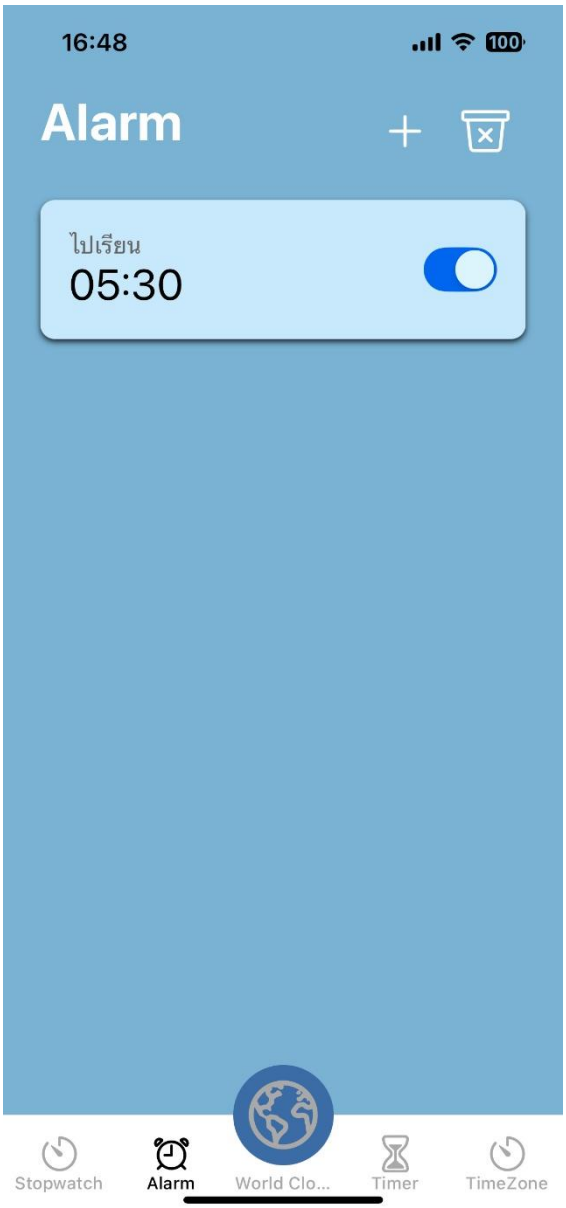

16:48
100
00:08.55
Lap
Stop
Lap 1
00:05.00
Lap 2
00:05.43
Lap 3
00:05.92
Stopwatch
Alarm
World Clo...
Timer
TimeZone

➢ Alarm

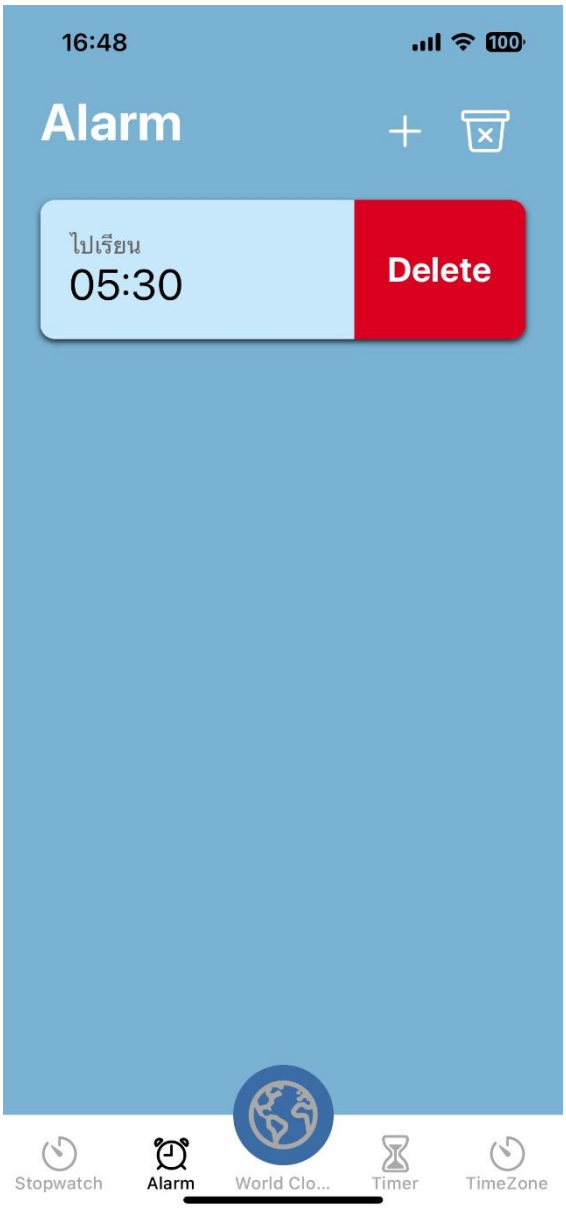
➢ Add Alarm


16:48
Cancel
Save
04
05
06
:
29
30
31
ไปเรียน
Alarm sound
Vibration

➢ Alarm Card


16:48
100
Alarm
ไปเรียน
05:30
Stopwatch
Alarm
World Clo...
Timer
TimeZone

➢ Delete Alarm



ออกแบบโครงสร้างฐานข้อมูล (Database Design)

ER Diagram:

Database Tables:

- **cities**

id (PK)	integer
city	string
country	string
code	string

Example:

- **cities**

id	city	country	code
1	Bangkok	Thailand	TH
2	Tokyo	Japan	JP
3	London	United Kingdom	GB

การพัฒนาโปรแกรม

ไฟล์	การทำงาน
App.js	หน้าแรก
TimeUtils.js	ไฟล์จัดการเวลา
AddAlarm.js	หน้าจอตั้งนาฬิกาปลุก
AddClock.js	หน้าจอเพิ่มเวลาของเมืองที่จะดู
Alarm.js	หน้าจอนาฬิกาปลุก
Stopwatch.js	หน้าจอจับเวลา
Timer.js	หน้าจอตั้งเวลานับถอยหลัง
WorldClock.js	หน้าจอดูเวลาโลก
TimeZoneConverter.js	หน้าจอแปลงเวลา
AlarmNavigator.js	stack navigator ของ Alarm เชื่อมต่อกับ AddAlarm
MainNavigator.js	tab navigator เป็น navigator หลัก
TimeZoneNavigator.js	stack navigator ของ TimeZoneConverter เชื่อมต่อกับ AddClock
WorldClockNavigator.js	stack navigator ของ WorldClock เชื่อต่อกับ AddClock
AlarmCard.js	component card ใช้ใน Alarm
ClockCard.js	component card ใช้ใน WorldClock
SelectTimer.js	เลือกเวลาของ Timer
TimeSelector.js	เลือกเวลาของ TimeZoneConverter และ Alarm
Color.js	ค่าคงที่ของสี

App.js

```
import { StatusBar } from "expo-status-bar";
import { NavigationContainer } from "@react-navigation/native";
import MainNavigator from "./navigator/MainNavigator";
// import { View } from 'react-native';

// import styles from './styles/styles';

export default function App() {
  return (
    <NavigationContainer>
      <MainNavigator />
      <StatusBar style="auto" />
    </NavigationContainer>
  );
}

```

TimeUtils.js

```
import moment from "moment-timezone";
import * as ct from "countries-and-timezones";

/**
 * Gets the current time for a given country's timezone.
 * @param {string} countryCode - The ISO country code.
 * @returns {string | null} The formatted time or null if the country is not found.
 */
export function getTimeByCountry(countryCode) {
  const country = ct.getCountry(countryCode);
  if (!country || !country.timezones.length) return null;

  const timezone = country.timezones[0];
  return moment.tz(timezone).format("HH:mm");
}

/**
 * Gets the current time for a given city's timezone.
 * @param {string} city - The city name.
 * @param {string} countryCode - The ISO country code.
 * @param {Date} [time] - The specific time to convert.
 * @returns {string | null} The formatted time or null if the city is not found.
 */
export function getTimeByCity(city, countryCode, time) {
  const country = ct.getCountry(countryCode);
  if (!country || !country.timezones.length) return null;

  const timezone = country.timezones[0];
  const momentTime = time ? moment(time) : moment();
  return momentTime.tz(timezone).format("HH:mm");
}

/**
 * Gets the current time for all countries' timezones.
 * @returns {Record<string, string | null>} An object with country codes as keys and formatted
times as values.
 */
export function getAllCountriesTime() {
  const countries = ct.getAllCountries();
  const countryTimes = {};

  for (const countryCode in countries) {
    countryTimes[countryCode] = getTimeByCountry(countryCode);
  }

  return countryTimes;
}

/**
 * Gets the current time for all popular cities' timezones.
```

```
 * @param {Array<{ city: string, country: string }>} cities - The list of cities and their country codes.
 * @returns {Record<string, string | null>} An object with city names as keys and formatted times as values.
 */
export function getAllCitiesTime(cities) {
  const cityTimes = {};

  cities.forEach(({ city, country }) => {
    cityTimes[city] = getTimeByCity(city, country);
  });

  return cityTimes;
}
```

AddAlarm.js

```
import React, { useState } from "react";
import {
  ScrollView,
  TouchableOpacity,
  Text,
  View,
  TextInput,
  Switch,
} from "react-native";
import TimeSelector from "../components/TimeSelector";
import styles from "../styles/styles";

export default function AddAlarm({ closeModal, saveAlarm }) {
  const [time, setTime] = useState({ hour: 0, minute: 0 });
  const [alarmName, setAlarmName] = useState("");
  const [alarmSound, setAlarmSound] = useState(true);
  const [vibration, setVibration] = useState(true);

  const handleSave = () => {
    const formattedTime = `${time.hour
      .toString()
      .padStart(2, "0")}:${time.minute.toString().padStart(2, "0")}`;
    saveAlarm(formattedTime, alarmName, alarmSound, vibration);
    closeModal();
  };

  const handleTimeChange = (hour, minute) => {
    setTime({ hour, minute });
  };

  return (
    <ScrollView
      style={[
        styles.amcontainer,
        {
          backgroundColor: "#4A628A",
          borderTopLeftRadius: 50,
          borderTopRightRadius: 50,
          paddingTop: 30,
        },
      ]}
    >
      <View
        style={{
          flexDirection: "row",
          justifyContent: "space-between",
          paddingHorizontal: 20,
        }}
      >
        <TouchableOpacity
```

```
        style={{ backgroundColor: "transparent" }}
        onPress={closeModal}
      >
        <Text
          style={{
            fontSize: 36,
            color: "#fff",
            fontWeight: 500,
            textAlign: "left",
          }}
        >
          Cancel
        </Text>
      </TouchableOpacity>
      <TouchableOpacity
        style={{ backgroundColor: "transparent" }}
        onPress={handleSave}
      >
        <Text
          style={{
            fontSize: 36,
            color: "#fff",
            fontWeight: 500,
            textAlign: "right",
          }}
        >
          Save
        </Text>
      </TouchableOpacity>
    </View>
    <View style={{ padding: 25 }}>
      <TimeSelector onTimeChange={handleTimeChange} />
      <View style={{ marginBottom: 50 }} />
      <View style={styles.settingsContainer}>
        <TextInput
          style={styles.aminputTransparent}
          placeholder="Alarm name"
          placeholderTextColor="#B9B9B9"
          value={alarmName}
          onChangeText={setAlarmName}
        />
        <View style={styles.amseparator} />
        <View style={styles.settingContainer}>
          <Text style={[styles.settingText, { marginLeft: 10 }]}>
            Alarm sound
          </Text>
          <Switch
            value={alarmSound}
            onValueChange={setAlarmSound}
            thumbColor={alarmSound ? "#DCF5FC" : "#f4f3f4"}
            trackColor={{ false: "#767577", true: "#0166EF" }}
          />
        </View>
```

```
104.          <View style={styles.amseparator} />
105.          <View style={styles.settingContainer}>
106.            <Text style={[styles.settingText, { marginLeft: 10 }]}>
107.              Vibration
108.            </Text>
109.            <Switch
110.              value={vibration}
111.              onValueChange={setVibration}
112.              thumbColor={vibration ? "#DCF5FC" : "#f4f3f4"}
113.              trackColor={{ false: "#767577", true: "#0166EF" }}
114.            />
115.          </View>
116.          <View style={styles.amseparator} />
117.          <View style={{ marginBottom: 15 }} />
118.        </View>
119.      </View>
120.    </ScrollView>
121.  );
122. }
```

AddClock.js

```
import React, { useState, useEffect } from "react";
import { ScrollView, TouchableOpacity, Text, View, TextInput } from "react-native";
import AsyncStorage from "@react-native-async-storage/async-storage";
import styles from "../styles/styles";
import { Ionicons } from "@expo/vector-icons";
import axios from 'axios';

export default function AddClock({ navigation, closeModal, route }) {
  const [addedCities, setAddedCities] = useState([]);
  const [availableCities, setAvailableCities] = useState([]);
  const [searchQuery, setSearchQuery] = useState('');
  const [searchType, setSearchType] = useState('city'); // 'city' หรือ 'country'
  const { fromScreen } = route.params;

  useEffect(() => {
    const loadAddedCities = async () => {
      const storedCities = await AsyncStorage.getItem("addedCities");
      if (storedCities) {
        setAddedCities(JSON.parse(storedCities));
      }
    };
    loadAddedCities();
  }, []);

  useEffect(() => {
    const fetchCities = async () => {
      try {
        const response = await axios.get('http://10.0.2.2/chronobreak-backend/api.php');
        const data = response.data;
        setAvailableCities(data.filter(
          (city) => !addedCities.some((addedCity) => addedCity.city === city.city)
        ));
      } catch (error) {
        console.error('Error fetching cities:', error);
      }
    };
    fetchCities();
  }, [addedCities]);

  const addCity = async (city, country) => {
    const newCities = [...addedCities, { city, country }];
    await AsyncStorage.setItem("addedCities", JSON.stringify(newCities));
    closeModal();
    navigation.navigate(fromScreen, { city, country });
  };

  // กรองรายการเมืองตามคำค้นหาและประเภทการค้นหา
  const filteredCities = availableCities.filter(city => {
```

```
    if (!searchQuery) return true;

    if (searchType === 'city') {
      return city.city.toLowerCase().includes(searchQuery.toLowerCase());
    } else { // searchType === 'country'
      return city.name.toLowerCase().includes(searchQuery.toLowerCase());
    }
  });

  return (
    <ScrollView
      style={[
        styles.amcontainer,
        {
          backgroundColor: "#4A628A",
          borderTopLeftRadius: 50,
          borderTopRightRadius: 50,
          paddingTop: 30,
        },
      ]}
    >
      <View
        style={{
          flexDirection: "row",
          justifyContent: "flex-end",
          paddingHorizontal: 20,
        }}
      >
        <TouchableOpacity
          style={{ backgroundColor: "transparent" }}
          onPress={closeModal}
        >
          <Text
            style={{
              fontSize: 36,
              color: "#fff",
              fontWeight: 500,
              textAlign: "right",
            }}
          >
            Cancel
          </Text>
        </TouchableOpacity>
      </View>

      {/* SearchBar และ Toggle Button */}
      <View style={searchStyles.searchContainer}>
        <View style={searchStyles.searchInputContainer}>
          <Ionicons name="search" size={20} color="#777" style={searchStyles.searchIcon} />
          <TextInput
            style={searchStyles.searchInput}
            placeholder={`Search by ${searchType === 'city' ? 'city' : 'country'}...`}
            placeholderTextColor="#777"
```

```
102.           value={searchQuery}
103.           onChangeText={setSearchQuery}
104.         />
105.         {searchQuery ? (
106.           <TouchableOpacity onPress={() => setSearchQuery('')}>
107.             <Ionicons name="close-circle" size={20} color="#777" />
108.           </TouchableOpacity>
109.         ) : null}
110.       </View>
111.
112.       <View style={searchStyles.toggleContainer}>
113.         <TouchableOpacity
114.           style={[searchStyles.toggleButton, searchType === 'city' && searchStyles.activeToggle]}
115.           onPress={() => setSearchType('city')}
116.         >
117.           <Text style={[searchStyles.toggleText, searchType === 'city' &&
searchStyles.activeToggleText]}>
118.             City
119.           </Text>
120.         </TouchableOpacity>
121.         <TouchableOpacity
122.           style={[searchStyles.toggleButton, searchType === 'country' &&
searchStyles.activeToggle]}
123.           onPress={() => setSearchType('country')}
124.         >
125.           <Text style={[searchStyles.toggleText, searchType === 'country' &&
searchStyles.activeToggleText]}>
126.             Country
127.           </Text>
128.         </TouchableOpacity>
129.       </View>
130.     </View>
131.
132.     <View style={{ marginBottom: 100 }}>
133.       {filteredCities.map((city, index) => (
134.         <TouchableOpacity
135.           key={index}
136.           style={styles.cityItem}
137.           onPress={() => addCity(city.city, city.country)}
138.         >
139.           <Text
140.             style={{
141.               fontSize: 26,
142.               fontWeight: "bold",
143.               padding: 5,
144.               color: "#fff",
145.             }}
146.           >
147.             {city.city}, {city.name}
148.           </Text>
149.         </TouchableOpacity>
150.       ))}
151.     </View>
```

```
    </ScrollView>
  );
}

// สไตล์สำหรับ SearchBar และ Toggle Button
const searchStyles = {
  searchContainer: {
    padding: 16,
    marginBottom: 10,
  },
  searchInputContainer: {
    backgroundColor: '#fff',
    borderRadius: 12,
    flexDirection: 'row',
    alignItems: 'center',
    paddingHorizontal: 10,
    marginBottom: 10,
  },
  searchIcon: {
    marginRight: 8,
  },
  searchInput: {
    flex: 1,
    padding: 12,
    fontSize: 16,
    color: '#333',
  },
  toggleContainer: {
    flexDirection: 'row',
    backgroundColor: '#2a3c5c',
    borderRadius: 8,
    marginVertical: 8,
    padding: 4,
  },
  toggleButton: {
    flex: 1,
    paddingVertical: 8,
    alignItems: 'center',
  },
  activeToggle: {
    backgroundColor: '#fff',
    borderRadius: 8,
  },
  toggleText: {
    color: '#ddd',
    fontWeight: 'bold',
    fontSize: 16,
  },
  activeToggleText: {
    color: '#2a3c5c',
  }
};

```

Alarm.js

```
import React, { useEffect, useState } from "react";
import { useNavigation, useIsFocused } from "@react-navigation/native";
import {
  View,
  Text,
  ScrollView,
  TouchableOpacity,
  RefreshControl,
  Modal,
  Vibration,
} from "react-native";
import styles from "../styles/styles";
import { Ionicons } from "@expo/vector-icons";
import AsyncStorage from "@react-native-async-storage/async-storage";
import AddAlarm from "./AddAlarm";
import AlarmCard from "../components/AlarmCard";
import { Audio } from "expo-av";

export default function Alarm() {
  const [refreshing, setRefreshing] = useState(false);
  const navigation = useNavigation();
  const [alarms, setAlarms] = useState([]);
  const isFocused = useIsFocused();
  const [modalVisible, setModalVisible] = useState(false);
  const [deletingAlarm, setDeletingAlarm] = useState(null);
  const [showDeleteButtons, setShowDeleteButtons] = useState(false);
  const [sound, setSound] = useState(null);

  useEffect(() => {
    const refreshControl = async () => {
      setRefreshing(true);
      const storedAlarms = await AsyncStorage.getItem("alarms");
      if (storedAlarms) {
        setAlarms(JSON.parse(storedAlarms));
      }
      setRefreshing(false);
    };
  }, []);

  useEffect(() => {
    const loadAlarms = async () => {
      const storedAlarms = await AsyncStorage.getItem("alarms");
      if (storedAlarms) {
        setAlarms(JSON.parse(storedAlarms));
      }
    };
    if (isFocused) {
      loadAlarms();
```

```
    }
  }, [isFocused]);

  useEffect(() => {
    let previousMinute = new Date().getMinutes();

    const checkAlarms = async () => {
      const now = new Date();
      const currentTime = `${now.getHours().toString().padStart(2, "0")}:${now
        .getMinutes()
        .toString()
        .padStart(2, "0")}`;
      const currentMinute = now.getMinutes();

      if (currentMinute !== previousMinute) {
        const updatedAlarms = [...alarms];
        for (const alarm of updatedAlarms) {
          if (alarm.isEnabled && alarm.time === currentTime) {
            alarm.isEnabled = false;
            if (alarm.sound) {
              await playSound();
            }
            if (alarm.vibration) {
              Vibration.vibrate([1000, 1000, 1000, 1000]);
            }
          }
        }
        setAlarms(updatedAlarms);
        await AsyncStorage.setItem("alarms", JSON.stringify(updatedAlarms));
        previousMinute = currentMinute;
      }
    };

    const interval = setInterval(checkAlarms, 1000);
    return () => clearInterval(interval);
  }, [alarms]);

  const playSound = async () => {
    const { sound } = await Audio.Sound.createAsync(
      require("../assets/alarm-sound.mp3")
    );
    setSound(sound);
    await sound.playAsync();
  };

  const saveAlarm = async (time, name, sound, vibration) => {
    const newAlarm = { time, name, sound, vibration, isEnabled: false };
    const newAlarms = [...alarms, newAlarm];
    setAlarms(newAlarms);
    await AsyncStorage.setItem("alarms", JSON.stringify(newAlarms));
  };

  const toggleDeleteButtons = () => {
```

```
    setShowDeleteButtons(!showDeleteButtons);
    setDeletingAlarm(null);
  };

  const handleDeleteAlarm = async (time) => {
    const newAlarms = alarms.filter((alarm) => alarm.time !== time);
    await AsyncStorage.setItem("alarms", JSON.stringify(newAlarms));
    setAlarms(newAlarms);
    setDeletingAlarm(null);
    setShowDeleteButtons(false);
  };

  const toggleAlarm = async (time) => {
    const newAlarms = alarms.map((alarm) =>
      alarm.time === time ? { ...alarm, isEnabled: !alarm.isEnabled } : alarm
    );
    setAlarms(newAlarms);
    await AsyncStorage.setItem("alarms", JSON.stringify(newAlarms));
  };

  return (
    <ScrollView
      style={styles.container}
      refreshControl={
        <RefreshControl
          refreshing={refreshing}
          onRefresh={() => {
            RefreshControl;
          }}
        />
      }
    >
      <View style={styles.headerContainer}>
        <Text style={styles.header}>Alarm</Text>
        <View style={{ flexDirection: "row" }}>
          <TouchableOpacity
            style={styles.headerButton}
            onPress={() => setModalVisible(true)}
          >
            <Ionicons
              name="add-outline"
              style={styles.headerButtonText}
            ></Ionicons>
          </TouchableOpacity>
          <TouchableOpacity
            style={styles.headerButton}
            onPress={toggleDeleteButtons}
          >
            <Ionicons
              name="trash-bin-outline"
              style={styles.headerButtonText}
            ></Ionicons>
          </TouchableOpacity>
```

```
155.         </View>
156.       </View>
157.       {alarms.map((alarm, index) => (
158.         <AlarmCard
159.           key={index}
160.           time={alarm.time}
161.           name={alarm.name}
162.           sound={alarm.sound}
163.           vibration={alarm.vibration}
164.           deleting={deletingAlarm === alarm.time}
165.           onConfirmDelete={() => handleDeleteAlarm(alarm.time)}
166.           showDeleteButton={showDeleteButtons}
167.           isEnabled={alarm.isEnabled}
168.           toggleSwitch={() => toggleAlarm(alarm.time)}
169.         />
170.       ))}
171.       <Modal
172.         animationType="slide"
173.         transparent={true}
174.         visible={modalVisible}
175.         onRequestClose={() => {
176.           setModalVisible(!modalVisible);
177.         }}
178.       >
179.         <AddAlarm
180.           navigation={navigation}
181.           closeModal={() => setModalVisible(false)}
182.           saveAlarm={saveAlarm}
183.         />
184.       </Modal>
185.     </ScrollView>
186.   );
187. }
188.
```

Stopwatch.js

```
1. import React, { useState, useRef } from "react";
2. import { View, Text, TouchableOpacity, FlatList } from "react-native";
3.
4. import styles from "../styles/styles";
5.
6. export default function Stopwatch() {
7.   const [time, setTime] = useState(0);
8.   const [isRunning, setIsRunning] = useState(false);
9.   const [laps, setLaps] = useState([]);
10.   const timerRef = useRef(null);
11.
12.   const startStop = () => {
13.     if (isRunning) {
14.       if (timerRef.current !== null) {
15.         clearInterval(timerRef.current);
16.       }
17.     } else {
18.       const startTime = Date.now() - time;
19.       timerRef.current = setInterval(() => {
20.         setTime(Date.now() - startTime);
21.       }, 10);
22.     }
23.     setIsRunning(!isRunning);
24.   };
25.
26.   const reset = () => {
27.     if (timerRef.current !== null) {
28.       clearInterval(timerRef.current);
29.     }
30.     setTime(0);
31.     setLaps([]);
32.     setIsRunning(false);
33.   };
34.
35.   const recordLap = () => {
36.     if (isRunning) {
37.       setLaps([...laps, time]);
38.     }
39.   };
40.
41.   const formatTime = (milliseconds) => {
42.     const minutes = Math.floor(milliseconds / 60000);
43.     const seconds = Math.floor((milliseconds % 60000) / 1000);
44.     const centiseconds = Math.floor((milliseconds % 1000) / 10);
45.     return `${String(minutes).padStart(2, "0")}:${String(seconds).padStart(
```

```
    2,
    "0"
  )}.${String(centiseconds).padStart(2, "0")}`;
};

return (
  <View style={styles.stopwatchContainer}>
    <View style={styles.timerContainer}>
      <Text style={styles.timer}>{formatTime(time)}</Text>
    </View>
    <View style={styles.buttonRow}>
      {isRunning ? (
        <TouchableOpacity
          style={[styles.stopwatchButton, styles.lapButton]}
          onPress={recordLap}
        >
          <Text style={styles.buttonLap}>Lap</Text>
        </TouchableOpacity>
      ) : (
        <TouchableOpacity
          style={[styles.stopwatchButton, styles.resetButton]}
          onPress={reset}
        >
          <Text style={styles.buttonLap}>Reset</Text>
        </TouchableOpacity>
      )}
      <TouchableOpacity
        style={[
          styles.stopwatchButton,
          isRunning ? styles.stopButton : styles.startButton,
        ]}
        onPress={startStop}
      >
        <Text style={styles.buttonStart}>{isRunning ? "Stop" : "Start"}</Text>
      </TouchableOpacity>
    </View>
    <FlatList
      data={laps}
      renderItem={({ item, index }) => (
        <View style={styles.flatListContainer}>
          <View style={styles.lapContainer}>
            <Text style={styles.lapText}>Lap {index + 1}</Text>
            <Text style={styles.lapText}>{formatTime(item)}</Text>
          </View>
          <View style={styles.separatorContainer}>
            <View style={styles.separator} />
          </View>
        </View>
      )}
      keyExtractor={(item, index) => index.toString()}
    />
  </View>
);
```

```
99. }
```

Timer.js

```
1. import React, { useState, useEffect, useRef } from "react";
2. import { View, Text, Alert, TouchableOpacity } from "react-native";
3. import SelectTimer from "../components/SelectTimer";
4. import styles from "../styles/styles";
5. import { Audio } from "expo-av";
6.
7. export default function Timer() {
8.   const [hour, setHour] = useState(0);
9.   const [minute, setMinute] = useState(0);
10.   const [second, setSecond] = useState(0);
11.   const [remainingTime, setRemainingTime] = useState(0);
12.   const [isRunning, setIsRunning] = useState(false);
13.   const [isPaused, setIsPaused] = useState(false);
14.   const intervalRef = useRef(null);
15.   const [sound, setSound] = useState(null);
16.
17.   useEffect(() => {
18.     return sound
19.       ? () => {
20.           sound.unloadAsync();
21.         }
22.       : undefined;
23.   }, [sound]);
24.
25.   const handleTimeChange = (h, m, s) => {
26.     setHour(h);
27.     setMinute(m);
28.     setSecond(s);
29.     setRemainingTime(h * 3600 + m * 60 + s);
30.   };
31.
32.   const startTimer = async () => {
33.     if (remainingTime > 0) {
34.       setIsRunning(true);
35.       setIsPaused(false);
36.       intervalRef.current = setInterval(() => {
37.         setRemainingTime((prev) => {
38.           if (prev <= 1) {
39.             clearInterval(intervalRef.current);
40.             setIsRunning(false);
41.             playSound();
42.             Alert.alert("Time is up!");
43.             return 0;
44.           }
```

```
45.           return prev - 1;
46.         });
47.       }, 1000);
48.     }
49.   };
50.
51.   const pauseTimer = () => {
52.     if (intervalRef.current) {
53.       clearInterval(intervalRef.current);
54.     }
55.     setIsPaused(true);
56.   };
57.
58.   const resumeTimer = () => {
59.     setIsPaused(false);
60.     startTimer();
61.   };
62.
63.   const deleteTimer = () => {
64.     pauseTimer();
65.     setRemainingTime(0);
66.     setIsRunning(false);
67.     setIsPaused(false);
68.   };
69.
70.   const playSound = async () => {
71.     const { sound } = await Audio.Sound.createAsync(
72.       require("../assets/notification.mp3")
73.     );
74.     setSound(sound);
75.     await sound.playAsync();
76.   };
77.
78.   const formatTime = (time) => {
79.     const h = Math.floor(time / 3600);
80.     const m = Math.floor((time % 3600) / 60);
81.     const s = time % 60;
82.     return `${h.toString().padStart(2, "0")}:${m
83.       .toString()
84.       .padStart(2, "0")}:${s.toString().padStart(2, "0")}`;
85.   };
86.
87.   return (
88.     <View style={styles.container}>
89.       <View style={styles.headerContainer}>
90.         <Text style={styles.header}>Timer</Text>
91.       </View>
92.       <View style={styles.timerDisplayContainer}>
93.         {isRunning ? (
94.           <Text style={styles.timerDisplay}>{formatTime(remainingTime)}</Text>
95.         ) : (
96.           <SelectTimer onTimeChange={handleTimeChange} />
97.         )}
```

```
      </View>
      <View style={styles.timerButtonContainer}>
        {isRunning ? (
          <View style={styles.timerRunningButtonContainer}>
            <TouchableOpacity
              style={styles.timerButtonDelete}
              onPress={deleteTimer}
            >
              <Text style={styles.timerButtonText}>Delete</Text>
            </TouchableOpacity>
            <TouchableOpacity
              style={styles.timerButton}
              onPress={isPaused ? resumeTimer : pauseTimer}
            >
              <Text style={styles.timerButtonText}>
                {isPaused ? "Resume" : "Pause"}
              </Text>
            </TouchableOpacity>
          </View>
        ) : (
          <TouchableOpacity style={styles.timerButton} onPress={startTimer}>
            <Text style={styles.timerButtonText}>Start</Text>
          </TouchableOpacity>
        )}
      </View>
    </View>
  );
}
```

WorldClock.js

```
import React, { useEffect, useState, useCallback } from "react";
import {
  View,
  Text,
  ScrollView,
  TouchableOpacity,
  RefreshControl,
  Modal,
} from "react-native";
import { useNavigation, useIsFocused } from "@react-navigation/native";
import AsyncStorage from "@react-native-async-storage/async-storage";
import { getAllCitiesTime } from "../utils/TimeUtils";
import { defaultCity } from "../data/cities";
import ClockCard from "../components/ClockCard";
import { Ionicons } from "@expo/vector-icons";
import styles from "../styles/styles";
import AddClock from "./AddClock";

export default function WorldClock() {
  const navigation = useNavigation();
  const isFocused = useIsFocused();
  const [modalVisible, setModalVisible] = useState(false);
  const [cityTimes, setCityTimes] = useState({});
  const [refreshing, setRefreshing] = useState(false);
  const [cities, setCities] = useState(defaultCity);
  const [deletingCity, setDeletingCity] = useState(null);
  const [showDeleteButtons, setShowDeleteButtons] = useState(false);

  const loadCityTimes = useCallback(() => {
    const validCities = cities.filter(city =>
      city && city.city && typeof city.city === 'string' && city.city.trim() !== ''
    );
    const times = getAllCitiesTime(validCities);
    setCityTimes(times);
  }, [cities]);

  useEffect(() => {
    const loadCities = async () => {
      const storedCities = await AsyncStorage.getItem("addedCities");
      if (storedCities) {
        const parsedCities = JSON.parse(storedCities);
        const uniqueCities = [...defaultCity, ...parsedCities].filter(
          (city, index, self) =>
```

```
            index === self.findIndex((c) => c.city === city.city)
          );
          setCities(uniqueCities);
        }
      };
      if (isFocused) {
        loadCities().then(loadCityTimes);
      }
    }, [isFocused, loadCityTimes]);

    const onRefresh = useCallback(() => {
      setRefreshing(true);
      loadCityTimes();
      setRefreshing(false);
    }, [loadCityTimes]);

    const toggleDeleteButtons = () => {
      setShowDeleteButtons(!showDeleteButtons);
      setDeletingCity(null);
    };

    const handleDeleteCity = async (city) => {
      const newCities = cities.filter((c) => c.city !== city);
      await AsyncStorage.setItem("addedCities", JSON.stringify(newCities));
      setCities(newCities);
      setDeletingCity(null);
      setShowDeleteButtons(false);
    };

    return (
      <ScrollView
        style={styles.container}
        refreshControl={
          <RefreshControl refreshing={refreshing} onRefresh={onRefresh} />
        }
      >
        <View style={styles.headerContainer}>
          <Text style={styles.header}>World Clock</Text>
          <View style={{ flexDirection: "row" }}>
            <TouchableOpacity
              style={styles.headerButton}
              onPress={() => setModalVisible(true)}
            >
              <Ionicons
                name="add-outline"
                style={styles.headerButtonText}
              ></Ionicons>
            </TouchableOpacity>
            <TouchableOpacity
              style={styles.headerButton}
              onPress={toggleDeleteButtons}
            >
              <Ionicons
```

```
            name="trash-bin-outline"
            style={styles.headerButtonText}
          ></Ionicons>
        </TouchableOpacity>
      </View>
    </View>
    {Object.entries(cityTimes).map(([city, time]) => (
      <ClockCard
        key={city}
        city={city}
        time={time}
        deleting={deletingCity === city}
        onConfirmDelete={() => handleDeleteCity(city)}
        showDeleteButton={showDeleteButtons}
      />
    ))}
    <Modal
      animationType="slide"
      transparent={true}
      visible={modalVisible}
      onRequestClose={() => setModalVisible(!modalVisible)}
    >
      <AddClock
        navigation={navigation}
        closeModal={() => setModalVisible(false)}
        route={{ params: { fromScreen: "WorldClock" } }}
      />
    </Modal>
    <View style={{ height: 100 }} />
  </ScrollView>
 );
}
```

TimeZoneConverter.js

```
import React, { useState, useEffect } from "react";
import { View, Text, ScrollView, TouchableOpacity, Modal } from "react-native";
import AsyncStorage from "@react-native-async-storage/async-storage";
import { getTimeByCity } from "../utils/TimeUtils";
import TimeSelector from "../components/TimeSelector";
import { Ionicons } from "@expo/vector-icons";
import AddClock from "./AddClock";
import styles from "../styles/styles";

export default function TimeZoneConverter({ navigation }) {
  const [cities, setCities] = useState([]);
  const [selectedCity, setSelectedCity] = useState("Bangkok");
  const [selectedTime, setSelectedTime] = useState(new Date());
  const [convertedTimes, setConvertedTimes] = useState({});
  const [modalVisible, setModalVisible] = useState(false);

  useEffect(() => {
    const loadCities = async () => {
      const storedCities = await AsyncStorage.getItem("addedCities");
      if (storedCities) {
        setCities(JSON.parse(storedCities));
      }
    };
    loadCities();
  }, []);

  useEffect(() => {
    const interval = setInterval(() => {
      if (selectedCity) {
        const newConvertedTimes = {};
        cities.forEach((city) => {
          const time = getTimeByCity(city.city, city.country, selectedTime);
          newConvertedTimes[city.city] = time;
        });
        setConvertedTimes(newConvertedTimes);
      }
    }, 1000);

    return () => clearInterval(interval);
  }, [selectedCity, selectedTime, cities]);
```

```
41.
42.  const handleTimeChange = (hour, minute) => {
43.    const updatedTime = new Date(selectedTime);
44.    updatedTime.setHours(hour);
45.    updatedTime.setMinutes(minute);
46.    setSelectedTime(updatedTime);
47.  };
48.
49.  return (
50.    <View style={styles.container}>
51.      <View style={styles.headerContainer}>
52.        <Text style={styles.header}>Time Zone Converter</Text>
53.      </View>
54.      <View style={styles.timezoneMain}>
55.        <Text style={{ fontSize: 28, fontWeight: "bold", color: "#fff" }}>
56.          Bangkok
57.        </Text>
58.        <TouchableOpacity
59.          style={styles.headerButton}
60.          onPress={() => setModalVisible(true)}
61.        >
62.          <Ionicons
63.            name="add-outline"
64.            style={styles.headerButtonText}
65.          ></Ionicons>
66.        </TouchableOpacity>
67.      </View>
68.      <View style={styles.timeSelectorContainer}>
69.        <TimeSelector onTimeChange={handleTimeChange} />
70.      </View>
71.      <View style={styles.timezoneContainer}>
72.        <ScrollView>
73.          {Object.entries(convertedTimes).map(([city, time]) => (
74.            <View key={city} style={styles.convertedTimeContainer}>
75.              <Text style={styles.convertedText}>{city}</Text>
76.              <Text style={styles.convertedTime}>{time}</Text>
77.            </View>
78.          ))}
79.        </ScrollView>
80.      </View>
81.      <Modal
82.        animationType="slide"
83.        transparent={true}
84.        visible={modalVisible}
85.        onRequestClose={() => setModalVisible(!modalVisible)}
86.      >
87.        <AddClock
88.          navigation={navigation}
89.          closeModal={() => setModalVisible(false)}
90.          route={{ params: { fromScreen: "TimeZoneConverter" } }}
91.        />
92.      </Modal>
93.    </View>
```

```
94.   );
95. }
```

AlarmNavigator.js

```
 1. import React from "react";
 2. import {
 3.   createStackNavigator,
 4.   TransitionSpecs,
 5.   CardStyleInterpolators,
 6. } from "@react-navigation/stack";
 7. import Alarm from "../screens/Alarm";
 8. import AddAlarm from "../screens/AddAlarm";
 9.
10. const Stack = createStackNavigator();
11.
12. export default function AlarmNavigator() {
13.   return (
14.     <Stack.Navigator
15.       initialRouteName="Alarm"
16.       screenOptions={{
17.         headerShown: false,
18.         transitionSpec: {
19.           open: TransitionSpecs.TransitionIOSSpec,
20.           close: TransitionSpecs.TransitionIOSSpec,
21.         },
22.         cardStyleInterpolator: CardStyleInterpolators.forVerticalIOS,
23.       }}
24.     >
25.       <Stack.Screen
26.         name="Alarm"
27.         component={Alarm}
28.         options={{ title: "Alarm" }}
29.       />
30.       <Stack.Screen
31.         name="CreateAlarm"
32.         component={AddAlarm}
33.         options={{ title: "Create Alarm" }}
34.       />
35.     </Stack.Navigator>
36.   );
37. }
38.
```

MainNavigator.js

```
import React from "react";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import { Ionicons } from "@expo/vector-icons";
import { View } from "react-native";
import Stopwatch from "../screens/Stopwatch";
import Alarm from "../screens/Alarm";
import Timer from "../screens/Timer";
import WorldClockNavigator from "./WorldClockNavigator";
import AlarmNavigator from "./AlarmNavigator";
import styles from "../styles/styles";
import TimeZoneNavigator from "./TimeZoneNavigator";

const Tab = createBottomTabNavigator();

export default function MainNavigator() {
  return (
    <Tab.Navigator
      screenOptions={({ route }) => ({
        tabBarIcon: ({ focused, color, size }) => {
          let iconName = "timer-outline";
          let iconStyle = {};

          if (route.name === "Stopwatch") {
            iconName = "timer-outline";
          } else if (route.name === "Alarm") {
            iconName = "alarm-outline";
          } else if (route.name === "World Clock") {
            iconName = "earth-outline";
            iconStyle = focused
              ? [styles.worldClockIcon, styles.worldClockIconFocused]
              : styles.worldClockIcon;
          } else if (route.name === "Timer") {
            iconName = "hourglass-outline";
          } else if (route.name === "TimeZone Converter") {
            iconName = "swap-horizontal-outline";
          }

          return (
            <View style={route.name === "World Clock" ? iconStyle : {}}>
```

```
            <Ionicons
              name={iconName}
              size={route.name === "World Clock" ? size * 2.2 : size * 1.2}
              color={color}
            />
          </View>
        );
      },
      tabBarActiveTintColor: "#000",
      tabBarInactiveTintColor: "#a9a9a9",
      tabBarStyle: {
        height: 70,
        paddingTop: 7,
      },
      tabBarLabelStyle: {
        width: "100%",
        textAlign: "center",
        fontSize: 12,
        fontFamily: "Poppins_700Bold",
      },
      headerShown: false,
    })}
    initialRouteName="World Clock"
  >
    <Tab.Screen name="Stopwatch" component={Stopwatch} />
    <Tab.Screen name="Alarm" component={AlarmNavigator} />
    <Tab.Screen name="World Clock" component={WorldClockNavigator} />
    <Tab.Screen name="Timer" component={Timer} />
    <Tab.Screen name="TimeZone" component={TimeZoneNavigator} />
  </Tab.Navigator>
);
}
```

TimeZoneNavigator.js

```
import React from "react";
import {
  createStackNavigator,
  TransitionSpecs,
  CardStyleInterpolators,
} from "@react-navigation/stack";
import TimeZoneConverter from "../screens/TimeZoneConverter";
import AddClock from "../screens/AddClock";

const Stack = createStackNavigator();

export default function TimeZoneNavigator() {
  return (
    <Stack.Navigator
      initialRouteName="TimeZoneConverter"
      screenOptions={{
        headerShown: false,
        transitionSpec: {
          open: TransitionSpecs.TransitionIOSSpec,
          close: TransitionSpecs.TransitionIOSSpec,
        },
        cardStyleInterpolator: CardStyleInterpolators.forVerticalIOS,
      }}
    >
      <Stack.Screen
        name="TimeZoneConverter"
        component={TimeZoneConverter}
        options={{ title: "Time Zone Converter" }}
      />
      <Stack.Screen
        name="AddClock"
        component={AddClock}
        options={{ title: "Add Clock" }}
      />
    </Stack.Navigator>
  );
}
```

WorldClockNavigator.js

```
1. import React from "react";
2. import {
3.   createStackNavigator,
4.   TransitionSpecs,
5.   CardStyleInterpolators,
6. } from "@react-navigation/stack";
7. import WorldClock from "../screens/WorldClock";
8. import AddClock from "../screens/AddClock";
9.
10. const Stack = createStackNavigator();
11.
12. export default function StackNavigator() {
13.   return (
14.     <Stack.Navigator
15.       initialRouteName="WorldClock"
16.       screenOptions={{
17.         headerShown: false,
18.         transitionSpec: {
19.           open: TransitionSpecs.TransitionIOSSpec,
20.           close: TransitionSpecs.TransitionIOSSpec,
21.         },
22.         cardStyleInterpolator: CardStyleInterpolators.forVerticalIOS,
23.       }}
24.     >
25.       <Stack.Screen
26.         name="WorldClock"
27.         component={WorldClock}
28.         options={{ title: "World Clock" }}
29.       />
30.       <Stack.Screen
31.         name="AddClock"
32.         component={AddClock}
33.         options={{ title: "Add Clock" }}
34.       />
35.     </Stack.Navigator>
36.   );
37. }
```

AlarmCard.js

```
import React from "react";
import { View, Text, StyleSheet, TouchableOpacity, Switch } from "react-native";

const AlarmCard = ({
  time,
  name,
  sound,
  vibration,
  deleting,
  onConfirmDelete,
  showDeleteButton,
  isEnabled,
  toggleSwitch,
}) => {
  return (
    <View style={[styles.card, deleting && styles.deletingCard]}>
      <View>
        <Text style={styles.nameText}>{name}</Text>
        <Text style={styles.timeText}>{time}</Text>
      </View>
      {showDeleteButton ? (
        <View style={styles.deletingContainer}>
          <TouchableOpacity
            onPress={onConfirmDelete}
            style={styles.deleteButton}
          >
            <Text style={styles.deleteButtonText}>Delete</Text>
          </TouchableOpacity>
        </View>
      ) : (
        <Switch
          trackColor={{ false: "#868B8F", true: "#0166EF" }}
          thumbColor={"#DCF5FC"}
          onValueChange={toggleSwitch}
          value={isEnabled}
        />
      )}
    </View>
  );
```

```
};

const styles = StyleSheet.create({
  card: {
    backgroundColor: "#C8E8FB",
    padding: 20,
    margin: 10,
    borderRadius: 10,
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.8,
    shadowRadius: 2,
    elevation: 1,
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center",
  },
  deletingCard: {
    borderColor: "red",
    borderWidth: 2,
  },
  timeText: {
    fontSize: 30,
    color: "#000",
  },
  nameText: {
    fontSize: 16,
    color: "#666",
  },
  detailText: {
    fontSize: 14,
    color: "#999",
  },
  deleteButton: {
    width: "100%",
    height: "100%",
    justifyContent: "center",
    alignItems: "center",
  },
  deletingContainer: {
    width: "40%",
    backgroundColor: "#db001f",
    borderTopEndRadius: 10,
    borderEndEndRadius: 10,
    justifyContent: "center",
    alignItems: "center",
    position: "absolute",
    right: 0,
    top: 0,
    bottom: 0,
  },
  deleteButtonText: {
    color: "white",
    fontSize: 24,
```

```
    fontWeight: "700",
  },
});

export default AlarmCard;
```

ClockCard.js

```
import React from "react";
import { View, Text, TouchableOpacity } from "react-native";
import styles from "../styles/styles";

const ClockCard = ({
  city,
  time = "N/A",
  onConfirmDelete,
  showDeleteButton,
}) => {
  return (
    <View style={styles.timeContainer}>
      <Text style={styles.cityName}>{city}</Text>
      {showDeleteButton ? (
        <View style={styles.deletingContainer}>
          <TouchableOpacity
            onPress={onConfirmDelete}
            style={styles.deleteButton}
          >
            <Text style={styles.deleteButtonText}>Delete</Text>
          </TouchableOpacity>
        </View>
      ) : (
        <Text style={styles.timeDisplay}>{time}</Text>
      )}
    </View>
  );
};

export default ClockCard;
```

SelectTimer.js

```
import React, { useState, useEffect, useRef, useCallback } from "react";
import { View, Text, ScrollView } from "react-native";
import styles from "../styles/styles";

const ITEM_HEIGHT = 60;
const BUFFER_COUNT_HOURS = 24 * 3;
const BUFFER_COUNT_MINUTES_SECONDS = 60 * 3;
const CENTER_OFFSET_HOURS = 24;
const CENTER_OFFSET_MINUTES_SECONDS = 60;

const SelectTimer = ({ onTimeChange }) => {
  const [hour, setHour] = useState(0);
  const [minute, setMinute] = useState(10);
  const [second, setSecond] = useState(0);

  const hourScrollRef = useRef(null);
  const minuteScrollRef = useRef(null);
  const secondScrollRef = useRef(null);

  useEffect(() => {
    onTimeChange(hour, minute, second);
  }, [hour, minute, second]);

  useEffect(() => {
    hourScrollRef.current?.scrollTo({
      y: (CENTER_OFFSET_HOURS + hour) * ITEM_HEIGHT,
      animated: false,
    });
    minuteScrollRef.current?.scrollTo({
      y: (CENTER_OFFSET_MINUTES_SECONDS + minute) * ITEM_HEIGHT,
      animated: false,
    });
    secondScrollRef.current?.scrollTo({
      y: (CENTER_OFFSET_MINUTES_SECONDS + second) * ITEM_HEIGHT,
      animated: false,
    });
  }, []); // Ensures initial scroll position is set when component mounts

  const handleScroll = useCallback(
```

```
    (event, type) => {
      const { contentOffset } = event.nativeEvent;
      const index = Math.round(contentOffset.y / ITEM_HEIGHT);

      if (type === "hour") {
        const adjustedIndex = ((index % 24) + 24) % 24;
        if (adjustedIndex !== hour) setHour(adjustedIndex);

        if (
          index < CENTER_OFFSET_HOURS - 12 ||
          index > CENTER_OFFSET_HOURS + 12
        ) {
          hourScrollRef.current?.scrollTo({
            y: (CENTER_OFFSET_HOURS + adjustedIndex) * ITEM_HEIGHT,
            animated: false,
          });
        }
      } else if (type === "minute") {
        const adjustedIndex = ((index % 60) + 60) % 60;
        if (adjustedIndex !== minute) setMinute(adjustedIndex);

        if (
          index < CENTER_OFFSET_MINUTES_SECONDS - 30 ||
          index > CENTER_OFFSET_MINUTES_SECONDS + 30
        ) {
          minuteScrollRef.current?.scrollTo({
            y: (CENTER_OFFSET_MINUTES_SECONDS + adjustedIndex) * ITEM_HEIGHT,
            animated: false,
          });
        }
      } else {
        const adjustedIndex = ((index % 60) + 60) % 60;
        if (adjustedIndex !== second) setSecond(adjustedIndex);

        if (
          index < CENTER_OFFSET_MINUTES_SECONDS - 30 ||
          index > CENTER_OFFSET_MINUTES_SECONDS + 30
        ) {
          secondScrollRef.current?.scrollTo({
            y: (CENTER_OFFSET_MINUTES_SECONDS + adjustedIndex) * ITEM_HEIGHT,
            animated: false,
          });
        }
      }
    },
    [hour, minute, second]
  ); // Dependency array ensures no unnecessary state changes

  const generateLoopedArray = (max, bufferCount) => {
    return Array.from({ length: bufferCount }, (_, i) => i % max);
  };

  return (
```

```
    <View style={styles.timeSelectorContainer}>
      <View style={styles.hmsContainer}>
        <Text style={styles.hmsText}>Hours</Text>
        <Text style={styles.hmsText}>Minutes</Text>
        <Text style={styles.hmsText}>Seconds</Text>
      </View>
      <View style={styles.wheelContainer}>
        <ScrollView
          ref={hourScrollRef}
          style={styles.scrollView}
          snapToInterval={ITEM_HEIGHT}
          decelerationRate="fast"
          showsVerticalScrollIndicator={false}
          onScroll={(event) => handleScroll(event, "hour")}
          scrollEventThrottle={16}
          contentContainerStyle={styles.scrollViewContent}
        >
          {generateLoopedArray(24, BUFFER_COUNT_HOURS).map((h, index) => (
            <View key={index} style={styles.scrollItem}>
              <Text
                style={
                  index % 24 === hour
                    ? styles.selectedItemText
                    : styles.itemText
                }
              >
                {h.toString().padStart(2, "0")}
              </Text>
            </View>
          ))}
        </ScrollView>
        <Text style={styles.colon}>:</Text>
        <ScrollView
          ref={minuteScrollRef}
          style={styles.scrollView}
          snapToInterval={ITEM_HEIGHT}
          decelerationRate="fast"
          showsVerticalScrollIndicator={false}
          onScroll={(event) => handleScroll(event, "minute")}
          scrollEventThrottle={16}
          contentContainerStyle={styles.scrollViewContent}
        >
          {generateLoopedArray(60, BUFFER_COUNT_MINUTES_SECONDS).map(
            (m, index) => (
              <View key={index} style={styles.scrollItem}>
                <Text
                  style={
                    index % 60 === minute
                      ? styles.selectedItemText
                      : styles.itemText
                  }
                >
                  {m.toString().padStart(2, "0")}
```

```
              </Text>
            </View>
          )
        )}
      </ScrollView>
      <Text style={styles.colon}>:</Text>
      <ScrollView
        ref={secondScrollRef}
        style={styles.scrollView}
        snapToInterval={ITEM_HEIGHT}
        decelerationRate="fast"
        showsVerticalScrollIndicator={false}
        onScroll={(event) => handleScroll(event, "second")}
        scrollEventThrottle={16}
        contentContainerStyle={styles.scrollViewContent}
      >
        {generateLoopedArray(60, BUFFER_COUNT_MINUTES_SECONDS).map(
          (s, index) => (
            <View key={index} style={styles.scrollItem}>
              <Text
                style={
                  index % 60 === second
                    ? styles.selectedItemText
                    : styles.itemText
                }
              >
                {s.toString().padStart(2, "0")}
              </Text>
            </View>
          )
        )}
      </ScrollView>
    </View>
  </View>
  );
};

export default SelectTimer;
```

TimeSelector.js

```
import React, { useState, useEffect, useRef, useMemo } from "react";
import { View, Text, ScrollView } from "react-native";
import styles from "../styles/styles";

const ITEM_HEIGHT = 60;
const BUFFER_MULTIPLIER = 5;
const TOTAL_HOURS = 24 * BUFFER_MULTIPLIER;
const TOTAL_MINUTES = 60 * BUFFER_MULTIPLIER;
const CENTER_OFFSET_HOURS = Math.floor(TOTAL_HOURS / 2);
const CENTER_OFFSET_MINUTES = Math.floor(TOTAL_MINUTES / 2);

const TimeSelector = ({ initialHour = 0, initialMinute = 0, onTimeChange }) => {
  const [hour, setHour] = useState(initialHour);
  const [minute, setMinute] = useState(initialMinute);

  const hourScrollRef = useRef(null);
  const minuteScrollRef = useRef(null);

  useEffect(() => {
    onTimeChange(hour, minute);
  }, [hour, minute]);

  useEffect(() => {
    hourScrollRef.current?.scrollTo({
      y: (CENTER_OFFSET_HOURS + initialHour) * ITEM_HEIGHT,
      animated: false,
    });
    minuteScrollRef.current?.scrollTo({
      y: (CENTER_OFFSET_MINUTES + initialMinute) * ITEM_HEIGHT,
      animated: false,
    });
  }, [initialHour, initialMinute]);

  const handleScrollEnd = (event, type) => {
    const { contentOffset } = event.nativeEvent;
    const index = Math.round(contentOffset.y / ITEM_HEIGHT);

    if (type === "hour") {
```

```
    const adjustedHour = index % 24;
    setHour(adjustedHour);

    if (index < 12 || index > TOTAL_HOURS - 12) {
      hourScrollRef.current?.scrollTo({
        y: CENTER_OFFSET_HOURS * ITEM_HEIGHT,
        animated: false,
      });
    }
  } else {
    const adjustedMinute = index % 60;
    setMinute(adjustedMinute);

    if (index < 30 || index > TOTAL_MINUTES - 30) {
      minuteScrollRef.current?.scrollTo({
        y: CENTER_OFFSET_MINUTES * ITEM_HEIGHT,
        animated: false,
      });
    }
  }
};

const generateLoopedArray = useMemo(
  () => ({
    hours: Array.from({ length: TOTAL_HOURS }, (_, i) => i % 24),
    minutes: Array.from({ length: TOTAL_MINUTES }, (_, i) => i % 60),
  }),
  []
);

return (
  <View style={styles.timeSelectorContainer}>
    <View style={styles.wheelContainer}>
      <ScrollView
        ref={hourScrollRef}
        style={styles.scrollView}
        snapToInterval={ITEM_HEIGHT}
        decelerationRate="fast"
        showsVerticalScrollIndicator={false}
        onMomentumScrollEnd={(event) => handleScrollEnd(event, "hour")}
        contentContainerStyle={styles.scrollViewContent}
      >
        {generateLoopedArray.hours.map((h, index) => (
          <View key={index} style={styles.scrollItem}>
            <Text
              style={
                index % 24 === hour
                  ? styles.selectedItemText
                  : styles.itemText
              }
            >
              {h.toString().padStart(2, "0")}
            </Text>
```

```
          </View>
        ))}
      </ScrollView>
      <View style={styles.colonContainer}>
        <Text style={styles.colon}>:</Text>
      </View>
      <ScrollView
        ref={minuteScrollRef}
        style={styles.scrollView}
        snapToInterval={ITEM_HEIGHT}
        decelerationRate="fast"
        showsVerticalScrollIndicator={false}
        onMomentumScrollEnd={(event) => handleScrollEnd(event, "minute")}
        contentContainerStyle={styles.scrollViewContent}
      >
        {generateLoopedArray.minutes.map((m, index) => (
          <View key={index} style={styles.scrollItem}>
            <Text
              style={
                index % 60 === minute
                  ? styles.selectedItemText
                  : styles.itemText
              }
            >
              {m.toString().padStart(2, "0")}
            </Text>
          </View>
        ))}
      </ScrollView>
    </View>
  </View>
  );
};
export default TimeSelector;
```

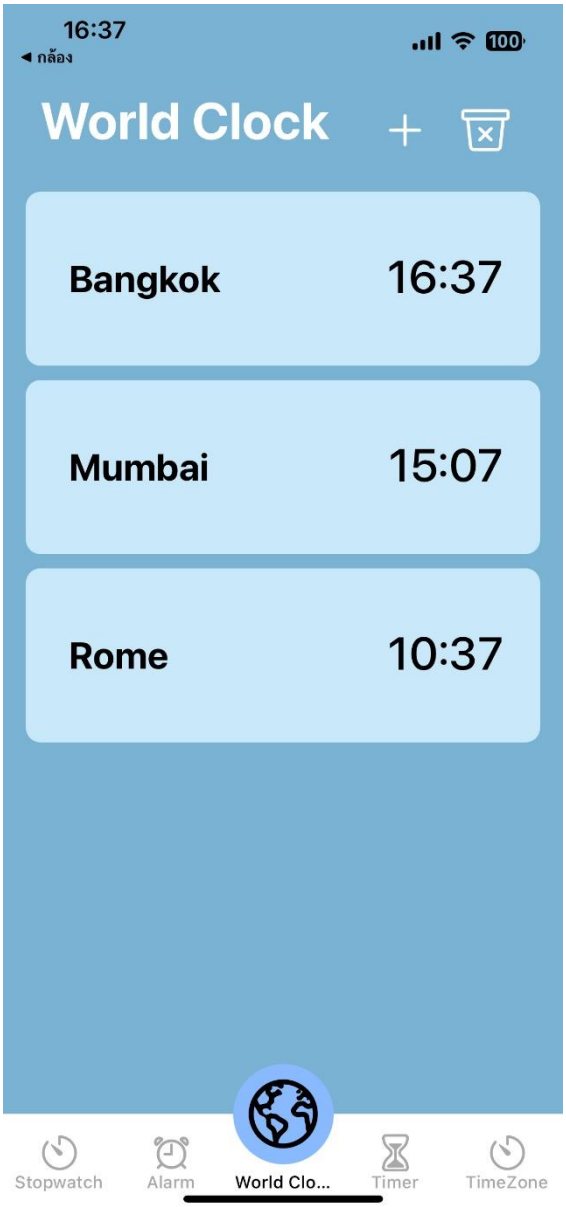
Color.js

```
const Colors = {
  backgroundColor: "#7ab2d3",
  secondaryColor: "#3c6ca5",
  buttonColor: "#158dec",
  textColor: "#fff",
  cardColor: "#c9e9fb",
  textCardColor: "#000",
  borderColor: "#ccc",
};
 export default Colors;
```

ผลลัพธ์การทำงานของโปรแกรม

https://github.com/MrWinRock/chronobreakjs

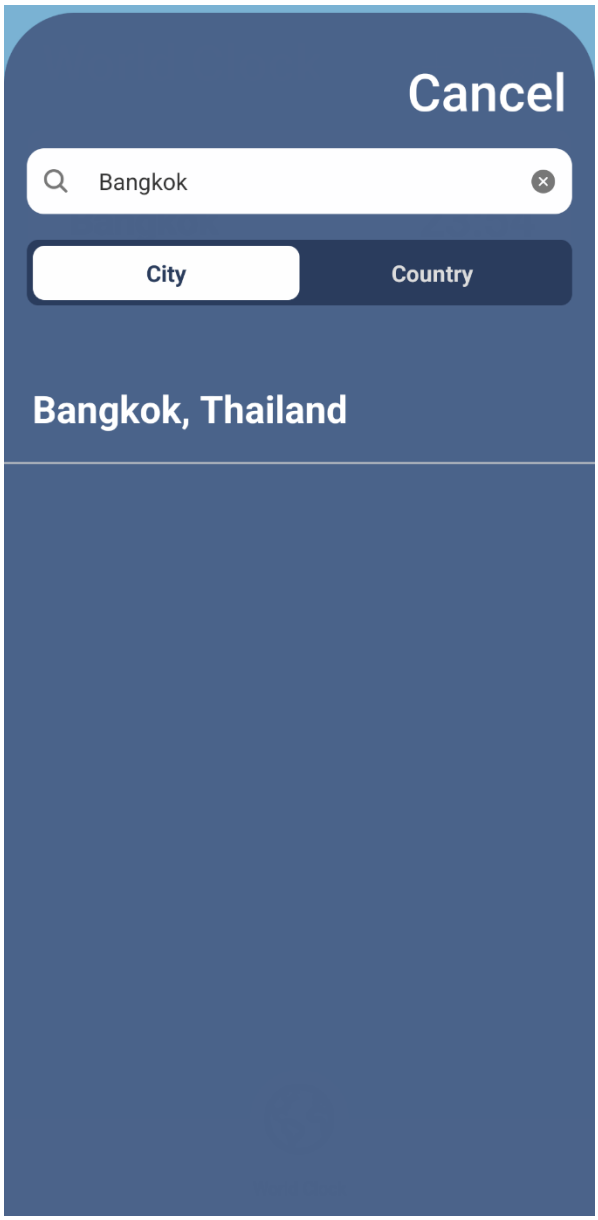
- World Clock



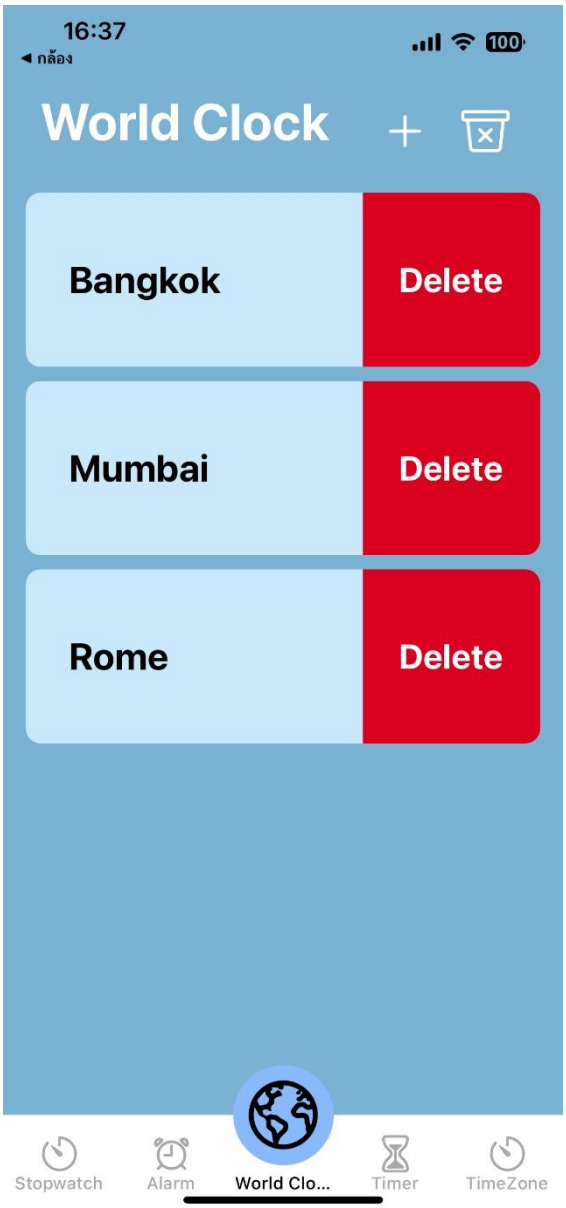
➢ Add Clock


Cancel
Search by city...
City
Country
Andorra la Vella, Andorra
Abu Dhabi, United Arab Emirates
Kabul, Afghanistan
Saint Johns, Antigua and Barbuda
The Valley, Anguilla
Tirana, Albania
Yerevan, Armenia

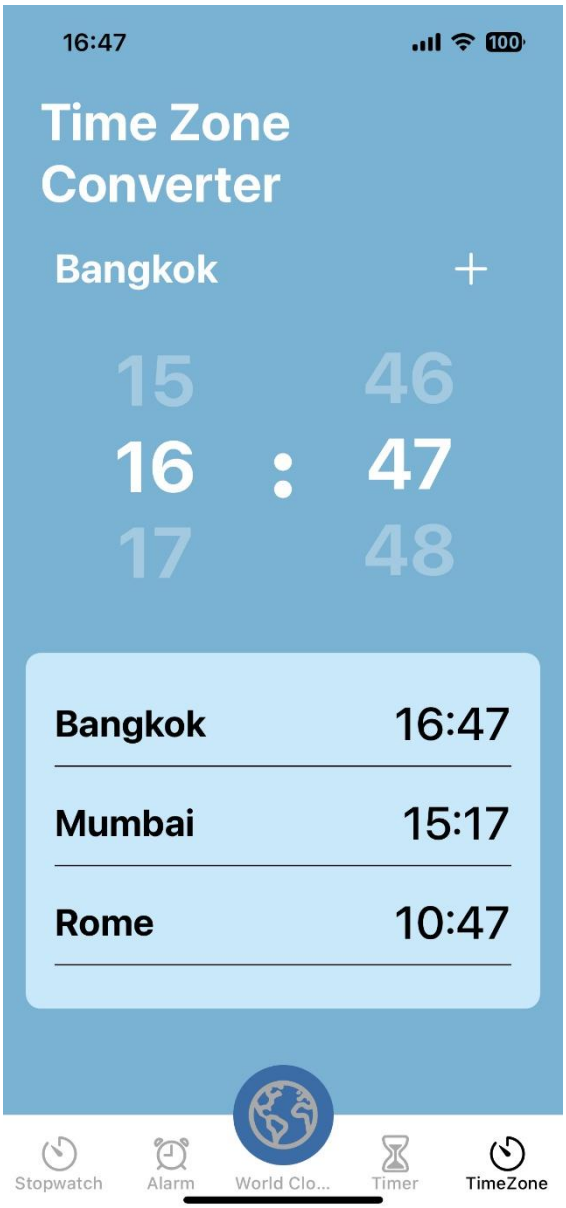
➢ Search Add Clock



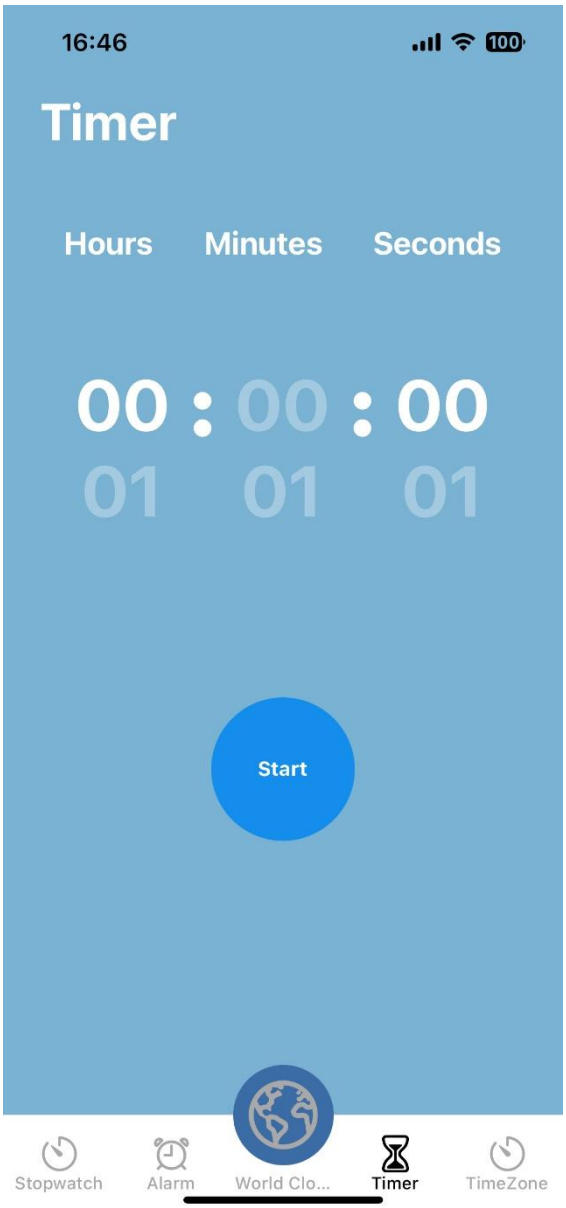
➢ Delete Clock



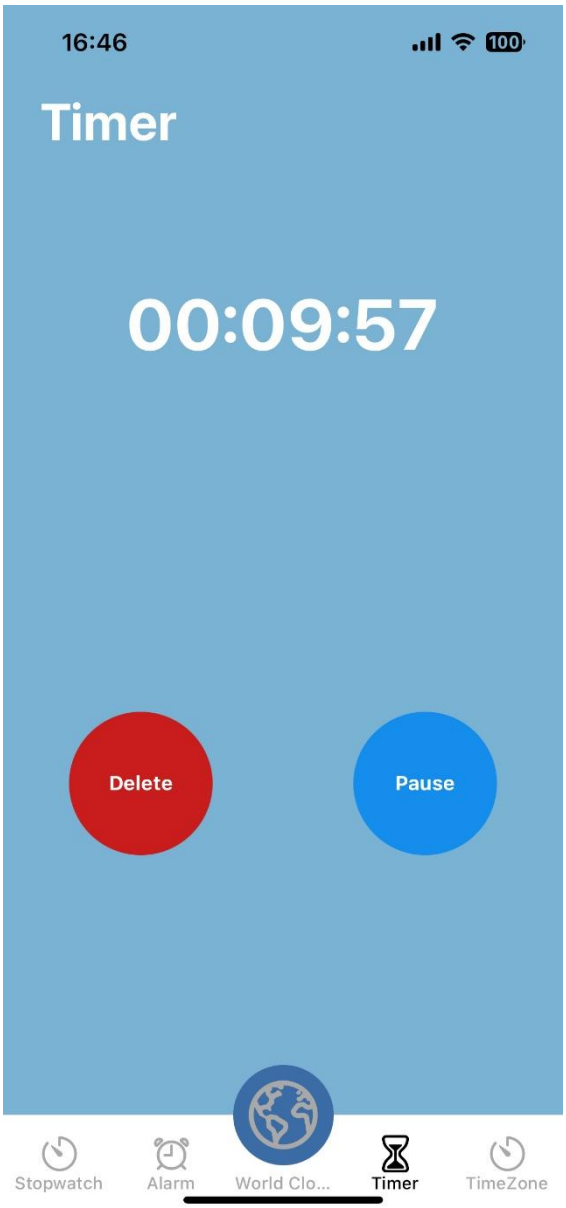
➢ Time Zone Converter



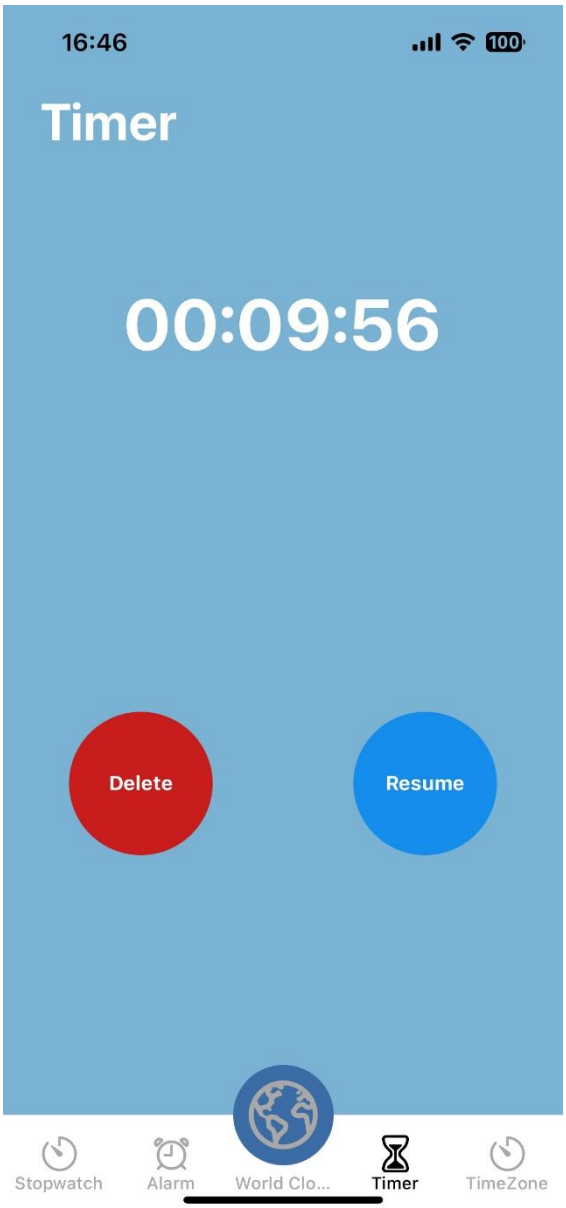
➢ Timer


16:46
100
Timer
Hours
Minutes
Seconds
00 : 00 : 00
01 01 01
Start
Stopwatch
Alarm
World Clo...
Timer
TimeZone

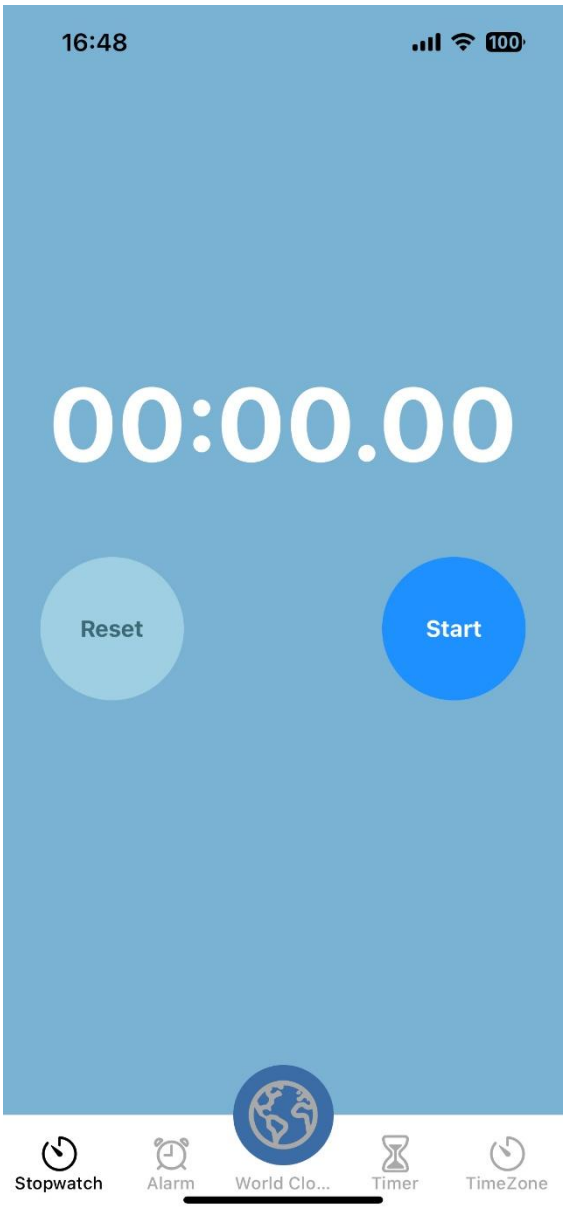
➢ Start Timer, Resume Timer



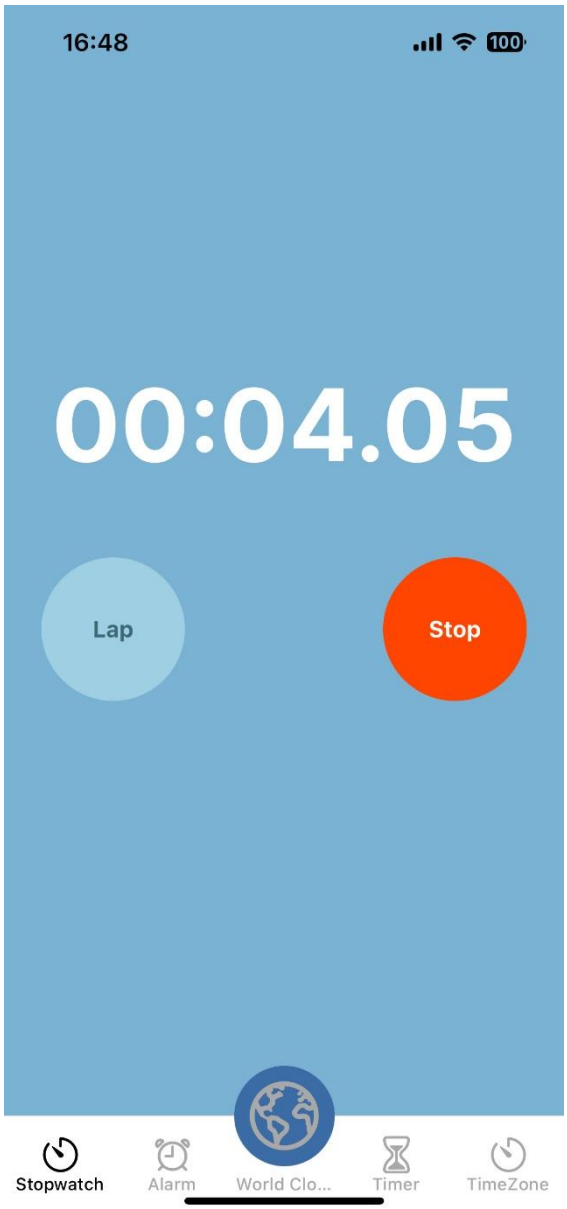
➢ Pause Timer, Delete Timer



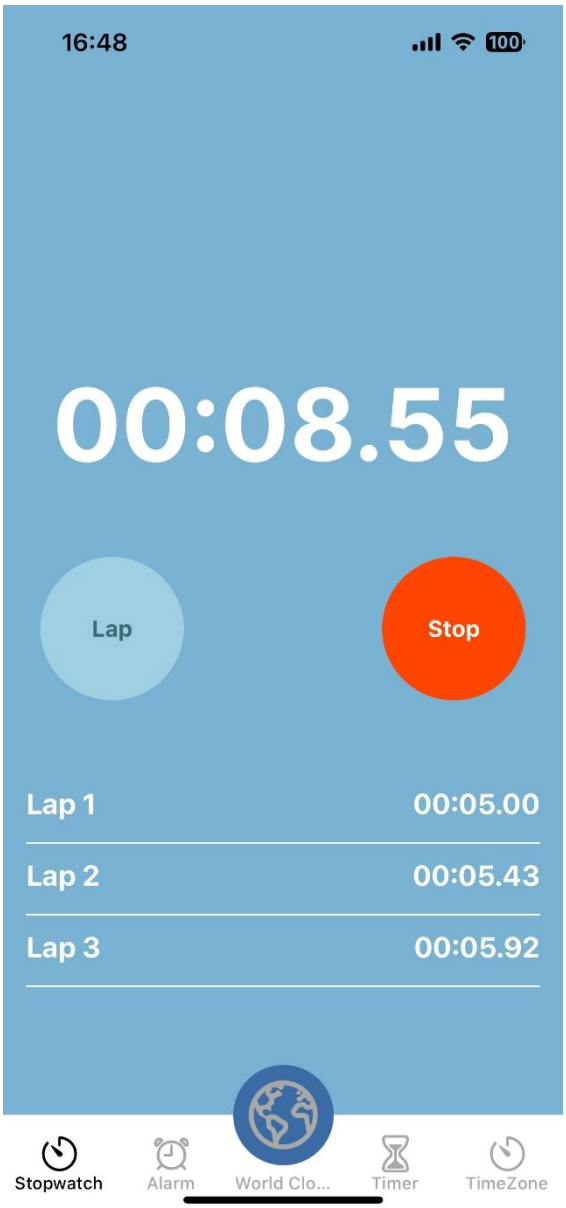
➢ Stopwatch, reset



➢ Start/Stop


16:48
100
00:04.05
Lap
Stop
Stopwatch
Alarm
World Clo...
Timer
TimeZone

➢ Record Lap

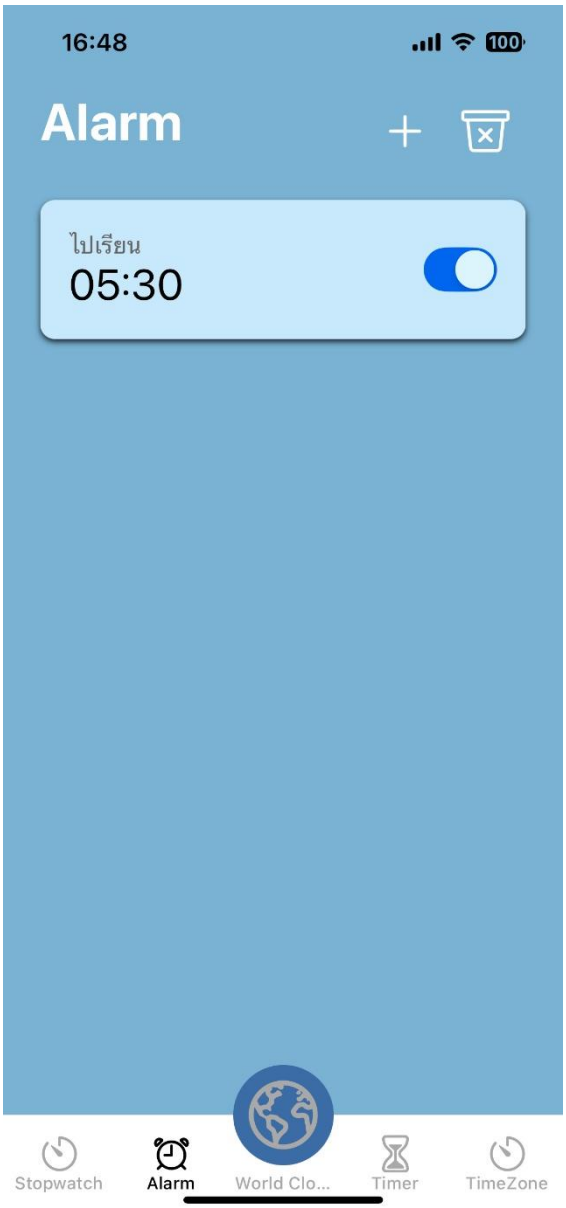

16:48
100
00:08.55
Lap
Stop
Lap 1
00:05.00
Lap 2
00:05.43
Lap 3
00:05.92
Stopwatch
Alarm
World Clo...
Timer
TimeZone

➢ Alarm

➢ Add Alarm


16:48
Cancel
Save
04
05
06
:
29
30
31
ไปเรียน
Alarm sound
Vibration

➢ Alarm Card


16:48
100
Alarm
ไปเรียน
05:30
Stopwatch
Alarm
World Clo...
Timer
TimeZone

➢ Delete Alarm