การพัฒนาแอปพลิเคชันบนมือถือโดยใช้ภาษา Flutter

แอปพลิเคชัน GlucoGuide

จัดทำโดย

นายกิตติศักดิ์ พุทธจันทร์ รหัสนักศึกษา 66109010183

นายจิรภาส ปราบมาก รหัสนักศึกษา 66109010185

นายภูธดา โตรัตน์ รหัสนักศึกษา 66109010198

นายวสุพล พิณวานิช รหัสนักศึกษา 66109010200

นายกิตกวิน แสงฤดี รหัสนักศึกษา 66109010445

นายนนทพัทธ์ เมฆทัศน์ รหัสนักศึกษา 66109010451

เสนอ

ผศ.ดร. ประมวล ชูรัตน์

รายงานนี้เป็นส่วนหนึ่งของรายวิชา CPE 121 Mobile Application Development

คณะวิศวกรรมศาสตร์ สาขาคอมพิวเตอร์

ภาคการศึกษาที่ 2 ปีการศึกษา 2566

มหาวิทยาลัยศรีนครินทรวิโรฒ

คำนำ

รายงานฉบับนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของรายวิชา CPE 121 Mobile Application Developments จัดทำขึ้นเพื่อนำเสนอเกี่ยวกับแอปพลิเคชัน GlucoGuide โดยรายงานฉบับนี้มีเนื้อหาประกอบด้วย ที่มาและความสำคัญ วัตถุประสงค์ ขอบเขตการทำงานของแอปพลิเคชัน การออกแบบหลักการการทำงานของแอปพลิเคชัน และการจำลองการทำงานของแอปพลิเคชัน

ทางคณะผู้จัดทำได้เลือกหัวข้อในการทำแอปพลิเคชันคำนวนการฉีดยาของผู้ป่วยเบาหวานชนิดที่ 2 (GlucoGuide) เนื่องจากในประเทศไทยพบผู้ป่วยที่เป็นโรคเบาหวานชนิดที่ 2 เป็นอันดับต้น ๆ ของประเทศ คณะผู้จัดทำหวังว่าแอปพลิเคชันนี้จะเป็นประโยชน์ต่อผู้ใช้งานไม่มากก็น้อย สุดท้ายนี้ทางคณะผู้จัดทำต้องขอขอบคุณอาจารย์ประมวล ชูรัตน์ ผู้ซึ่งเป็นอาจารย์ที่ปรึกษาและให้ความรู้ รวมไปถึงแนวทางการทำ แอปพลิเคชัน คณะผู้จัดหวังว่ารายงานฉบับนี้จะเป็นประโยชน์แก่ผู้อ่านทุกท่าน ไม่มากก็น้อย หากมีข้อผิดพลาดประการใดคณะผู้จัดทำต้องขออภัยมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ

ที่มาและความสำคัญ

ในปัจจุบันในประเทศไทยมีปริมาณผู้ป่วยที่เป็นโรคเบาหวานมากและมีแนวโน้มที่จะเพิ่มขึ้นเรื่อย ๆ ในช่วงไม่กี่ปีที่ผ่านมา จากข้อมูลของกรมควบคุมโรค กระทรวงสาธารณสุขในปี 2565 ระบุว่า ประมาณ 5.08% ของประชากรในประเทศไทยป่วยเป็นโรคเบาหวาน ซึ่งแปลว่ามีผู้ป่วยประมาณ 3.3 ล้านคน เพิ่มขึ้นจากปี พ.ศ. 2564 มากถึง 1.5 แสนคน และ มีแนวโน้มผู้ป่วยใหม่ปีละ 3 แสนคน ทั่วโลกมีผู้ป่วยโรคเบาหวานมากถึง 537 ล้านคน และ คาดว่าภายในปี 2573 จะเพิ่มขึ้นเป็น 643 ล้านคน และ โรคเบาหวานมีส่วนทำให้เสียชีวิต สูงถึง 6.7 ล้านคน หรือเสียชีวิต 1 คน ในทุก ๆ 5 วินาที ในประเทศไทย ปี 2563 มีผู้เสียชีวิตทั้งหมด 16,388 คน

ผู้ที่ป่วยเป็นโรคเบาหวานสามารถแบ่งออกเป็น 4 ประเภท ดังนี้

1. โรคเบาหวานชนิดที่ 1 เบาหวานขึ้นต่ออินซูลิน ("เบาหวานวัยเด็ก" หรือ Juvenile Diabetes)
2. โรคเบาหวานชนิดที่ 2 เบาหวานไม่ขึ้นต่ออินซูลิน ("เบาหวานผู้ใหญ่" หรือ Adult-Onset Diabetes, Non-Insulin Dependent Diabetes (NIDDM))
3. เบาหวานขณะตั้งครรภ์ (GDM)
4. โรคเบาหวานที่มีสาเหตุมาจากภาวะทางพันธุกรรม โรคทางกาย หรือการใช้ยาบางชนิด

ซึ่งในผู้ป่วยโรคเบาหวานชนิดที่ 1 เบาหวานขึ้นต่ออินซูลิน ("เบาหวานวัยเด็ก" หรือ Juvenile Diabetes) จำเป็นต้องฉีดอินซูลินตลอดชีวิต เนื่องจากการทำงานที่ผิดปกติของระบบภูมิคุ้มกันจนทำให้เกิดการทำลายเซลล์ที่ทำหน้าที่สร้างอินซูลินในตับอ่อน ส่งผลให้ร่างกายไม่สามารถสร้างอินซูลินได้อีกต่อไป ผู้ป่วยจึงต้องรักษาด้วยอินซูลินทุกวัน ซึ่งรวมถึงการฉีดอินซูลินก่อนอาหารหรือบางครั้งระหว่างอาหาร เพื่อควบคุมระดับน้ำตาลในเลือด

ผู้ป่วยโรคเบาหวานชนิดที่ 2 เบาหวานไม่ขึ้นต่ออินซูลิน ("เบาหวานผู้ใหญ่" หรือ Adult-Onset Diabetes, Non-Insulin Dependent Diabetes (NIDDM)) เป็นชนิดที่พบบ่อยที่สุด ผู้เป็นเบาหวานชนิดนี้ มักมีอายุมากกว่า 35 ปี สาเหตุเนื่องมาจากการที่ร่างกายตอบสนองอินซูลินได้ไม่ดีเท่าที่ควรหรือที่เรียกว่าภาวะดื้ออินซูลิน ร่วมกับการที่เบต้าเซลล์ของตับอ่อนถูกทำลาย โดยมักเกิดจากความอ้วน และถ่ายทอดได้ทางพันธุกรรม ส่งผลให้อินซูลินที่ทำหน้าที่ลดระดับน้ำตาลในเลือดทำงานได้ไม่ดีและปริมาณที่ร้างได้ลดลง เรื่อย ๆ บางรายอาจต้องใช้ยาเม็ด แต่บางรายอาจจำเป็นต้องฉีดอินซูลินหรือตัวยาฉีดอื่น ๆ เช่น GLP-1 receptor agonists (เช่น liraglutide) ซึ่งมักจะต้องฉีดก่อนอาหาร และ ประเภทอื่นอาจจะจำเป็นต้องฉีดยา กินยา หรือไม่ก็ได้ขึ้นอยู่กัยที่แพทย์แนะนำ

สำหรับผู้ป่วยที่ต้องฉีดอินซูลิน มักจะมีการฉีดอินซูลินชนิดต่าง ๆ เช่น อินซูลินระยะสั้น (Short-acting Insulin) อินซูลินระยะกลาง (Intermediate-acting Insulin) และ อินซูลินระยะยาว (Long-acting Insulin) โดยขึ้นอยู่กับระดับน้ำตาลในเลือดและการตอบสนองของร่างกาย การฉีดยาอินซูลินสำหรับผู้ป่วยโรคเบาหวาน โดยเฉพาะผู้ที่เป็นโรคเบาหวานชนิดที่ 1 และบางกรณีของโรคเบาหวานชนิดที่ 2 จำเป็นต้องมีการคำนวณปริมาณอินซูลินที่ต้องฉีดอย่างแม่นยำ เพื่อให้สามารถควบคุมระดับน้ำตาลในเลือดได้อย่างมีประสิทธิภาพ

คณะผู้จัดทำได้เห็นถึงความสำคัญของการคำนวณการฉีดยาอินซูลินในผู้ป่วยโรคเบาหวาน ทางคณะผู้จัดทำจึงนำความรู้ด้านการเขียนโปรแกรมด้วยภาษา Java จากการเรียนนำมาประยุกต์ใช้กับ Flutter และโปรแกรม Android studio เพื่อสร้าง Application ที่มีชื่อว่า "GlucoGuide" เพื่อผู้ที่ใช้จะสามารถคำนวณการฉีดยาอินซูลินได้อย่างแม่นยำ สะดวกสบาย ปลอดภัยมากขึ้น และให้คำแนะนำสำหรับผู้ที่ป่วยเป็นโรคเบาหวาน

วัตถุประสงค์

1. เพื่อเป็นเครื่องคำนวนค่าการฉีดยาหรือค่า dose ของผู้ป่วยเบาหวานชนิดที่สองที่ต้องฉีดยาก่อนอาหารทุกมื้อ
2. เก็บค่าการฉีดยาในแต่ละมื อ เพื่อสามารถนำไปเป็นข้อมูลช่วยในการรักษา
3. เพื่อควบคุมระดับน้ำตาลของผู้ป่วยให้คงที่ในแต่การฉีดยา

ขอบเขตของแอปพลิเคชัน

GlucoGuide เป็นแอปพลิเคชันที่จะช่วยในการเกิดควบคุมระดับน้ำตาลในเลือกให้คงที่ โดยจะคำนวนค่า dose ของ insulin ให้เป็นมาตราฐานตามสูตรของโรงพยาบาล รวมถึงยังสามารถบันทึกประวัติการฉีดยาในแต่ละมื้อเพื่อสามารถนำไปเป็นข้อมูลในการรักษาครั้งต่อไปของผู้ป่วยได้ โดยแอปพลิเคชันนี้เบื้องต้นสามารถใช้งานได้ทั้งระบบ Android และ iOS

การออกแบบและหลักการทำงานของแอปพลิเคชัน

GlucoGuide เป็นแอปพลิเคชันที่ออกแบบมาเพื่อคำนวนระดับน้ำตาลของผู้ป่วยรวมถึงการเก็บบันทึกการฉีด ซึ่งการทำงานที่สำคัญ จะประกอบไปด้วย 3 ฟังก์ชัน

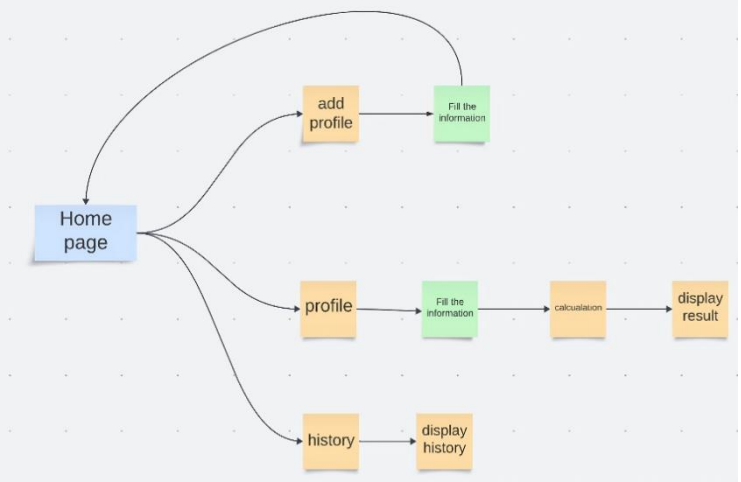
1. Profile : เป็นฟังชัน เก็บค่าส่วนตัว เช่น ชื่อ, ค่าความไว้ต่ออินซูลิน ลงในตัวเครื่อง รวมถึงการปรับแต่งแก้ไขโปรไฟล์ผู้ใช้

2. Calculation : เป็นฟังก์ชันการคำนวนยา โดย สูตรการคำนวน คือ (อินซูลินปัจจุบัน - อินซูลินเป้าหมาย) / ค่าความดือต่อ insulin, คาร์โบไฮเดรตที่กิน / สัดส่วนคาร์โบไฮเดรต

3. History : บันทึกผลลัพธ์ต่าง ๆ ที่ได้จากการคำนวน แสดงออกทางหน้าจอ

การทำงานในแต่ละส่วนของแอปพลิเคชัน (Coding)

- Block Diagram



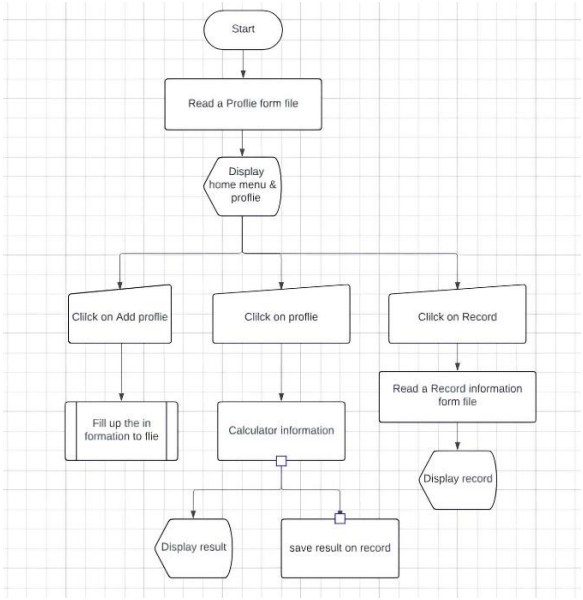
1. Home : เมื่อเข้าสู่แอปจะแสดงผล ปุ่มคำสั่ง ต่าง ๆ ประกอบด้วย 1) add profile 2) profile 3) history

2. Add profile : มีหน้าที่ไว้สำหรับการสร้างโปรไฟล์

3. Profile : มีหน้าที่ไว้สำหรับการกรอกค่าเพื่อนำไปคำนวณ

4. Calculation : มีหน้าที่ไว้สำหรับการนำค่าจากโปรไฟล์ของผู้ใช้มาคำนวณ

5. History : มีหน้าที่ไว้สำหรับแสดงผลการคำนวนย้อนหลัง



Coding

1. Main

1.1 ส่วนของไฟล์ (Files)

เป็นฟังก์ชันต่างๆที่ประกาศไว้เพื่อเรียกใช้งานในคำสั่งต่าง ๆ ที่เกี่ยวกับไฟล์ประกอบไปด้วย

_getFilePath ใช้ในการคืนค่า path ในการเก็บไฟล์ของโปรไฟล์

```
Future<String> _getFilePath() async {
  final directory = await getApplicationDocumentsDirectory();
  String path = '${directory.path}/profiles.txt';
  return path;
}
```

_getHistoryFilePath ใช้ในการคืนค่า path ในการเก็บไฟล์ของประวัติการคำนวน

```
Future<String> _getHistoryFilePath() async {
  final directory = await getApplicationDocumentsDirectory();
  return '${directory.path}/history.txt';
}
```

_loadProfiles เป็นฟังก์ชันที่ทำหน้าที่โหลดไฟล์ข้อมูลจากในเครื่องที่มีอยู่โดยเรียกผ่าน _getFilePath หลังจากนั้นใช้ setstate เพื่ออัปเดตลิสต์ของโปรไฟล์ นอกจากนี้ยังใช้ try-catch เพื่อดักจับ Error ในกรณีที่หากไม่พบไฟล์หรือไม่มีไฟล์อยู่

```
Future<void> _loadProfiles() async {
  try {
    final file = File(await _getFilePath());
    if (await file.exists()) {
      List<String> lines = await file.readAsLines();
      setState(() {
        _profiles = lines.map((line) => UserEntity.fromString(line)).toList();
      });
      print("Profiles loaded: $_profiles");
    } else {
      print("File does not exist.");
    }
  } catch (e) {
    print("Error loading profiles: $e");
  }
}
```

_saveProfiles เป็นฟังก์ชันที่ทำหน้าที่ในการ Save ไฟล์ของโปรไฟล์

```
Future<void> _saveProfiles() async {
  try {
    final file = File(await _getFilePath());
    String profilesString = _profiles.map((p) => p.toString()).join('\n');
    await file.writeAsString(profilesString);
    print("Profiles saved.");
  } catch (e) {
    print("Error saving profiles: $e");
  }
}
```

_saveHistory เป็นฟังก์ชันทำหน้าที่ในการ Save ไฟล์ของประวัติการคำนวณ

```
Future<void> _saveHistory() async {
  try {
    final file = File(await _getHistoryFilePath());
    final historyString = _history.map((e) => e.toString()).join('\n');
    await file.writeAsString(historyString);
    print("History saved.");
  } catch (e) {
    print("Error saving history: $e");
  }
}
```

_loadHistory เป็นฟังก์ชันที่ทำหน้าที่โหลดไฟล์ข้อมูลประวัติการคำนวนจากในเครื่องที่มีอยู่โดยเรียกผ่าน _getHistoryFilePath หลังจากนั้นใช้ setstate เพื่ออัปเดตลิสต์ของโปรไฟล์ นอกจากนี้ยังใช้ try-catch เพื่อดักจับ Error ในกรณีที่หากไม่พบไฟล์หรือไม่มีไฟล์อยู่

```
Future<void> _loadHistory() async {
  try {
    final file = File(await _getHistoryFilePath());
    if (await file.exists()) {
      final lines = await file.readAsLines();
      setState(() {
        _history =
            lines.map((line) => HistoryEntry.fromString(line)).toList();
      });
      print("History loaded: $_history");
    } else {
      print("History file does not exist.");
    }
  } catch (e) {
    print("Error loading history: $e");
  }
}
```

นอกจากนี้ยังมี _addProfile, _addHistory,_updateProfile, _deleteProfile ทำหน้าที่เพิ่มโปรไฟล์ เพิ่มประวัติการคำนวน อัปเดตโปรไฟล์และลบโปรไฟล์ตามลำดับ

```
void _addProfile(UserEntity profile) {
  setState(() {
    _profiles.add(profile);
    _saveProfiles();
  });
}

void _updateProfile(int index, UserEntity profile) {
  setState(() {
    _profiles[index] = profile;
    _saveProfiles();
  });
}

void _deleteProfile(int index) {
  setState(() {
    _profiles.removeAt(index);
    _saveProfiles();
  });
```

```
void _addHistory(HistoryEntry entry) {
  setState(() {
    _history.add(entry);
    _saveHistory();
  });
}
```

1.2 ส่วนของการแสดงผล

ทำหน้าที่ในการแสดงผลข้อมูล โดยจะประกอบไปด้วย AppBar ที่แสดงคำว่า (“Welcome To GlucoGuide”) ซึ่งจะมีการแสดงผลของ ListViewbuilder ซึ่งเป็นการแสดงลิสต์ของโปรไฟล์ หากผู้ใช้เริ่มต้นใช้งานจะไม่ขึ้นอะไร

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Welcome To GlucoGuide'),
      backgroundColor: const Color.fromARGB(255, 79, 133, 177),
    ), // AppBar
    body: ListView.builder(
      itemCount: _profiles.length,
      itemBuilder: (context, index) {
        return Container(
          margin: EdgeInsets.symmetric(vertical: 5, horizontal: 10),
          padding: EdgeInsets.all(10),
          decoration: BoxDecoration(
            border: Border.all(color: Colors.grey),
            borderRadius: BorderRadius.circular(5),
          ), // BoxDecoration
          child: ListTile(
            title: Text(_profiles[index].name),
            trailing: Row(
```

```dart
bottomNavigationBar: BottomNavigationBar(
  items: const <BottomNavigationBarItem>[
    BottomNavigationBarItem(
      icon: Icon(Icons.home),
      label: "Home",
    ), // BottomNavigationBarItem
    BottomNavigationBarItem(
      icon: Icon(Icons.book),
      label: "History",
    ), // BottomNavigationBarItem
  ], // <BottomNavigationBarItem>[]
  onTap: (index) {
    if (index == 1) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => HistoryScreen(
            history: _history,
```

2. Profile

2.1 การรับค่าของข้อมูล (กรอกข้อมูล)

ในการรับค่าของข้อมูลของโปรไฟล์ จะประกอบไปด้วย ชื่อ – นามสกุล อายุ ค่าความดื้อของอินซูลิน (แต่ละคนจะมีค่าเป็นของตนเอง อาจเปลี่ยนแปลงได้หลังพบแพทย์) ค่าคาร์โบไฮเดรตต่อกรัม

```dart
class _ProfileScreenState extends State<Profile> {
  final TextEditingController firstandlastName = TextEditingController();
  final TextEditingController age = TextEditingController();
  final TextEditingController InsuliSensitive = TextEditingController();
  final TextEditingController CarbPerUnit = TextEditingController();
```

```dart
  TextField(
    controller: firstandlastName,
    decoration: InputDecoration(labelText: 'โปรดใส่ ชื่อ - นามสกุล'),
  ), // TextField
  TextField(
    controller: age,
    decoration: InputDecoration(labelText: 'โปรดใส่ อายุ'),
  ), // TextField
  TextField(
    controller: InsuliSensitive,
    decoration:
        InputDecoration(labelText: 'โปรดใส่ค่าความดื้อต่ออินซูลิน'),
  ), // TextField
  TextField(
    controller: CarbPerUnit,
    decoration:
        InputDecoration(labelText: 'โปรดใส่ค่าคาร์โบไฮเดรต/กรัม'),
  ), // TextField
```

2.2 การแสดงผล

เมื่อมีการใส่ค่าข้อมูล firstandlastName, age, InsuliSensitive, CarbPerUnit โปรแกรมจะตรวจสอบว่าค่าที่ใส่เข้ามาเป็นค่าข้อมูลประเภทตรงกันหรือไม่ หรือ มีการใส่ข้อมูลครบทุกช่องหรือไม่

```
void _saveProfile() {
  try {
    // ตรวจสอบค่าชื่อ
    if (!RegExp(r'^[a-zA-Z\s]+$').hasMatch(firstandlastName.text)) {
      throw FormatException(
          'ท่านใส่ค่าข้อมูลไม่ถูกต้อง โปรดใส่ข้อมูลที่เป็นพยัญชนะภาษาไทยและภาษาอังกฤษ');
    }
    // ตรวจสอบค่าอายุ
    if (!RegExp(r'^\d+$').hasMatch(age.text)) {
      throw FormatException(
          'ท่านใส่ค่าข้อมูลไม่ถูกต้อง โปรดใส่ค่าข้อมูลเป็นตัวเลข');
    }
    // ตรวจสอบว่าฟิลด์ทั้งหมดไม่ว่างเปล่า
    if (firstandlastName.text.isNotEmpty &&
        age.text.isNotEmpty &&
        InsuliSensitive.text.isNotEmpty &&
        CarbPerUnit.text.isNotEmpty) {
      final user = UserEntity(
        firstandlastName.text,
        double.parse(age.text),
```

```
void _showErrorDialog(String message) {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text('Error'),
        content: Text(message),
        actions: <Widget>[
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
            },
            child: Text('OK'),
          ), // TextButton
        ], // <Widget>[]
      ); // AlertDialog
    },
```

3. Calculation

3.1 การรับค่าข้อมูล (ในการคำนวณ)

หลังจากที่ User กรอกข้อมูลส่วนตัวแล้ว ก็จะมีการใส่ค่าข้อมูลในการคำนวณ ซึ่งจะต้องมีการตรวจสอบค่าเลือด ณ ตอนนั้น ค่าข้อมูลที่ต้องกรอกประกอบด้วย currentBloodSugar, targetBloodSugar, carbValue

```
class _InputcalState extends State<Inputcal> {
  final TextEditingController currentBloodSugar = TextEditingController();
  final TextEditingController targetBloodSugar = TextEditingController();
  final TextEditingController carbValue = TextEditingController();
```

```
  TextField(
    controller: currentBloodSugar,
    decoration:
        InputDecoration(labelText: 'โปรดใส่ค่าของเลือดปัจจุบัน'),
    keyboardType: TextInputType.number,
  ), // TextField
  TextField(
    controller: targetBloodSugar,
    decoration: InputDecoration(
        labelText: 'โปรดใส่ค่าเป้าหมายของน้ำตาลในเลือด'), // InputDecoration
    keyboardType: TextInputType.number,
  ), // TextField
  TextField(
    controller: carbValue,
    decoration: InputDecoration(labelText: 'โปรดใส่ค่าคาร์โบไฮเดรต'),
    keyboardType: TextInputType.number,
  ), // TextField
```

3.2 การคำนวณ

หลังจากที่มีการรับค่าแล้ว ค่าของแต่ละตัวแปรจะไปที่ฟังก์ชัน _calculateDose จะมีการตรวจสอบค่าที่มีการรับเข้ามาว่าประเภทของข้อมูลตรงกันไหมหรือใส่ค่าครบทุกช่องหรือไม่

```
void _calculateDose() {
  try {
    if (currentBloodSugar.text.isEmpty) {
      throw FormatException('โปรดใส่ค่าของเลือดปัจจุบัน');
    }
    if (targetBloodSugar.text.isEmpty) {
      throw FormatException('โปรดใส่ค่าเป้าหมายของน้ำตาลในเลือด');
    }
    if (carbValue.text.isEmpty) {
      throw FormatException('โปรดใส่ค่าคาร์บ');
    }

    double insulinSensitivity = widget.userProfile.insulinSensitivity;
    double carbPerUnit = widget.userProfile.carbPerUnit;

    if (!RegExp(r'^\d+(\.\d+)?$').hasMatch(currentBloodSugar.text)) {
      throw FormatException('โปรดใส่ค่าของเลือดปัจจุบันเป็นตัวเลข');
    }
    if (!RegExp(r'^\d+(\.\d+)?$').hasMatch(targetBloodSugar.text)) {
      throw FormatException('โปรดใส่ค่าเป้าหมายของน้ำตาลในเลือดเป็นตัวเลข');
```

หลังจากนั้นแปลงค่าจาก String เป็น Double เพื่อนำไปใช้ในการคำนวณ จากนั้นส่งค่าไปที่คลาสของการคำนวน (mainCal)

```
double currentBS = double.parse(currentBloodSugar.text);
double targetBS = double.parse(targetBloodSugar.text);
double carbs = double.parse(carbValue.text);

mainCal calculator = mainCal(currentBS, targetBS, carbs);
```

ในการคำนวณเพื่อให้ได้ผลลัพธ์ที่ต้องการ จำเป็นต้องมีการตรวจสอบค่าที่นำมาใช้ว่าถูกต้องหรือไม่ เช่น ค่าความดื้อของอินซูลินต้องไม่มีค่าน้อยกว่าหรือเท่ากับ 0

```
double calculateCorrectionDose(double insulinSensitivity) {
  // Result Dose from blood sugar
  if (insulinSensitivity <= 0) {
    throw ArgumentError('Insulin sensitivity must be greater than zero.');
  }
  double correctionDose =
      (currentBloodSugar - targetBloodSugar) / insulinSensitivity;
  return correctionDose.ceilToDouble(); // round up
}

 double calculateCarbDose(double carbPerInsulin) {
   // Result Dose from carbs
   if (carbPerInsulin <= 0) {
     throw ArgumentError(
         'Carb per unit of insulin must be greater than zero.');
   }
   double carbDose = carbValue / carbPerInsulin;
   return carbDose.ceilToDouble(); // round up
 }

 double calculateTotalDose(double insulinSensitivity, double carbPerInsulin) {
   // Sum of the doses required
   double totalDose = calculateCorrectionDose(insulinSensitivity) +
       calculateCarbDose(carbPerInsulin);
   return totalDose.ceilToDouble(); // round up
 }
}
```

3.3 แสดงผลลัพธ์

หลังจากคำนวณคลาสของ mainCal แล้วจะส่งค่าผลลัพธ์ไปที่คลาสของ Displayresult ที่ฟังก์ชันของ showResultDialog ซึ่งจะแสดงค่าของ Correction Dose, Carb Dose, Total Dose

```
double correctionDose =
    calculator.calculateCorrectionDose(insulinSensitivity);
double carbDose = calculator.calculateCarbDose(carbPerUnit);
double totalDose =
    calculator.calculateTotalDose(insulinSensitivity, carbPerUnit);

Displayresult.showResultDialog(
    context, correctionDose, carbDose, totalDose);
```

```
class Displayresult {
  static void showResultDialog(BuildContext context, double correctionDose,
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: Text('Result of Dose'),
          content: Column(
            mainAxisSize: MainAxisSize.min,
            children: <Widget>[
              Text('Correction Dose: $correctionDose'),
              Text('Carb Dose: $carbDose'),
              Text('Total Dose: $totalDose'),
            ], // <Widget>[]
          ), // Column
          actions: <Widget>[
            TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text('OK'),
            ), // TextButton
```

4. ประวัติการคำนวณ

4.1 การคำนวณ

ค่าที่มาจากส่วนการคำนวณ นำมาตั้งค่ารูปแบบของการบันทึกไฟล์

```
final historyEntry = HistoryEntry(
  profileName: widget.userProfile.name,
  description:
      'Correction Dose: $correctionDose, Carb Dose: $carbDose, Total Dose: $totalDose',
  timestamp: DateTime.now(),
  result: totalDose
      .toString(), // Assuming you want to store the total dose as the result
); // HistoryEntry
widget.addHistory(historyEntry);
```

4.2 การแสดงผลลัพธ์

เป็นการแสดงประวัติการคำนวณ ว่าผู้คำนวณคือใคร ค่า Correction Dose, Carb Dose, Total Dose เท่าไหร่ เวลาเมื่อคำนวณกี่โมง

```
HistoryEntry({
  required this.profileName,
  required this.description,
  required this.timestamp,
  required String result,
});

@override
String toString() {
  return '$profileName|$description|${timestamp.toIso8601String()}';
}

static HistoryEntry fromString(String string) {
  final parts = string.split('|');
  return HistoryEntry(
    profileName: parts[0],
    description: parts[1],
    timestamp: DateTime.parse(parts[2]),
    result: '',
```

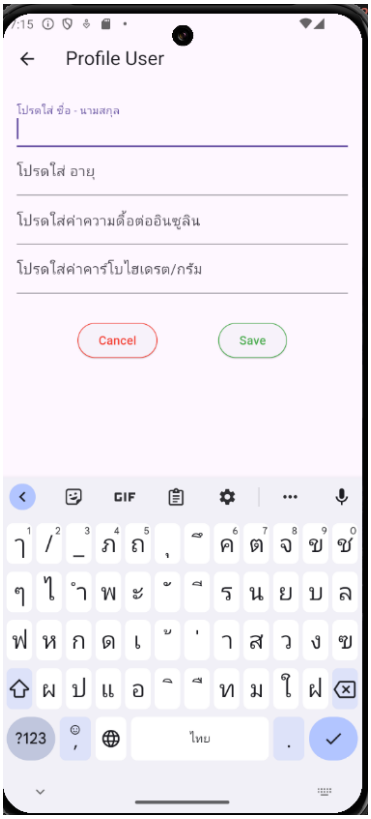
```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('History'),
    ), // AppBar
    body: ListView.builder(
      itemCount: _history.length,
      itemBuilder: (context, index) {
        return ListTile(
          title: Text(_history[index].description),
          subtitle: Text(_history[index].timestamp.toString()),
          trailing: IconButton(
            icon: Icon(Icons.delete),
            onPressed: () => _deleteHistory(index),
          ), // IconButton
        ); // ListTile
      },
    ), // ListView.builder
  ); // Scaffold
}
```

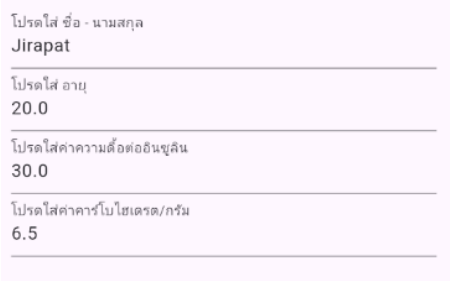
การจำลองการทำงานของแอปพลิเคชัน

1. Icon ของ Application

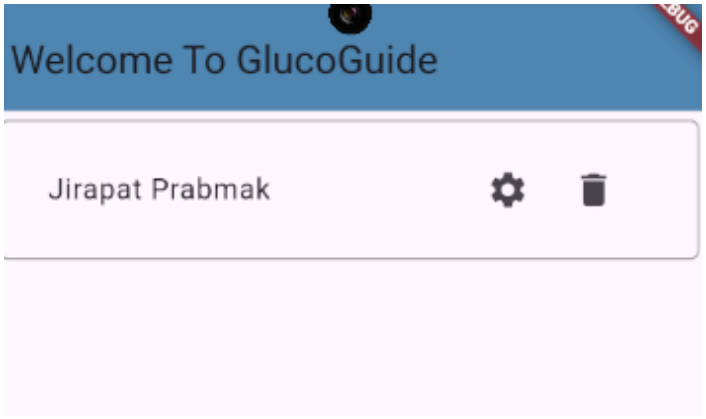
2. หน้าหลักของ Application

เมื่อกดปุ่ม + ที่หน้าหลักของแอปพลิเคชัน จะเป็นการให้ User ได้กรอกข้อมูลประวัติส่วนตัวและข้อมูลที่จำเป็นต่อการคำนวณแล้วกด Save

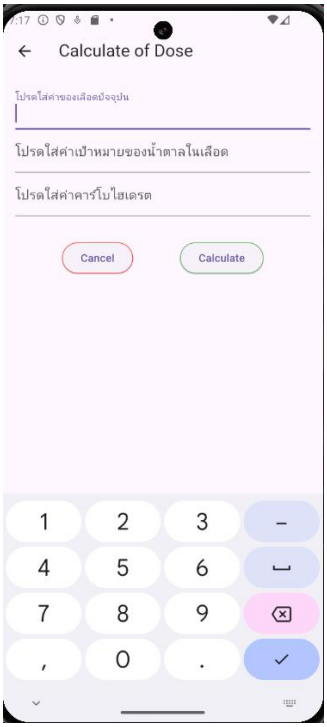


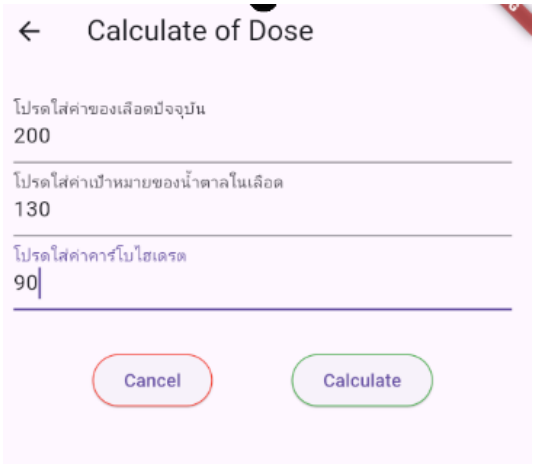


ข้อมูลของ User จะบันทึกลงไฟล์และแสดงในหน้าหลักของแอปพลิเคชัน หาก User ต้องการคำนวณอินซูลินของตน ต้องกดที่ชื่อของตนเองหรือบริเวณในกรอบของตน

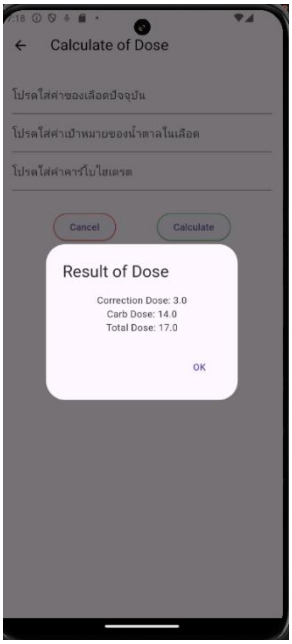


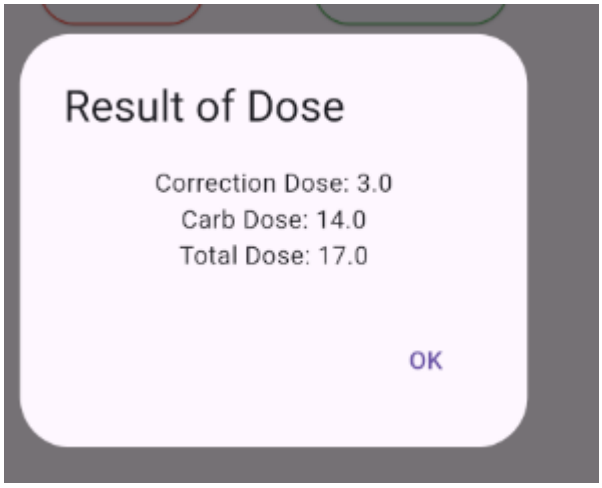
เมื่อกดแถบประวัติแล้วแอปพลิเคชันจะให้ User กรอกข้อมูลที่ใช้ในการคำนวณ ซึ่งบางค่าได้จากการวัด เช่น ค่าของเลือดในปัจจุบัน



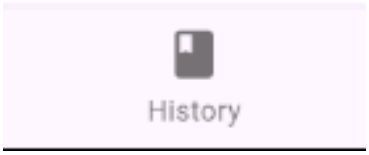


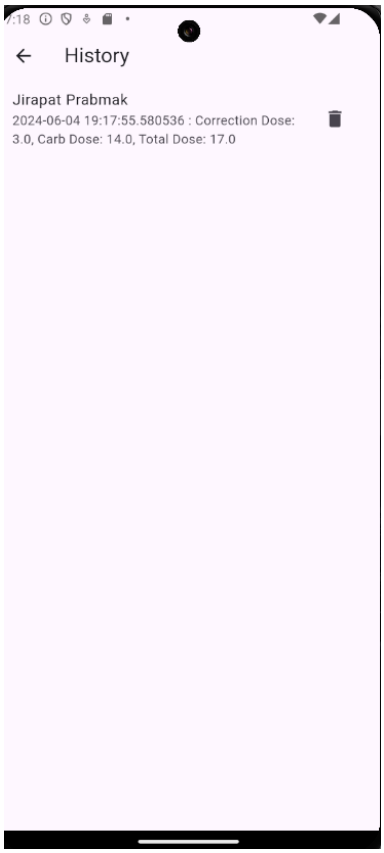
เมื่อกรอกข้อมูลจนเสร็จแล้วกดปุ่ม Calculate และจะได้ผลลัพธ์

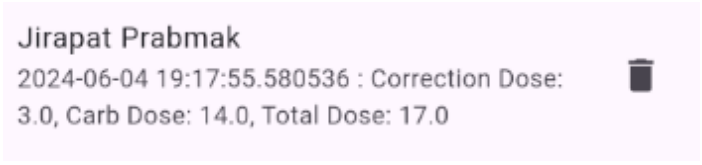




เมื่อ User ทราบค่าของยาที่ต้องฉีดและกดปุ่ม “Ok” แอปพลิเคชันจะกลับไปยังหน้าหลักของแอปพลิเคชัน ซึ่ง User สามารถตรวจสอบประวัติการคำนวนของตนเองได้ โดยการกดปุ่มที่คำว่า “History” แอปพลิเคชันจะแสดงประวัติการคำนวณย้อนหลังทั้งหมด ซึ่งรายละเอียดประวัติการคำนวณจะประกอบไปด้วย ชื่อ, วันเวลาที่คำนวณ, ค่าของ CorrectionDose, CarbDose, TotalDose







บรรณานุกรม

โรคเบาหวาน อาการ สาเหตุ และวิธีรักษา รู้ก่อนสายก็สุขภาพดีได้

โรคเบาหวาน อาการ สาเหตุ และวิธีรักษา รู้ก่อนสายก็สุขภาพดีได้ (dwpharma.co), 19 พฤษภาคม 2567.

“เบาหวาน” แต่ละชนิด ต่างกันอย่างไร?

“เบาหวาน” แต่ละชนิด ต่างกันอย่างไร? | โรงพยาบาลพญาไท (phyathai.com), 20 พฤษภาคม 2567.

โรคเบาหวาน (Diabetes)

โรคเบาหวาน (Diabetes) อาการ สาเหตุ การตรวจวินิจฉัยและการรักษา | MedPark Hospital, 20 พฤษภาคม 2567.

โรคเบาหวานอันตราย แตกต่างอย่างไรใน 4 ชนิด

โรคเบาหวานอันตราย แตกต่างอย่างไรใน 4 ชนิด | โรงพยาบาลพญาไท (phyathai.com), 21 พฤษภาคม 2567.

เข้าใจเบาหวาน

เข้าใจเบาหวาน (dmthai.org), 21 พฤษภาคม 2567.