6

Storage:
Todo list

Asst. Prof. Dr. Chanankorn Jandaeng

To-do List Application Using React Native

- To introduce the topic of the lesson/workshop.
- Inform learners that:
  - The application to be developed is a To-do List app.
  - The development framework used is React Native.
  - The app includes persistent local data storage using AsyncStorage.
  - It supports CRUD operations: Create, Read, Update, and Delete.

Data Structure

- **Array**
  - The array is the primary data structure used to store multiple to-do items in a sequential manner.
  - It is utilized within the component's state to manage and display the entire list, typically rendered using FlatList.

```
const [todos, setTodos] = useState([
  { id: '1', title: 'อ่านหนังสือ' },
  { id: '2', title: 'เขียนโค้ด' }
]);
```

  - To add a new item to the list:

```
const newTodo = { id: Date.now().toString(), title: text };
setTodos([...todos, newTodo]);
```

Data Structure

- **Object: Representing Each To-do Item**
  - Each individual to-do item is structured as an object, allowing multiple fields (e.g., id, title) to be encapsulated within a single entity.
  - The use of a unique **id** is essential for accurately deleting or updating specific items in the list.

```
{
  id: "1691203400871",
  title: "ไปตลาด"
}
```

Data Structure

- **Key–Value: Data Storage in AsyncStorage**
  - *AsyncStorage* operates based on a key–value paradigm, where both the key and value are strings.
  - Therefore, the array of to-do items (*todos[]*) must be converted to a string before storage.
  - Storage

```
await AsyncStorage.setItem('@todos', JSON.stringify(todos));
```

  - Load data:

```
const data = await AsyncStorage.getItem('@todos');
if (data) setTodos(JSON.parse(data));
```

Data Structure

- **JSON (JavaScript Object Notation)**
  - Since AsyncStorage does not directly support storing objects or arrays, JSON is used for serialization and deserialization:
  - *JSON.stringify()* converts an object or array into a string for storage.
  - *JSON.parse()* converts the string back into its original object or array form when retrieved.

```
const raw = JSON.stringify([{ id: '1', title: 'นอน' }]); // Save
const todos = JSON.parse(raw); // Load
```

CRUD the Data Storage

```
// 1. เพิ่มรายการใหม่
setTodos([...todos, { id: Date.now().toString(), title: text }]);

// 2. ลบรายการ
setTodos(todos.filter(item => item.id !== id));

// 3. บันทึกลงเครื่อง
await AsyncStorage.setItem('@todos', JSON.stringify(todos));
```

CRUD: Create

- **Create: Adding a New Item**
  - Allow users to add a to-do item using a button, based on input from a TextInput.
- **Example Code:**

```
const handleAdd = () => {
  const newTodo = { id: Date.now().toString(), title: text };
  const updated = [...todos, newTodo];
  setTodos(updated);
  AsyncStorage.setItem('@todos', JSON.stringify(updated));
  setText('');
};
```

- Summary:
  - A new object is created and added to the array.
  - The state is updated and saved to AsyncStorage.
  - Date.now() is used to generate a unique id.

CRUD: Read

- **Read: Loading Items from AsyncStorage**
  - Load to-do items from local storage (AsyncStorage) when the app launches for the first time.
  - **Example Code:**

```
useEffect(() => {
  loadTodos();
}, []);

const loadTodos = async () => {
  const json = await AsyncStorage.getItem('@todos');
  if (json) setTodos(JSON.parse(json));
};
```

  - **Summary:**
    - useEffect() ensures the data is loaded once when the app starts.
    - The loaded data is stored in the todos state and rendered using FlatList.

CRUD: Update

- **Update: Editing an Item**
  - Users tap "Edit" to load a to-do into a TextInput, modify the text, then confirm with an "Update" action.
  - **Example Code:**

```
const handleUpdate = () => {
  const updated = [...todos];
  updated[editingIndex].title = text;
  setTodos(updated);
  AsyncStorage.setItem('@todos', JSON.stringify(updated));
  setEditingIndex(null);
  setText('');
};
```

  - **Summary:**
    - The item is accessed via its index, modified in-place.
    - The updated array is saved back to AsyncStorage.
    - Clears editing state and input after update.

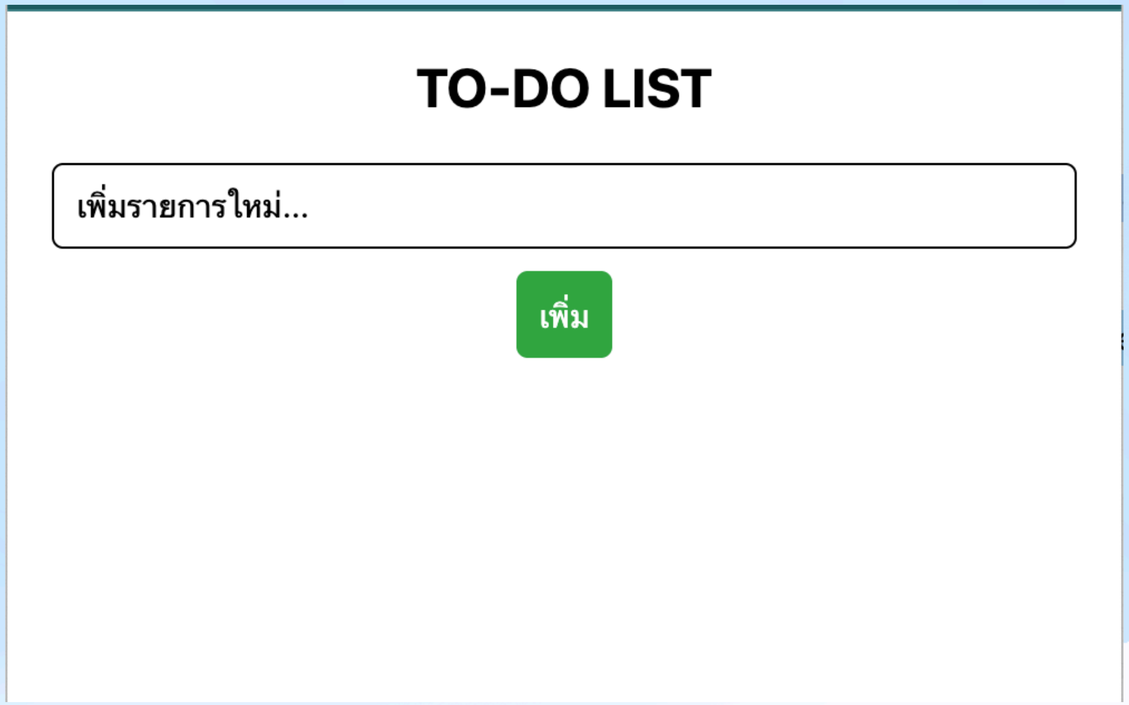
CRUD: Delete

- **Delete: Removing an Item**
  - Concept:
    - Use .filter() to remove a specific item by its id.
  - **Example Code:**

```
const handleDelete = (id) => {
  const updated = todos.filter(item => item.id !== id);
  setTodos(updated);
  AsyncStorage.setItem('@todos', JSON.stringify(updated));
};
```

  - **Summary:**
    - A new array excluding the specified id is created.
    - The state and stored data are both updated accordingly.

Create Project



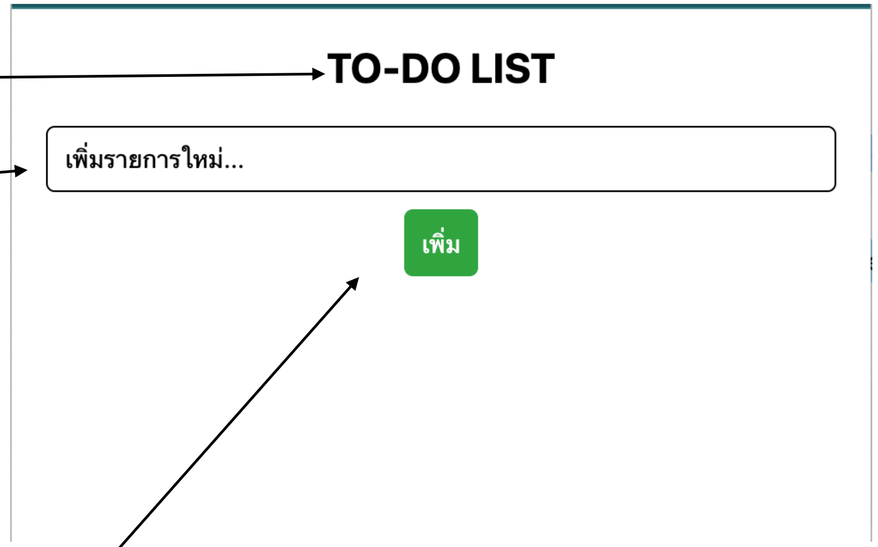
- Install Expo CLI (if not already installed)
  - *npm install -g expo-cli*
- Create a New Project
  - *expo init TodoApp*
- Select Template:

  Choose "blank (JavaScript)" when prompted.
- Install AsyncStorage Package
  - *npx expo install @react-native-async-storage/async-storage*
- Run the App

  Use either of the following commands:
  - *npm start*
  - or
  - *expo start*

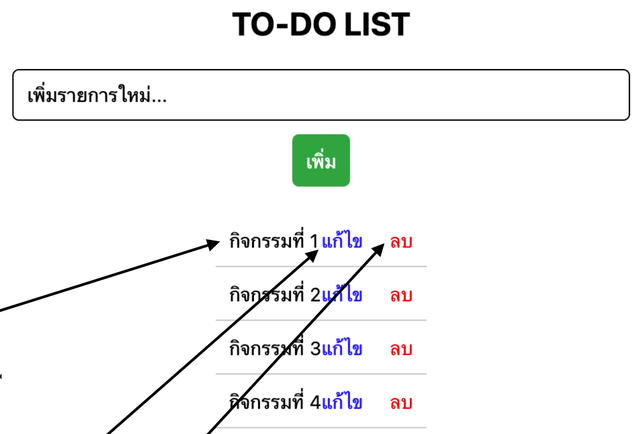
Create UI and its stylesheet

```
return (
  <View style={styles.container}>
    <Text style={styles.title}>TO-DO LIST</Text>

    <TextInput
      style={styles.input}
      value={text}
      onChangeText={setText}
      placeholder="เพิ่มรายการใหม่..."
    />

    {editingIndex !== null ? (
      <TouchableOpacity style={styles.button} onPress={handleUpdate}>
        <Text style={styles.buttonText}>อัปเดต</Text>
      </TouchableOpacity>
    ) : (
      <TouchableOpacity style={styles.button} onPress={handleAdd}>
        <Text style={styles.buttonText}>เพิ่ม</Text>
      </TouchableOpacity>
    )}
```



Create UI and its stylesheet

```
<FlatList
  data={todos}
  keyExtractor={(item) => item.id}
  renderItem={({ item, index }) => (
    <View style={styles.todoItem}>
      <Text>{item.title} </Text>
      <View style={styles.buttons}>
        <TouchableOpacity onPress={() => handleEdit(item, index)}>
          <Text style={styles.edit}>แก้ไข</Text>
        </TouchableOpacity>
        <TouchableOpacity onPress={() => handleDelete(item.id)}>
          <Text style={styles.delete}>ลบ</Text>
        </TouchableOpacity>
      </View>
    </View>
  )}
/>
```



Add Activity

```
<TouchableOpacity style={styles.button} onPress={handleAdd}>
  <Text style={styles.buttonText}>เพิ่ม</Text>
</TouchableOpacity>
```

```
import React, { useState, useEffect } from 'react';
import { View, Text, TextInput, TouchableOpacity, FlatList, StyleSheet, Alert } from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';

export default function App() {
  const [todos, setTodos] = useState([]);
  const [text, setText] = useState('');
  const [editingIndex, setEditingIndex] = useState(null);
```

```
<TextInput
  style={styles.input}
  value={text}
  onChangeText={setText}
  placeholder="เพิ่มรายการใหม่..."
/>
```

```
const handleAdd = () => {
  if (text.trim() === '') {
    Alert.alert('กรุณาป้อนรายการก่อน');
    return;
  }
  const newItem = { id: Date.now().toString(), title: text };
  const updated = [...todos, newItem];
  setTodos(updated);
  saveTodos(updated);
  setText('');
};
```

SaveTodo

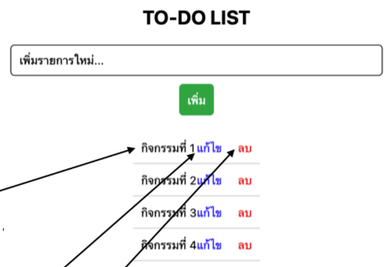
```
const saveTodos = async (newTodos) => {
  try {
    await AsyncStorage.setItem('@todos', JSON.stringify(newTodos));
  } catch (e) {
    console.error('Failed to save todos.');
  }
};
```

Read - AutoReload()

```
useEffect(() => {
  loadTodos();
}, []);
```

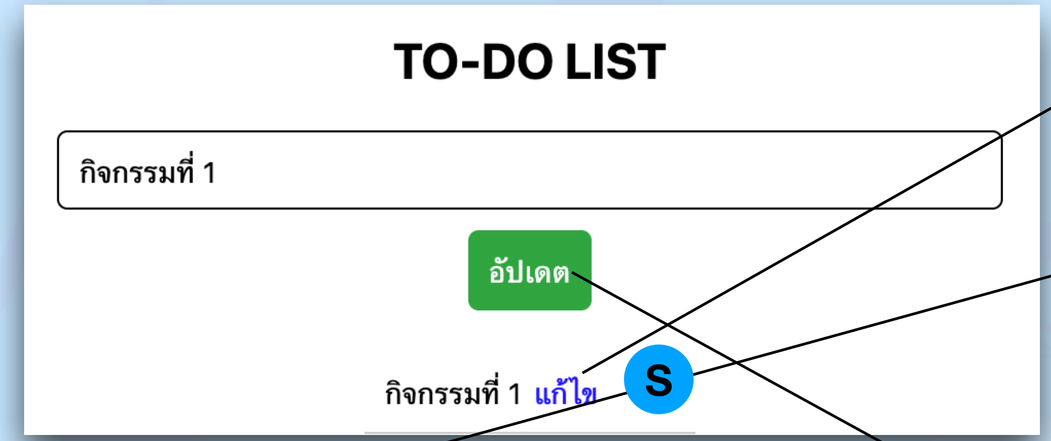
```
const loadTodos = async () => {
  try {
    const data = await AsyncStorage.getItem('@todos');
    if (data !== null) {
      setTodos(JSON.parse(data));
    }
  } catch (e) {
    console.error('Failed to load todos.');
  }
};
```

```
<FlatList
  data={todos}
  keyExtractor={(item) => item.id}
  renderItem={({ item, index }) => (
    <View style={styles.todoItem}>
      <Text>{item.title} </Text>
      <View style={styles.buttons}>
        <TouchableOpacity onPress={() => handleEdit(item, index)}>
          <Text style={styles.edit}>แก้ไข</Text>
```



Update

```
keyExtractor={(item) => item.id}
renderItem={({ item, index }) => (
  <View style={styles.todoItem}>
    <Text>{item.title} </Text>
    <View style={styles.buttons}>
      <TouchableOpacity onPress={() => handleEdit(item, index)}>
        <Text style={styles.edit}>แก้ไข</Text>
      </TouchableOpacity>
```



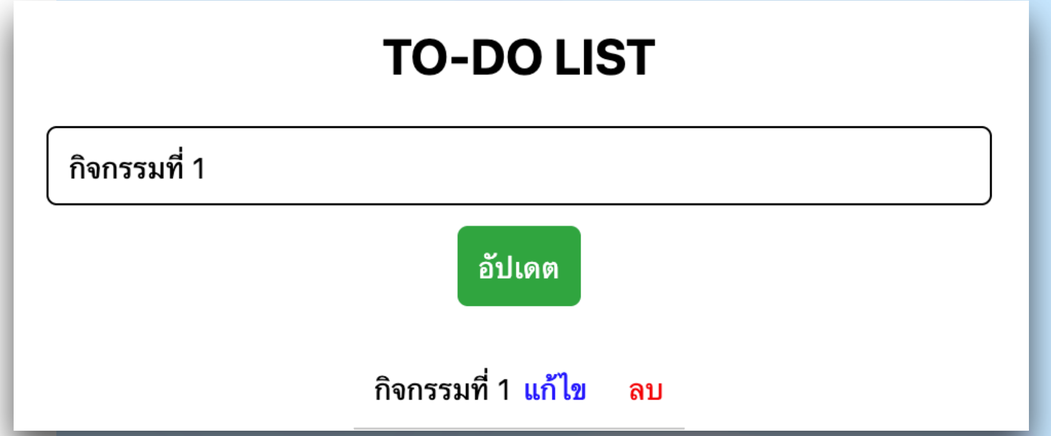

```
const handleEdit = (item, index) => {
  setText(item.title);
  setEditingIndex(index);
};
```

```
{editingIndex !== null ? (
  <TouchableOpacity style={styles.button} onPress={handleUpdate}>
    <Text style={styles.buttonText}>อัปเดต</Text>
  </TouchableOpacity>
) : (
  <TouchableOpacity style={styles.button}
    <Text style={styles.buttonText}>เพิ่ม</
  </TouchableOpacity>
)}
```

```
const handleUpdate = () => {
  const updated = [...todos];
  updated[editingIndex].title = text;
  setTodos(updated);
  saveTodos(updated);
  setEditingIndex(null);
  setText('');
};
```

Delete

```
<FlatList
  data={todos}
  keyExtractor={(item) => item.id}
  renderItem={({ item, index }) => (
    <View style={styles.todoItem}>
      <Text>{item.title} </Text>
      <View style={styles.buttons}>
        <TouchableOpacity onPress={() => handleEdit(item, index)}>
          <Text style={styles.edit}>แก้ไข</Text>
        </TouchableOpacity>
        <TouchableOpacity onPress={() => handleDelete(item.id)}>
          <Text style={styles.delete}>ลบ</Text>
        </TouchableOpacity>
      </View>
    </View>
  )}
/>
```




```
const handleDelete = (id) => {
  const updated = todos.filter(item => item.id !== id);
  setTodos(updated);
  saveTodos(updated);
};
```

Style Sheet

```
const styles = StyleSheet.create({
  container: {
    flex: 1, justifyContent: 'center', alignItems: 'center', padding: 20,
  },
  title: {
    fontSize: 24, fontWeight: 'bold', marginBottom: 20,
  },
  input: {
    width: '100%', borderWidth: 1, padding: 10, marginBottom: 10, borderRadius: 5,
  },
  button: {
    backgroundColor: '#28a745', padding: 10, borderRadius: 5, marginBottom: 20,
  },
  buttonText: {
    color: 'white',
  },
  todoItem: {
    width: '100%', padding: 10, borderBottomWidth: 1, borderColor: '#ccc',
    flexDirection: 'row', justifyContent: 'space-between', alignItems: 'center',
  },
  buttons: {
    flexDirection: 'row', gap: 10,
  },
  edit: {
    color: 'blue', marginRight: 10,
  },
  delete: {
    color: 'red',
  },
});
```

Practice

```
return (
  <View style={styles.container}>
    <Text style={styles.heading}>📒 Notes Keeper</Text>

    <TextInput
      style={styles.input}
      value={title}
      onChangeText={setTitle}
      placeholder="Enter note title..."
    />

    <View style={styles.pickerContainer}>
      <Text style={styles.label}>Category:</Text>
      <Picker
        selectedValue={category}
        style={styles.picker}
        onValueChange={(itemValue) => setCategory(itemValue)}
      >
        <Picker.Item label="Personal" value="Personal" />
        <Picker.Item label="Work" value="Work" />
        <Picker.Item label="Study" value="Study" />
      </Picker>
    </View>

    <TouchableOpacity style={styles.button} onPress={handleAddNote}>
      <Text style={styles.buttonText}>Add Note</Text>
    </TouchableOpacity>
```