Mobile Application Development

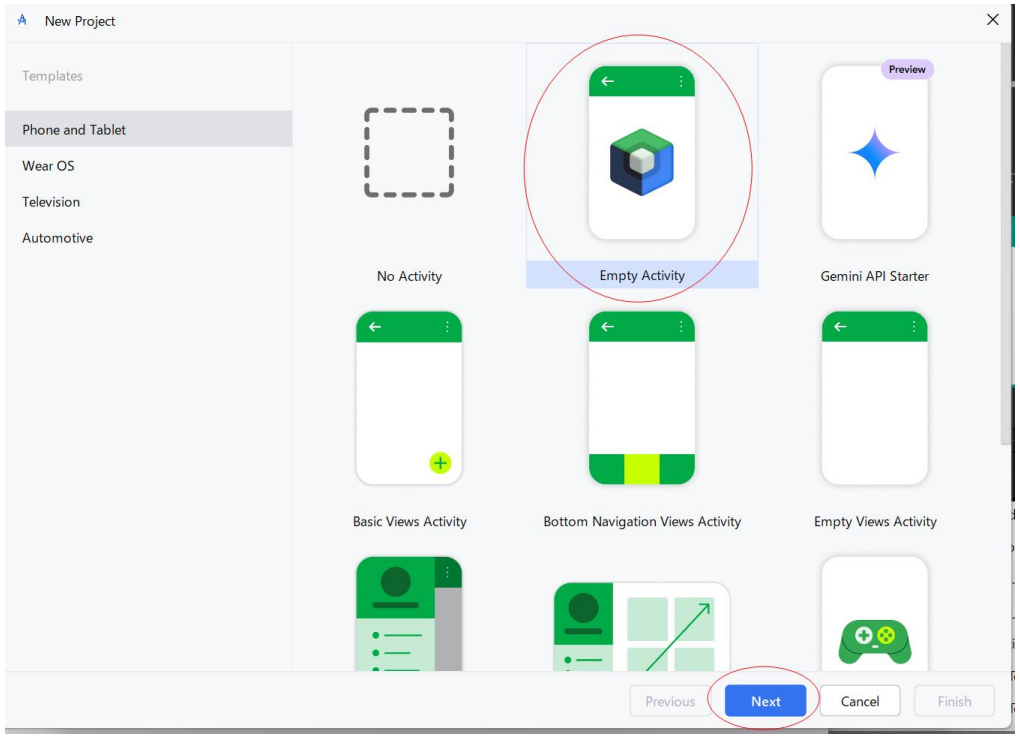
Week5　Advanced UI Design

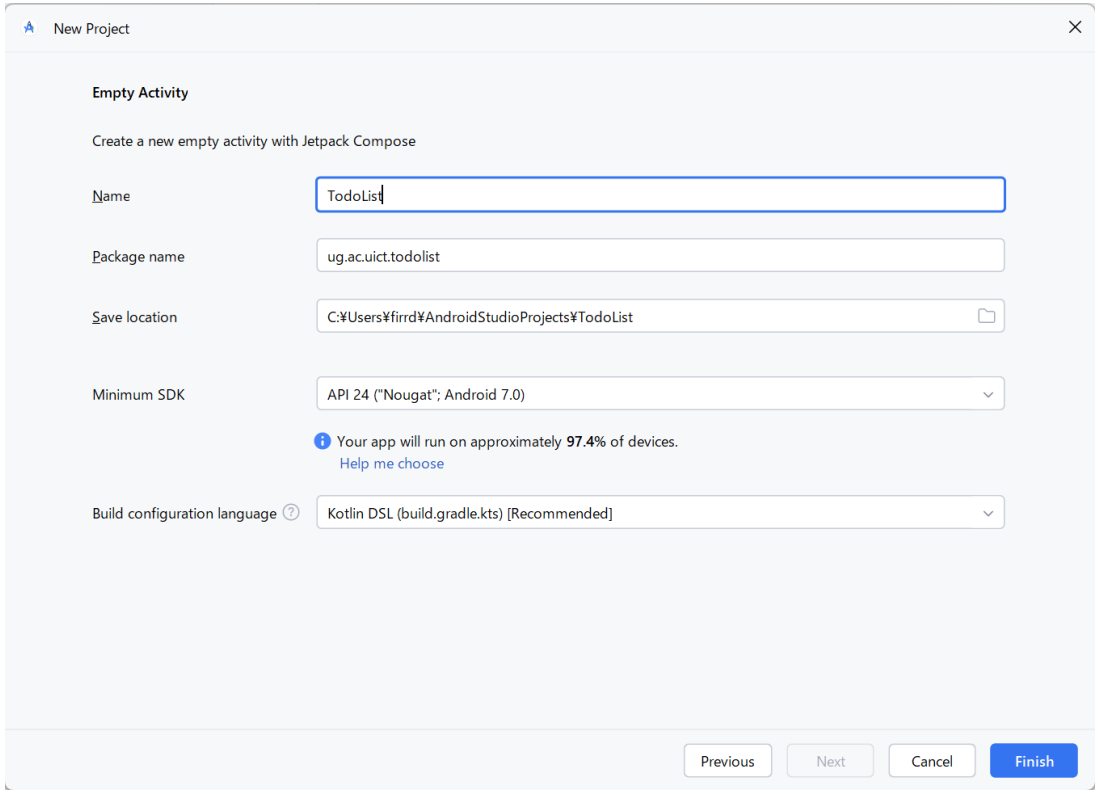
Yi SUN
Kobe Institute of Computing

Design a To-do List App

Start a new Project

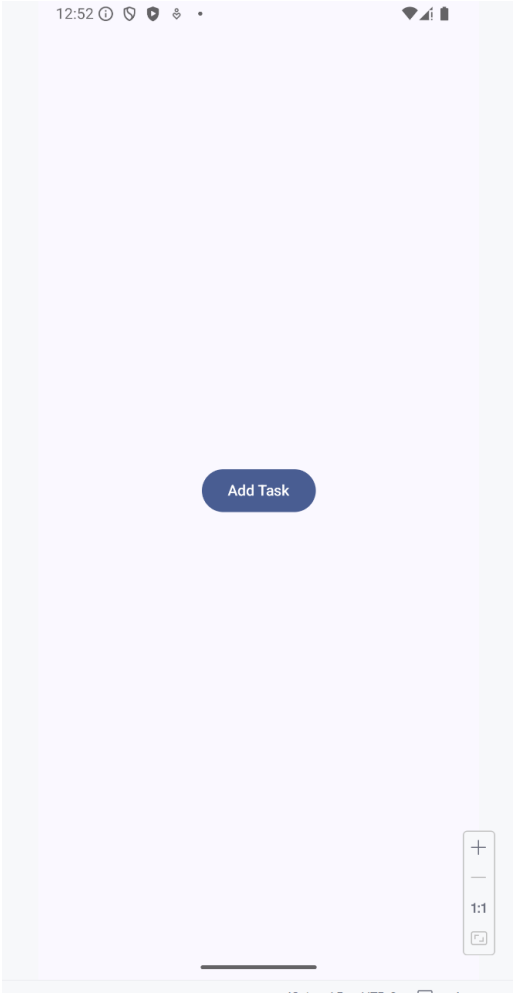
1. Open AndroidStudio
2. Select `File > New > NewProject`
3. Choose `Empty Activity`
4. Name: TodoList





Add a Button to the screen

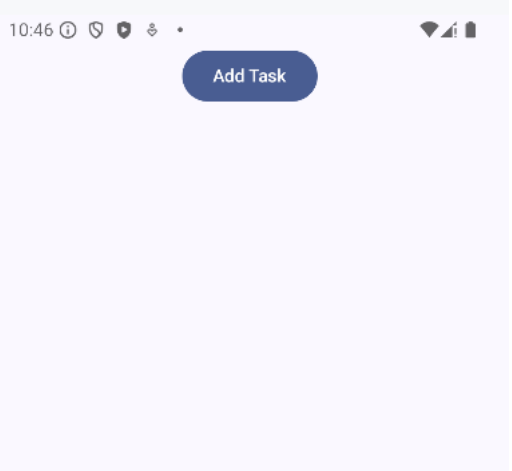
```
Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
    Column (
        modifier = Modifier.fillMaxSize().padding(innerPadding),
        verticalArrangement = Arrangement.Center
    ){
        Button(
            onClick = {},
            modifier = Modifier.align(Alignment.CenterHorizontally)
        ) {
            Text("Add Task")
        }
    }
}
```



Adding the LazyColumn and the Data Class

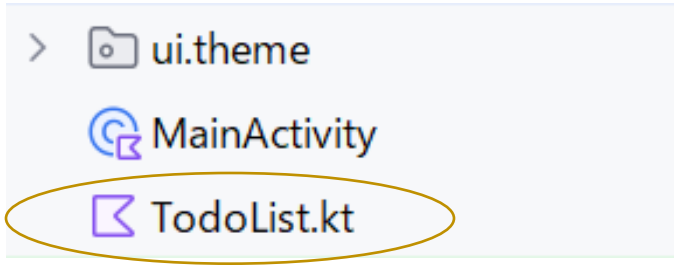
```
Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
    var todoItems by remember { mutableStateOf(listOf<TodoItem>()) }
    Column (
        modifier = Modifier.fillMaxSize().padding(innerPadding),
        verticalArrangement = Arrangement.Center
    ){
        Button(
            onClick = {},
            modifier = Modifier.align(Alignment.CenterHorizontally)
        ) {
            Text("Add Task")
        }
        LazyColumn(
            modifier = Modifier.fillMaxSize().padding(16.dp)
        ) {
            items(todoItems){

            }
```

```
data class TodoItem(
    val id: Int,
    var name: String,
    var deadline: String,
    var isEditing: Boolean = false
)
```



Move the TodoItem class to individual file

Make New file “TodoList.kt” and move the data class TodoItem to the new file.
and move the main screen design to new compose.




```
data class TodoItem(
    val id: Int,
    var name: String,
    var deadline: String,
    var isEditing: Boolean = false
)
```

```
@Composable
fun TodoListApp(modifier: Modifier = Modifier){
    var todoItems by remember { mutableStateOf(listOf<TodoItem>()) }
    Column (
        modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Center
    ){
        Button(
            onClick = {},
            modifier = Modifier.align(Alignment.CenterHorizontally)
        ) {
            Text("Add Task")
        }
        LazyColumn(
            modifier = Modifier.fillMaxSize().padding(16.dp)
        ) {
            items(todoItems){}
```

Setup a AlertDialog

```
fun TodoListApp(modifier: Modifier = Modifier){
    var todoItems by remember { mutableStateOf(listOf<TodoItem>()) }
    var showDialog by remember { mutableStateOf(false)}
    Column (
        modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Center
    ){
        Button(
            onClick = { showDialog = true },
            modifier = Modifier.align(Alignment.CenterHorizontally)
        ) {
            Text("Add Task")
        }
```

Add a new var to remember the button status

When click the button the status become true

```
if(showDialog){
    AlertDialog(onDismissRequest = {showDialog=false}){
        Text("This is a Alert Dialog")
    }
}
```

Add the BasicAlertDialog function

Modify the AlertDialog to Form

```
var itemName by remember { mutableStateOf("")}
var itemDeadline by remember { mutableStateOf("")}
```

Add two var

```
if(showDialog){
    AlertDialog(
        onDismissRequest = {showDialog=false},
        confirmButton = {},
        title = { Text("Add Todo Item")},
        text = {
            Column {
                OutlinedTextField(
                    value = itemName,
                    onValueChange = {itemName = it},
                    singleLine = true,
                    modifier = Modifier.fillMaxWidth().padding(8.dp)
                )
                OutlinedTextField(
                    value = itemDeadline,
                    onValueChange = {itemDeadline = it},
                    singleLine = true,
                    modifier = Modifier.fillMaxWidth().padding(8.dp)
                )
            }
```

Change the AlertDialog to Textfield

set up the confirmButton

```
onDismissRequest = {showDialog=false},
confirmButton = {
    Row(
        modifier = Modifier.fillMaxWidth().padding(8.dp),
        horizontalArrangement = Arrangement.SpaceBetween
    ){
        Button(onClick = {
            if(itemName.isNotBlank()){
                val newItem = TodoItem(
                    id = todoItems.size+1,
                    name = itemName,
                    deadline = itemDeadline
                )
                todoItems = todoItems + newItem
                showDialog = false
                itemName = "“
                itemDeadline = ""
            }
        }) {
            Text("Add")
        }
        Button(onClick = { showDialog = false}){
            Text("Cancel")
        }
    }
},
```

Design a new compose to show the Todo item

```
@Composable
fun TodoListItem(
    item: TodoItem,
    onEditClick: () -> Unit,
    onDeleteClick: () -> Unit,
){
    Row(
        modifier = Modifier.padding(8.dp).fillMaxWidth()
    ){
        Text(text = item.name, modifier = Modifier.padding(8.dp))
    }
}
```

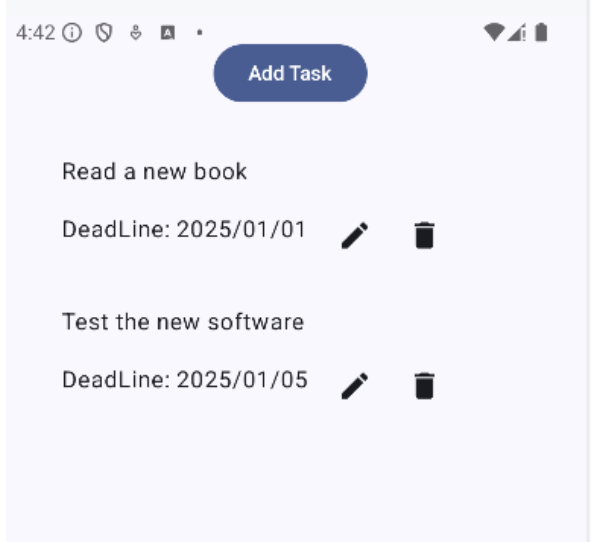
Add a new compose to show the Todo item

```
LazyColumn(
    modifier = Modifier.fillMaxSize().padding(16.dp)
) {
    items(todoItems){
        TodoListItem(it,{},{})
    }
}
```

Add the compose Lazy Column

Add the Button to the Todo item List

```
fun TodoListItem(
    item: TodoItem,
    onEditClick: () -> Unit,
    onDeleteClick: () -> Unit,
){
    Column(
        modifier = Modifier.padding(8.dp).fillMaxWidth()
    ){
        Text(text = item.name, modifier = Modifier.padding(8.dp))
        Row(modifier = Modifier.fillMaxWidth()){
            Text(text = "DeadLine: ${item.deadline}", modifier = Modifier.padding(8.dp))
            IconButton(onClick = onEditClick) {
                Icon(imageVector = Icons.Default.Edit, contentDescription = "edit")
            }
            IconButton(onClick = onDeleteClick) {
                Icon(imageVector = Icons.Default.Delete, contentDescription = "Delete")
            }
        }
    }
```



Add a new compose to edit the Todo Item

```
@Composable
fun TodoItemEditor(item: TodoItem,onEditComplete: (String, String) -> Unit){
    var editedName by remember { mutableStateOf(item.name) }
    var editedDeadline by remember { mutableStateOf(item.deadline) }
    var isEditing by remember { mutableStateOf(item.isEditing) }
    Row(
        modifier = Modifier.fillMaxWidth().background(Color.White).padding(8.dp),
        horizontalArrangement = Arrangement.SpaceEvenly
    ){
        Column {
            BasicTextField(
                value = editedName,
                onValueChange = {editedName = it},
                singleLine = true,
                modifier = Modifier.wrapContentSize().padding(8.dp)
            )
            BasicTextField(
                value = editedDeadline,
                onValueChange = {editedDeadline = it},
                singleLine = true,
                modifier = Modifier.wrapContentSize().padding(8.dp)
            )
        }
        Button(
            onClick = {
                isEditing = false
                onEditComplete(editedName,editedDeadline)
        }
        ){
            Text("Save")
        }
    }
}
```

A callback function that is triggered when editing is finished. It takes two strings—the edited name and deadline—as parameters.

The first BasicTextField displays and allows editing of the editedName. It is set to single line and padded.The second BasicTextField similarly handles the editedDeadline value.

Finalizing the TodoList App

```
LazyColumn(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp)
) {
    items(todoItems){
        item ->
        if(item.isEditing) {
            TodoItemEditor(item = item, onEditComplete = { editedName, editedDeadline ->
                // Step 1: Reset editing mode for all items
                todoItems = todoItems.map { it.copy(isEditing = false) }
                // Step 2: Locate the specific item being edited by matching its id
                val editedItem = todoItems.find { it.id == item.id }
                // Step 3: If the item exists, update its name and deadline with the new values
                editedItem?.let {
                    it.name = editedName
                    it.deadline = editedDeadline
                }
            })
        } else {
            TodoListItem(item = item, onEditClick = {
                // When edit is requested, mark this item as editing and others as not editing
                todoItems = todoItems.map { it.copy(isEditing = it.id == item.id) }
            }, onDeleteClick = {
                // Remove the item from the list when delete is clicked
                todoItems = todoItems - item
            })
```

Map method

```
fun main() {
    val numbers = listOf(1,2,3)
    val doubled = numbers.map{it * 2}
    println(doubled)
}
```

Result

```
[2, 4, 6]
```

You can test the source code from Kotlin Playground https://play.kotlinlang.org/

Copy method

```
fun main() {
    val toyota1 = Car(make = "Toyota", color="Red")
    val toyota2 = toyota1.copy(color="White")
    println(toyota1)
    println(toyota2)
}

data class Car(val make:String, val color:String)
```

Result

```
Car(make=Toyota, color=Red)
Car(make=Toyota, color=White)
```

Use let function to safely transform nullable string

```
fun main() {
    val name:String? = "Mario"
    println(name.uppercase())
}
```

The error message

Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?

```
fun main() {
    val name:String? = "Mario"
    name?.let{
        println(name.uppercase())
    }
}
```

Result

```
MARIO
```