### Design Pattern Refactor 1: Factory Method (time\_entry.dart)

TimeEntry createTimeEntry(Map<String, dynamic> data, String id) {

return TimeEntry(

id: id, // Pass Firestore document ID here

date: data['date'], // Direct access without type safety

from: data['from'],

to: data['to'],

task: data['task'],

tag: data['tag'],

);

}

### Refactored Code

factory TimeEntry.fromMap(Map<String, dynamic> data, String id) {

return TimeEntry(

id: id, // Pass Firestore document ID here

date: data['date'] as String,

from: data['from'] as String,

to: data['to'] as String,

task: data['task'] as String,

tag: data['tag'] as String,

);

}

**Explanation of Refactor:**

Smell: The old code does not use the Factory Method pattern, leading to scattered and inconsistent object creation logic across the application.

Solution: The Factory Method pattern centralizes object creation by encapsulating the logic within TimeEntry.fromMap. This makes sure the type safety using explicit casting (as String).

### Design Pattern Refactor 2: Builder Pattern (record\_time\_screen.dart)

final id = Uuid().v4();

final date = \_dateController.text.isEmpty ? "N/A" : \_dateController.text;

final from = \_fromController.text.isEmpty ? "N/A" : \_fromController.text;

final to = \_toController.text.isEmpty ? "N/A" : \_toController.text;

final taskName = \_taskController.text.isEmpty ? "Untitled" : \_taskController.text;

final tag = \_tagController.text.isEmpty ? "General" : \_tagController.text;

final task = TimeEntry(

id: id,

date: date,

from: from,

to: to,

task: taskName,

tag: tag,

);

### Refactored Code

final task = TimeEntry(

id: Uuid().v4(),

date: \_dateController.text,

from: \_fromController.text,

to: \_toController.text,

task: \_taskController.text,

tag: \_tagController.text,

);

**Explanation of Refactor:**

Smell: Repeated code for handling default values clutters the code and introduces potential inconsistencies.

Solution: Centralized field initialization in the timeentry constructor to streamline object creation.

### Design Pattern Refactor 3: Command Pattern (record\_time\_screen.dart)

void \_saveTask() async {

if (\_formKey.currentState!.validate()) {

final task = TimeEntry(

id: Uuid().v4(),

date: \_dateController.text,

from: \_fromController.text,

to: \_toController.text,

task: \_taskController.text,

tag: \_tagController.text,

);

await FirebaseFirestore.instance.collection('time\_entries').add(task.toMap())

.then((\_) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text('Task added successfully!')),

);

})

.catchError((error) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text('Error saving task: $error')),

);

});

}

}

### Refactored Code

void \_saveTask() async {

if (\_formKey.currentState!.validate()) {

final task = TimeEntry(

id: Uuid().v4(),

date: \_dateController.text,

from: \_fromController.text,

to: \_toController.text,

task: \_taskController.text,

tag: \_tagController.text,

);

await \_firebaseService.addTask(task);

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text('Task added successfully!')),

);

}

}

**Explanation of Refactor:**

Smell: Direct firestore interaction connects the UI layer to the backend, complicating maintenance and testing.

Solution: Abstracted Firestore logic into \_firebaseService.addTask(task) to get rid of any possible issues and align more with the Command Pattern.

### Design Pattern Refactor 4**:** Observer Pattern(record\_time\_screen.dart)

void \_updatePreviewManually() {

\_previewDate = \_dateController.text.isNotEmpty ? \_dateController.text : "YYYY/MM/DD";

\_previewFrom = \_fromController.text.isNotEmpty ? \_fromController.text : "HH:MM AM/PM";

\_previewTo = \_toController.text.isNotEmpty ? \_toController.text : "HH:MM AM/PM";

\_previewTask = \_taskController.text.isNotEmpty ? \_taskController.text : "Sample Task";

\_previewTag = \_tagController.text.isNotEmpty ? \_tagController.text : "Sample Tag";

// Manually call setState each time after updating preview variables

setState(() {});

}

### Refactored Code

void \_updatePreview() {

setState(() {

\_previewDate = \_dateController.text.isNotEmpty ? \_dateController.text : "YYYY/MM/DD";

\_previewFrom = \_fromController.text.isNotEmpty ? \_fromController.text : "HH:MM AM/PM";

\_previewTo = \_toController.text.isNotEmpty ? \_toController.text : "HH:MM AM/PM";

\_previewTask = \_taskController.text.isNotEmpty ? \_taskController.text : "Sample Task";

\_previewTag = \_tagController.text.isNotEmpty ? \_tagController.text : "Sample Tag";

});

}

**Explanation of Refactor:**

Smell: Manual updates require setState call after updating each field, which makes the code repetitive.

Solution: Encapsulated preview within a single setState block so that there are consistent UI notifications using the Observer Pattern.