**Sprint 1: Foundation & Core UI**

**Sprint Goal:** Create the fundamental structure of the app and a basic, view-only timetable.

* **Key User Stories:**
  + As a student, I want to view my full weekly timetable in the main app.
  + As a student, I want to persist my timetable data even if I close the app.
* **Tasks:**
  + Set up the Android Studio project with the correct dependencies for Room Database and LiveData.
  + Define the TimetableEntry data model (POJO).
  + Create the TimetableDao interface for database operations.
  + Implement the TimetableDatabase using Room.
  + Design and implement the activity\_main.xml layout with a TabLayout for days of the week and a RecyclerView for class entries.
  + Create the MainActivity.java to display static timetable data from the database.
  + Create a clean, reusable XML layout for a single class card item.
  + Implement the TimetableRepository to provide data to the ViewModel.
  + Create the TimetableViewModel to hold and manage UI-related data.

**Sprint 2: App Widget & Basic CRUD**

**Sprint Goal:** Implement the core App Widget functionality and enable users to add, edit, and delete classes.

* **Key User Stories:**
  + As a student, I want to add a new class to my timetable.
  + As a student, I want to edit an existing class entry.
  + As a student, I want to delete a class from my timetable.
  + As a student, I want to add a widget to my home screen.
  + As a student, I want the widget to automatically display my classes for the current day.
* **Tasks:**
  + Design and implement the fragment\_add\_class.xml layout for the add/edit form.
  + Add the necessary Java code in the ViewModel to handle adding, editing, and deleting classes.
  + Add a Floating Action Button (FAB) to the MainActivity to open the add/edit form.
  + Create the TimetableWidgetProvider broadcast receiver.
  + Create the TimetableWidgetFactory to populate the widget's list view with data from the TimetableRepository.
  + Configure the app's manifest file to support the widget.
  + Test the end-to-end data flow: adding a class in the app and seeing it reflected in the widget.

**Sprint 3: Day Order & Refinements**

**Sprint Goal:** Introduce the day order feature and finalize core logic.

* **Key User Stories:**
  + As a student, I want to switch between a standard weekly view and a day order view.
  + As a student, I want to assign a day of the week or a day order number to each class.
  + As a student, I want to set the current day order for the week.
* **Tasks:**
  + Update the TimetableEntry data model to include a field for "day order."
  + Modify the MainActivity's TabLayout logic to toggle between day-of-the-week and day-order based data.
  + Implement the "Day Order Settings Screen" with a dropdown to select the current day order.
  + Update the TimetableRepository and TimetableViewModel to correctly query the database based on the selected view.
  + Modify the widget's logic to display the classes for the current day based on the saved day order setting.
  + Add basic animations and transitions to the UI for a smoother user experience.

**Sprint 4: Polish, Testing & Deployment**

**Sprint Goal:** Prepare the app for release by fixing bugs, improving the user experience, and completing final checks.

* **Key User Stories:**
  + As a student, I want the app to be fast and bug-free.
  + As a student, I want the UI to be easy to use and visually appealing.
* **Tasks:**
  + Conduct a full round of unit and integration testing.
  + Perform user acceptance testing (UAT) with a few people to get feedback.
  + Refine the app's icons and branding.
  + Write a brief, helpful welcome screen or tutorial.
  + Optimize app performance, including memory usage and loading times.
  + Prepare the app for release on the Google Play Store by setting up release builds and signing.
* **DETAILS:**

### 1. Visual Polish and Interactivity

The app is currently very functional, but we can make it more visually appealing and responsive.

* **Implement UI Animations:** Add subtle animations to make the user interface feel more alive. For example, a fade-in or slide-in animation when a new fragment loads, or a slight scale effect on buttons when they are tapped. The TimetableListFragment layout has a RecyclerView that could benefit from animations for adding or removing items.
* **Refine the Color Palette:** The current design uses standard Material Components colors. We can update your colors.xml and themes.xml to use a custom color palette that is both visually appealing and unique to your app's brand.
* **Improve List Item Design:** The timetable\_entry\_item.xml can be enhanced with more modern design elements. Adding rounded corners, shadows, and clear visual separation between items will make the timetable easier to read at a glance.