# MVP-Checkliste: In 3 Tagen zum Flutter-Stacked-MVP

Diese Checkliste führt dich Schritt für Schritt durch den Bau eines Minimum Viable Products (MVP) mit Flutter und dem Stacked-Framework. Sie ist so aufgebaut, dass du am Freitag startest und am Montag bereits eine lauffähige Demo vorzeigen kannst. Halte dich an das Prinzip **time-boxed** – klare Prioritäten statt Perfektion.

## 1. Scope Cut & Erfolgskriterien

Bevor du mit dem Coden beginnst, musst du den Umfang präzise festlegen. Nutze die MosCoW-Methode (Must/Should/Won't), um deine Ideen zu ordnen. Das verhindert, dass du dich verzettelst.

- **Must-Haves:** Funktionen, ohne die die App nicht ihre Kernidee erfüllt (z. B. Login und eine einzige Kernfunktion).
- **Should-Haves:** Verbesserungen, die den MVP abrunden, aber im Zweifel weggelassen werden können.
- **Won't-Haves:** Ideen, die ausdrücklich auf später verschoben werden; sie gehören nicht in den 3-Tage-MVP.
- **Erfolgskriterien definieren:** Was bedeutet „Demo fertig"? Lege messbare Ziele fest, z. B. „Der Nutzer kann ein Objekt anlegen und speichern". Das erleichtert die Entscheidung, wann der MVP abgeschlossen ist.

## 2. Tech-Setup (Stacked, Services, Routing, DI)

Ein sauberes Setup spart Zeit und Nerven. Das Stacked-Framework basiert auf dem MVVM-Prinzip und bringt nützliche Helfer für Navigation, Service-Locator und Code-Generierung mit. Richte am ersten Tag dein Projekt wie folgt ein:

1. **Neues Flutter-Projekt erstellen.**
2. **Abhängigkeiten einrichten:** Füge `stacked`, `stacked_services` und (für die Code-Generierung) `auto_route`, `injectable` hinzu.
3. **Projektstruktur anlegen:** Ordner für `views`, `viewmodels`, `services` und einen app-Ordner für Routing und Dependency Injection.
4. **Routing und DI konfigurieren:** Verwende `NavigationService` und registriere deine Services im Service Locator (`get_it`). Nutze Codegen-Tools, um Boilerplate zu sparen.

## 3. Tag 1 – Domain & Screens (Mock-Data first)

Ziel des ersten Tages ist, eine klickbare Oberfläche mit Dummy-Daten zu erzeugen, damit du den Nutzerfluss testen kannst. Arbeite screen-basiert und konzentriere dich nur auf die Must-Haves.

1. **Domain-Modelle definieren:** Lege einfache Dart-Modelle an (z. B. `Task`, `User`), die später die echten Daten repräsentieren.
2. **Screens bauen:** Erstelle Views und ViewModels für deine Kernseiten (Start-Screen, Hauptfunktion etc.).
3. **Mock-Daten verwenden:** Fülle die ViewModels zunächst mit statischen oder Fake-Daten, um die UI zum Laufen zu bringen.
4. **Navigation einrichten:** Nutze `NavigationService` oder `auto_route`, um zwischen den Screens zu wechseln.
5. **State-Management:** Verwende `BaseViewModel` und `rebuildUi()` für UI-Updates, damit Änderungen an den Daten sofort in der Oberfläche sichtbar werden.

## 4. Tag 2 – Persistenz / Backend

Am zweiten Tag ersetzt du die Mock-Daten durch echte Datenquellen und sorgst für Robustheit.

1. **Backend auswählen und anbinden:** Entscheide dich für eine REST-API, GraphQL, Firebase usw. und implementiere einen Service, der echte Daten liefert.
2. **API-Calls implementieren:** Ersetze die Dummy-Services durch echte Methoden (`fetchData()`, `saveItem()`), die Daten abholen und speichern.
3. **Dependency Injection nutzen:** Registriere die neuen Services im Service Locator, damit deine ViewModels sie beziehen können.
4. **Ladezustände und Fehler abbilden:** Nutze `setBusy()` oder `FutureViewModel`, um Ladeindikatoren zu zeigen und Fehler sicher abzufangen.
5. **Eingaben validieren:** Stelle sicher, dass Benutzereingaben geprüft werden und unerwartete API-Antworten nicht zum Absturz führen.

## 5. Tag 3 – Polish, Analytics, Crashlytics, Testing

Der letzte Tag dient dem Feinschliff und der Vorbereitung auf den ersten Testlauf.

1. **UI-Politur:** Behebe Layout-Fehler, sorge für ein konsistentes Design und verbessere die Benutzerführung.
2. **Analytics integrieren:** Binde z. B. Firebase Analytics ein, um später Nutzungsdaten zu erfassen.
3. **Crashlytics einrichten:** Sammle Crash-Reports automatisch, um Fehler schnell zu finden.
4. **Interner Test:** Veröffentliche die App via TestFlight (iOS) oder internen Test-Track (Android) und hole dir Feedback aus dem Team.
5. **Bug-Fixing:** Behebe die wichtigsten Fehler, die während des Tests auftauchen.
6. **Validierung der KPIs:** Überprüfe, ob die App die definierten Erfolgskriterien erfüllt (z. B. Nutzer können eine Aufgabe erstellen ohne Absturz).

## 6. Post-MVP – Validierung & Roadmap

Nach der Demo ist vor der nächsten Iteration. Deine Aufgabe: validiere den Nutzen deiner Idee anhand von Daten und Feedback.

1. **KPIs analysieren:** Achte auf Kennzahlen wie Nutzerzahl, Session-Dauer, Retention und Crash-Rate.
2. **Feedback einholen:** Führe Gespräche mit Testern, um qualitative Rückmeldungen zu sammeln.
3. **Prioritäten neu setzen:** Entscheide basierend auf den Daten, welche Should- oder Won't-Features in der nächsten Version angegangen werden.
4. **Technische Schulden abbauen:** Plane Refactoring und Verbesserungen an Code und Architektur ein, bevor du neue Funktionen entwickelst.
5. **Iterieren:** Wiederhole den Prozess: scope definieren, entwickeln, messen – und nur das hinzufügen, was Nutzer wirklich brauchen.

---

Mit dieser Checkliste bist du bereit, innerhalb von drei Tagen eine funktionierende Flutter-App zu bauen, sie vorzuführen und anschließend strukturiert weiterzuentwickeln. Viel Erfolg bei deinem MVP-Sprint!

---