Byggplan för ZapLearn

# 0) Förutsättningar

• Node 20+, npm 10+  
• Git + GitHub/GitLab  
• Docker Desktop (för senare steg)

# 1) Initiera projekt

1. npm create vite@latest flashcards -- --template react-ts  
2. cd flashcards && git init && git add -A && git commit -m "chore: init"  
3. Lägg till grundläggande konfig (.editorconfig, .nvmrc, README.md)  
 - I tsconfig.json: aktivera "strict": true och sätt paths-alias @/\*

# 2) Installera beroenden

Kärna:  
npm i zustand localforage zod framer-motion react-hook-form @hookform/resolvers react-router-dom  
  
UI & styling:  
npm i -D tailwindcss postcss autoprefixer  
  
PWA, kvalitet & test:  
npm i -D vite-plugin-pwa eslint @typescript-eslint/eslint-plugin @typescript-eslint/parser eslint-config-prettier vitest jsdom @testing-library/react @testing-library/user-event @types/node  
  
Initiera Tailwind: npx tailwindcss init -p (koppla in index.css).

# 3) Lägg in shadcn/ui

1. npx shadcn-ui@latest init  
2. Lägg till komponenter du behöver direkt:  
 npx shadcn-ui@latest add button card input textarea dialog sheet select table progress badge toast  
3. Uppdatera tailwind.config.ts → content inkluderar ./src och ./components.  
4. Lägg till dark mode-toggle (finns exempel i shadcn/ui docs).

# 4) Mappstruktur

src/  
 app/ # layout, router, providers  
 components/ # UI-komponenter (shadcn) + egna  
 features/  
 decks/ # deckRepo (IndexedDB), deckService, deckStore (Zustand)  
 train/ # repetitionlogik, vyer  
 lib/ # helpers (uuid/hash, sanitize, hotkeys)  
 pages/ # Home, Train, Edit, Manage  
 types/ # Deck/Card typer + Zod schema  
 styles/ # tailwind.css etc  
 main.tsx, App.tsx  
public/  
 runtime/ # config.json (skapad av docker-entrypoint)

# 5) Typer & schema (Zod-validering)

• Skapa types/deck.ts med TS-typer och Zod-schema (DeckSchema) för uppladdad JSON.  
• Visa tydliga fel i UI vid ogiltiga filer (rad/kort om möjligt).

# 6) Persistens (IndexedDB)

• features/decks/deckRepo.ts med localforage: saveDeck, listDecks, deleteDeck, getDeck.  
• Separera stores: decks, progress, settings. Lägg schemaVersion i metadata för framtida migreringar.

# 7) Laddning av data

• deckService.ts:  
 - loadSeedDeckFromRuntime(): läser /runtime/config.json om det finns.  
 - importDeckFromFile(file): parse + validera + spara.  
 - importDeckFromUrl(url): (senare) med ETag/If-None-Match.  
• UI: Uploader (drag & drop + knapp) och DeckManager (lista, byt, export).

# 8) State (Zustand)

• deckStore.ts: activeDeckId, setActiveDeck, cards, filters, shuffleSeed.  
• progressStore.ts: per cardId: bucket (0/1/2), ease, interval, dueAt. Actions: markCorrect/markWrong/resetProgress.

# 9) Routing (React Router)

• Installera: react-router-dom (ingår ovan).  
• I main.tsx: wrappa appen i <BrowserRouter>.  
• Rekommenderad struktur:  
 / = Home (importera/hantera lekar)  
 /train/:deckId = Övningsläge (flip + rätt/fel, SR)  
 /edit/:deckId = Editor (redigera fråga/svar, taggar, svårighet)  
 /manage = Lista alla lekar, exportera, nollställ progress  
• Lägg till shadcn Navbar/Sidebar för navigation.

# 10) Kort & flip-UI

• Bygg Flashcard-komponent med shadcn Card + Framer Motion (flip 0°/180°).  
• Tangentbordsstöd: Space/Enter = vänd, ←/→ = föregående/nästa, 1/2 = fel/rätt.  
• Tillgänglighet: role="button", tabIndex, aria-attributes, respektera prefers-reduced-motion.

# 11) Övningsläge (MVP → enkel SR)

• 3 buckets: nytt → lärs → behärskat.  
• Rätt: öka intervallet (t.ex. +1h → +1d → +3d). Fel: flytta till lägre bucket.  
• Senare: SM-2 (ease, repetitions, interval) för mer exakt schemaläggning.

# 12) Editor

• EditorPanel (Dialog/Sheet) med Input/Textarea (shadcn) och RHF+Zod.  
• Lägg till/ta bort/ändra kort, autospara (debounce 300–500 ms).  
• Forka seed-lek till lokal kopia innan redigering (read-only seed).

# 13) Filter, sök & vyer

• Filter: kategori, taggar, svårighet (Select/Checkbox/Badge från shadcn).  
• Sökfält: Input + debounce.  
• Vyer: Bläddra, Öva, Quiz (fri text/multival senare).

# 14) Import/Export

• Exportera deck (+ valfritt progress) som JSON.  
• Importera: ersätt allt eller behåll progress om card.id matchar (fuzzy-match vid behov).  
• Sanera ev. markdown/HTML (dompurify).

# 15) PWA (offline)

• vite-plugin-pwa: cachea statiska assets och ge installbar app.  
• Hantera /runtime/config.json med stale-while-revalidate (inte hårdcacha seed-konfig).

# 16) Testning

• Enhet: Zod-validering, repetitionlogik, repositories (mocka localforage).  
• UI: React Testing Library – flip, tangentbord, importfel, editor autosave.

# 17) Kvalitet

• ESLint + Prettier, CI för lint/typecheck/test.  
• A11y-pass: fokus, kontrast, aria.  
• Säkerhet: CSP (minst default-src 'self'), X-Content-Type-Options: nosniff.

# 18) Bygg för produktion

npm run build (Vite bygger till dist/)  
npm run preview

# 19) Docker (seed + statisk server)

Dockerfile (2-steg: build → Nginx). docker-entrypoint.sh genererar runtime/config.json med DECK\_URL.  
Nginx SPA-routing: try\_files $uri /index.html;  
Körningsexempel:  
docker build -t viktor/flipdeck:0.1.0 .  
docker run --rm -p 8080:80 -e DECK\_URL="/data/projektledning.json" -v $(pwd)/decks:/usr/share/nginx/html/data viktor/flipdeck:0.1.0

# 20) CI/CD (rekommenderat)

• GitHub Actions: install → lint → typecheck → test → build.  
• Release: bygg & pusha Docker-image (tags: latest + git-tag).

# 21) Definition of Done (MVP)

[ ] UI-uppladdning med Zod-validering  
[ ] Visa/flip kort (tangentbord + touch)  
[ ] IndexedDB-lagring (lekar + progress)  
[ ] Enkel 3-bucket repetition  
[ ] Editor (Dialog/Sheet) + autosave  
[ ] React Router (Home/Train/Edit/Manage)  
[ ] Export/import  
[ ] Docker-image med seed via DECK\_URL

# 22) Nästa steg (post-MVP)

• Multival & fritext-quiz (auto-rättning)  
• Statistik (per kategori/tagg)  
• URL-import + "Ladda om från källa" (ETag)  
• Anki/CSV-import  
• AI-assist: generera kort från text/URL

Yes! Här är en **koncis, komplett step-by-step byggprocess** för ditt projekt (React + TS + Vite + shadcn/ui + React Router + localForage + Docker). Du kan följa den rakt av på en tom dator.

**0) Förutsättningar**

* **Node 20+**, **npm 10+**
* **Git**
* (Senare) **Docker Desktop**

**1) Skapa projektet (Vite + React + TS)**

npm create vite@latest flashcards -- --template react-ts

cd flashcards

git init && git add -A && git commit -m "chore: init"

**2) Installera beroenden**

# App-beroenden

npm i react-router-dom zustand localforage zod framer-motion react-hook-form @hookform/resolvers

# UI & styling

npm i -D tailwindcss postcss autoprefixer

# shadcn/ui CLI

npx shadcn-ui@latest init

# Kvalitet & test (valfritt men rekommenderat)

npm i -D eslint @typescript-eslint/eslint-plugin @typescript-eslint/parser eslint-config-prettier \

vitest jsdom @testing-library/react @testing-library/user-event @types/node

# PWA (senare)

npm i -D vite-plugin-pwa

**3) Initiera Tailwind**

npx tailwindcss init -p

tailwind.config.ts – lägg till shadcn-paths:

export default {

content: [

"./index.html",

"./src/\*\*/\*.{ts,tsx}",

"./components/\*\*/\*.{ts,tsx}" // shadcn/ui genererar här

],

theme: { extend: {} },

plugins: [],

}

src/index.css (eller src/styles/tailwind.css om du hellre vill):

@tailwind base;

@tailwind components;

@tailwind utilities;

**4) Sätt path-alias**

tsconfig.json:

{

"compilerOptions": {

"baseUrl": ".",

"paths": { "@/\*": ["src/\*"] },

"strict": true,

"jsx": "react-jsx",

"moduleResolution": "node",

"esModuleInterop": true

}

}

vite.config.ts:

import { defineConfig } from "vite"

import react from "@vitejs/plugin-react"

import path from "path"

export default defineConfig({

plugins: [react()],

resolve: {

alias: { "@": path.resolve(\_\_dirname, "./src") }

}

})

**5) Lägg in shadcn/ui-komponenter**

Addera de du garanterat använder:

npx shadcn-ui@latest add button card input textarea dialog sheet select table progress badge toast

Tips: Lägg senare till fler med npx shadcn-ui@latest add <komponent> när behov uppstår.

**6) Skapa mappstruktur**

src/

app/ # layout, providers

components/ # dina UI-komponenter (utöver shadcn/ui)

features/

decks/ # deckRepo, deckService, deckStore

train/ # trainLogic, trainStore, TrainView

lib/ # helpers (hash/uuid, sanitize, hotkeys)

pages/ # Home, Train, Edit, Manage

types/ # Deck + Zod schema

main.tsx

App.tsx

public/

runtime/ # config.json (skapas av Docker vid runtime)

**7) Typer & schema (Zod)**

src/types/deck.ts

import { z } from "zod"

export const CardSchema = z.object({

id: z.string(),

question: z.string(),

answer: z.string(),

category: z.string().optional(),

tags: z.array(z.string()).optional(),

difficulty: z.number().min(1).max(3).optional()

})

export const DeckSchema = z.object({

title: z.string(),

lang: z.string().optional(),

cards: z.array(CardSchema).min(1)

})

export type Card = z.infer<typeof CardSchema>

export type Deck = z.infer<typeof DeckSchema>

**8) Lagring med localForage**

src/features/decks/deckRepo.ts

import localforage from "localforage"

import type { Deck } from "@/types/deck"

localforage.config({ name: "flashcards", storeName: "data" })

const DECKS\_KEY = "decks"

const PROGRESS\_KEY = "progress"

export async function listDecks(): Promise<Deck[]> {

return (await localforage.getItem<Deck[]>(DECKS\_KEY)) ?? []

}

export async function saveDeck(deck: Deck) {

const all = await listDecks()

const idx = all.findIndex(d => d.title === deck.title)

if (idx >= 0) all[idx] = deck; else all.push(deck)

await localforage.setItem(DECKS\_KEY, all)

}

export async function deleteDeck(title: string) {

const all = await listDecks()

await localforage.setItem(DECKS\_KEY, all.filter(d => d.title !== title))

}

export async function getProgress() {

return (await localforage.getItem<Record<string, unknown>>(PROGRESS\_KEY)) ?? {}

}

export async function setProgress(p: Record<string, unknown>) {

await localforage.setItem(PROGRESS\_KEY, p)

}

**9) State med Zustand**

src/features/decks/deckStore.ts

import { create } from "zustand"

import type { Deck } from "@/types/deck"

type DeckState = {

active?: Deck

setActive: (d?: Deck) => void

}

export const useDeckStore = create<DeckState>((set) => ({

active: undefined,

setActive: (d) => set({ active: d })

}))

src/features/train/trainStore.ts

import { create } from "zustand"

import type { Card } from "@/types/deck"

type TrainState = {

queue: Card[]

index: number

flipped: boolean

setQueue: (q: Card[]) => void

next: () => void

flip: () => void

}

export const useTrainStore = create<TrainState>((set) => ({

queue: [], index: 0, flipped: false,

setQueue: (q) => set({ queue: q, index: 0, flipped: false }),

next: () => set(s => ({ index: Math.min(s.index + 1, s.queue.length - 1), flipped: false })),

flip: () => set(s => ({ flipped: !s.flipped }))

}))

**10) Routing (React Router)**

src/main.tsx

import React from "react"

import ReactDOM from "react-dom/client"

import { BrowserRouter } from "react-router-dom"

import App from "./App"

import "./index.css"

ReactDOM.createRoot(document.getElementById("root")!).render(

<React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</React.StrictMode>

)

src/App.tsx

import { Routes, Route, Navigate } from "react-router-dom"

import Home from "@/pages/Home"

import Train from "@/pages/Train"

import Edit from "@/pages/Edit"

import Manage from "@/pages/Manage"

export default function App() {

return (

<Routes>

<Route path="/" element={<Home />} />

<Route path="/train/:deckId" element={<Train />} />

<Route path="/edit/:deckId" element={<Edit />} />

<Route path="/manage" element={<Manage />} />

<Route path="\*" element={<Navigate to="/" replace />} />

</Routes>

)

}

**11) Sidor (skelett)**

src/pages/Home.tsx – importera JSON + välj lek

import { useState } from "react"

import { useNavigate } from "react-router-dom"

import { Button } from "@/components/ui/button"

import { Card, CardContent } from "@/components/ui/card"

import { DeckSchema, type Deck } from "@/types/deck"

import { saveDeck, listDecks } from "@/features/decks/deckRepo"

export default function Home() {

const [decks, setDecks] = useState<Deck[]>([])

const nav = useNavigate()

async function refresh() { setDecks(await listDecks()) }

async function onFile(e: React.ChangeEvent<HTMLInputElement>) {

const f = e.target.files?.[0]; if (!f) return

const text = await f.text()

const parsed = DeckSchema.parse(JSON.parse(text))

await saveDeck(parsed); await refresh()

}

return (

<div className="p-6 space-y-4">

<div className="flex items-center gap-3">

<input type="file" accept="application/json" onChange={onFile}/>

<Button onClick={refresh}>Ladda lekar</Button>

<Button variant="secondary" onClick={() => nav("/manage")}>Hantera</Button>

</div>

<div className="grid sm:grid-cols-2 lg:grid-cols-3 gap-4">

{decks.map(d => (

<Card key={d.title} className="cursor-pointer" onClick={() => nav(`/train/${encodeURIComponent(d.title)}`)}>

<CardContent className="p-4">

<div className="font-semibold">{d.title}</div>

<div className="text-sm opacity-70">{d.cards.length} kort</div>

</CardContent>

</Card>

))}

</div>

</div>

)

}

src/pages/Train.tsx – flip + nästa

import { useParams } from "react-router-dom"

import { useEffect, useState } from "react"

import { listDecks } from "@/features/decks/deckRepo"

import type { Deck } from "@/types/deck"

import { useTrainStore } from "@/features/train/trainStore"

import { Button } from "@/components/ui/button"

import { Card as UiCard, CardContent } from "@/components/ui/card"

import { motion } from "framer-motion"

export default function Train() {

const { deckId } = useParams()

const [deck, setDeck] = useState<Deck>()

const { queue, index, flipped, setQueue, next, flip } = useTrainStore()

useEffect(() => {

(async () => {

const all = await listDecks()

const d = all.find(x => x.title === decodeURIComponent(deckId ?? ""))

setDeck(d)

if (d) setQueue(d.cards)

})()

}, [deckId, setQueue])

if (!deck) return <div className="p-6">Hittar inte lek.</div>

const c = queue[index]; if (!c) return <div className="p-6">Inga kort.</div>

return (

<div className="p-6 space-y-4">

<div className="text-lg font-semibold">{deck.title}</div>

<UiCard className="w-[28rem] h-56 cursor-pointer" onClick={flip}>

<CardContent className="relative w-full h-full">

<motion.div

className="absolute inset-0 flex items-center justify-center p-4 text-center text-lg"

animate={{ rotateY: flipped ? 180 : 0 }}

transition={{ duration: 0.6 }}

style={{ backfaceVisibility: "hidden", transformStyle: "preserve-3d" }}

>

{flipped ? c.answer : c.question}

</motion.div>

</CardContent>

</UiCard>

<div className="flex gap-2">

<Button onClick={flip}>Vänd</Button>

<Button variant="secondary" onClick={next}>Nästa</Button>

</div>

</div>

)

}

Edit.tsx och Manage.tsx kan börja som enkla sidor och fyllas på senare.

**12) Kör utvecklingsservern**

npm run dev

Öppna http://localhost:5173 (eller porten Vite visar).

**13) Bygg för produktion**

npm run build

npm run preview

**14) Dockerisera (seed + statisk server)**

**Dockerfile**

# Build

FROM node:20-alpine AS build

WORKDIR /app

COPY package\*.json ./

RUN npm ci

COPY . .

RUN npm run build

# Runtime

FROM nginx:alpine

COPY --from=build /app/dist /usr/share/nginx/html

COPY docker-entrypoint.sh /docker-entrypoint.d/10-gen-config.sh

# SPA fallback

RUN sed -i 's|try\_files \$uri \$uri/ =404;|try\_files \$uri /index.html;|' /etc/nginx/conf.d/default.conf

**docker-entrypoint.sh**

#!/bin/sh

set -e

DECK\_URL="${DECK\_URL:-/data/deck.json}"

mkdir -p /usr/share/nginx/html/runtime

echo "{\"deckUrl\":\"$DECK\_URL\"}" > /usr/share/nginx/html/runtime/config.json

**Build & run**

docker build -t viktor/flipdeck:0.1.0 .

docker run --rm -p 8080:80 \

-e DECK\_URL="/data/projektledning.json" \

-v $(pwd)/decks:/usr/share/nginx/html/data \

viktor/flipdeck:0.1.0

**15) (Valfritt) PWA**

* Lägg till vite-plugin-pwa i vite.config.ts och ett manifest.
* Cachea statiska assets; använd **stale-while-revalidate** för /runtime/config.json.

**16) (Valfritt) Test & kvalitet**

* Vitest + React Testing Library för komponenter (flip, importfel).
* ESLint/Prettier och gärna en GitHub Actions-workflow för lint → typecheck → test → build.

**Klart!**

Med stegen ovan kan du:

* Ladda upp JSON i UI:t (Home), lagra lokalt med **localForage**.
* Öva med flip-kort (Train).
* Navigera med **React Router**.
* Köra som **Docker**-image med valfri **seed-lek** utan att tappa möjligheten att importera/redigera i UI:t.

Vill du att jag paketerar en **zip med alla skelettfiler** ovan så du kan droppa rakt in i repo:t?