Mandatory assignment 3: Vibe coding

Elisabeth Mathisen S381128
Per-Erik Glærum Grønvik S385710
Umeshgaar Balasubramaniam S385698

Vibe coding is a novel term for a software development strategy that involves prompting a Large Language Model(LLM) to generate code.

In this assignment, you will try to use vibe coding to produce an application. It can be a game, a web application, or anything else you find interesting or useful.

You can use Cursor AI (https://cursor.com/

Links to an external site.

), like we did in the lab. You can sign up for a free trial on their website. Furthermore, you are also free to use any other vibe coding tool or a plain LLM to generate your application.

Write a report, between 2 and 5 pages long, 1500 words at most, and submit it as a .pdf file. There is no need to submit the code, as there is no requirement to have a working app by the end of this assignment. **Do not use an LLM, like chatGPT, to write the report.**

Reflect on what works well, and what does not. You can choose your own focus, such as the code quality, the LLM's choice of technologies, or the security of the application. You can also focus entirely on the prompting and user experience of the vibe coding process.

Remember to cite any sources you use.

Work in groups of 1-4, and submit a single .pdf file containing your report.

Good luck!

Group 16 - En TV app

Introduksjon

Vibe koding

Vibe Koding er en kodeteknikk hvor du spør en LLM om den kan generere en kodesnutt basert på noen parametre du gir den. Den som utvikler programmet sjekker nødvendigvis ikke over koden som har blitt generert og redigerer det heller ikke. De spør heller LLM-en om de kan vurdere og evaluere kvaliteten på koden og ber den også gjøre endringer og forbedringer. Det er veldig lite menneskelig intuitet som blir brukt under denne kodeprosessen.

LLM's

LLM står for Large Language Model, hvis man vil oversette det til norsk så blir det Stor Språk Modell. Det er en språkmodell som ved hjelp av AI er trent opptil å gå gjennom og forstå enorme mengder med tekst og gjenkjenne mønstre i disse tekstene. De kan også generere store mengder tekst i forhold til hva modellen blir spurt om av oss mennesker, og basert på all teksten språkmodellen har iterert gjennom så gjenkjenner den mønstre og svarer på forespørselen mennesker spør om.

Formål

Formålet med oppgaven er å teste Visual Studio Code med alle beta-funksjoner i bakteppe. Det ligger en forelesning med Cursor som er basert på den.

Struktur

I metodedelen beskriver vi hvilke verktøy som ble brukt, hva applikasjonen som er valgt er, prompt prosessen og hvordan prompts ble forbedret og endret.

Hvilke begrensninger hadde Vibekodemiljøet.

Resultater, Hva som skjedde underveis.

Diskusjon om hva som fungerte bra, hva fungerte dårlig, hvordan var kodekvaliteten fordeler og ulemper med metoden.

Konklusjonsdel en kjapp oppsumering av diskusjonsdelens konklusjoner, refleksjoner og hvilken rolle AI har i utvikling.

Metode

Verktøy

Visual Studio Code utgave 1.105.1 heretter VS Code, er et koderedigeringsprogram som er gratis og har åpen kildekode. Det har et aktivt opensource miljø og det bygges stadig nye extensions.

Claude sonnet 4.5 er laget av Anthropic. Claude er fult integrert i VS code, dersom det aktiveres og har tilgang til repoet.

Github og git brukes til versjonskontroll

Docker Desktop for å kjøre docker.

TheMovieDB API nøkkel som er gratis å bruke.

Vite for å reloade koden live og se endringer.

Applikasjonen som er valgt er en TV meny som viser hvilket innhold som er tilgjengelig og det er mulig å scrolle uendelig i 8 retninger.

Før utvikler begynte å prompte så satt han opp en filstruktur med public, scripts, src, en env fil med api nøkkel og readme.

Resultater

Eksempel på prompts:

Prompt 1
“create a gitignore file that ignores env” fungerte.

Prompt 2
“
I want a tv screen scaled app with.

React + Vite + plain CSS

A matrix of a grid that is 15x15. and just fill it with movie names for now.

The app shows a search bar.
and under that it has a 3x3 grid showing in the middle.

3x3 grids is always shown, but the movies in it is based on the big grid.
Button presses are used to navigate.
If 1 is pressed the grid moves one left
if 2 or 'up arrow' is pressed the grid moves up.
if 3 is pressed right and up
if 4 or 'left arrow', is pressed left.
if 5 is pressed select
if 6 or 'right arrow' is pressed right.
if 7 is pressed left down
if 8 or 'down arrow', is pressed down
if 9 is pressed down right.
“
LLm’en skjønte automatisk hvor og i hvilken mappe den skulle plassere innholdet.

Videre ba utvikler den om å sette opp å starte en docker med prosjektet og kjørte docker og kunne nå se kodeendringer live også mens LLM eller at han kodet. Hvis han endret padding eller margin i CSS filen så han det live.

Siden applikasjonen kunne vises live, ble det mulig å gi løpende instruksjoner om endringer i utseendet.

Prompt for å endre på utseende.
“
remove the search bar.

Inside the tv screen container to separate two parts of content. add one invisible div that uses 15 percent for a menu on the left and the grid on the right.

put the grid inside the right div and and make it use the entire space by percent scaling the 9 squares.

On the left inside the left grid of the screen add a menubar with no spacing, no margin no padding, no icons. The menubar has 4 options with the title "Movie picker"
under is has a searchbar, user, and settings bar, back option.

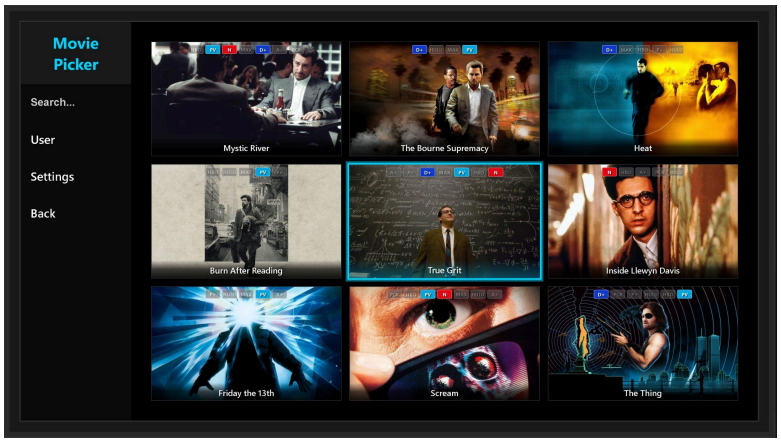
the bar follow a simplistic tv style. where selected is light blue. (none are selected as the movie picked is in focus.) by pressing b it goes back to the button.
”

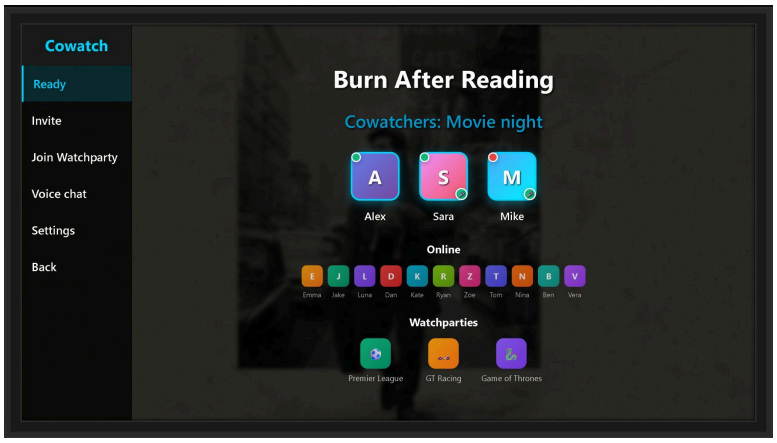
Prompt fordi LLM ikke forstår konteksten om hvordan divs virker

“

The right side needs another div around the grid with some padding, for the grid to stay in it's place.
”

Den ferdige TV appen





Diskusjon

Erfaringer med prompting

Det som fungerte best var å bruke koderfaring og kodebegreper og min forståelse av hva LLM'en hadde laget til å påvirke den i promptene mine.

Før utvikler kom fram til prompt 2, så måtte utvikler prøve meg fram og fikk ikke det utvikler ønsket. Men det var hele tiden mulig å angre og endre promptet slik at det ble som han ønsket å ha det. For å få best mulig prompt kvalitet så må en skrive litt robotisk og rart.

LLM'en reloadet dockeren selv om den var live med Vite mange ganger og det var helt unødvendig, heldigvis sluttet den med det etter en stund.

Det var en mismatch mellom film navnene og filmene, som skyldtes at LLM'en laget et skript til filmene som var fra X og Y og et skript til filmnavnene som var Y og X. Utvikleren fikk den til å rette det opp, men så la han til beskrivelse for filmene. LLM lastet inn beskrivelse i Y og X og så endret den titlene til Y og X slik at det ikke lenger passet sammen med bildene. Han brukte mange prompts for å få den til å rette det opp, men den klarte ikke og ved hvert prompt så endret den ting og la til ting som ikke var forespurt. Ved å gjenopprette med et savepoint i GIT ungikk løste han problemet. Det førte til en ny strategi som var å lagre hvert eneste prompt i git. Og å legge til "Do not do anything I didn't tell you to do" i prompt.

Kodekvalitet og struktur

Koden var litt rotete og Utvilkler måtte endre ting slik som padding og marginer i CSS manuelt. Logikken i skript, titler og beskrivelse sammen med infinite scroll fungerte aldri 100%. Det indikerer at hvis utvikler skulle publisert produktet, så hadde han måtte brukt tid på å rette opp logikken. Siden det er bygd videre på den feile logikken så kunne det ha tatt med tid på grunn av debugging og å prøve å fikse koden. Enn om han bare hadde gjort det selv til å begynne med.

Mange ganger skrev modellen kode som ikke var forespurt, la til meny items og en del av tiden gikk med på å lete gjennom koden og slette disse. Når modellen produserte riktig kode med ekstra ting, så var det lettere å fikse det selv enn å rulle tilbake og redigere promptet. Det å kjøre prompt kunne noen ganger ta mye tid.

LLM-atferd og begrensinger

Metoden fungerer ganske bra til rask prototyping som ikke trenger å være stabil. Utvikler ble ikke like godt kjent med kodebasen som ved vanlig koding og det resultatet tyder på at terskelen for videre bruk av vibecoding for å fikse småting og dermed lage nye feil øker. Når modellen setter seg fast så må utvikler bruke mye tid på å nøste opp i problemer som modellen har laget i kodebasen som han ikke er kjent med.

Konklusjon

Vibekoding har en rolle spesielt i rask prototyping, vi kan se for oss at å lage en fungerende utgave av en app som ikke er helt perfekt er bedre enn å vise til et design i figma. Det er mer realistisk og nært det faktiske produktet som kunden kommer til å få til slutt. Og hvis en kan å kode og leser veldig nøye gjennom hva modellen lager, så kan det brukes til å kode noen deler av et produkt.