

Low-Code Development and Generative AI

Thiago Rocha Silva (trsi@mmmi.sdu.dk)
Associate Professor

AI-empowered low-code development

- The recent breakthroughs in **generative AI** through **large language models (LLMs)** have opened up many new possibilities for text, image, and **code generation**.
- **Low-code development platforms** have, therefore, started experimenting with **LLMs** for **program synthesis**.
- **Program synthesis** is the task of **automatically finding programs** from the underlying programming language that satisfy **user intent** expressed in **some form of constraints**.

AI-empowered low-code development

- Program synthesizers typically perform **some form of search** over the space of programs to **generate a program** that is **consistent with a variety of constraints**.
 - ❖ These **constraints** can be in the form of **input-output examples**, demonstrations, **natural language**, partial programs, assertions, etc.
- The problem of **program synthesis** has long been considered the **holy grail** of Computer Science.

Home > Software Development > No-Code and Low-Code

Why generative AI will turbocharge low-code and no-code development

The evolution of generative AI models will further lower barriers to using low-code and no-code development tools, and potentially lead to the birth of a whole new class of intelligent developer technology.



Home > Software Development

Mendix aims to add generative AI to its low-code platform by year-end

The company has been trying out GPT-based features for its low-code platform, planning to add capabilities to streamline app development and offer guidance on best practices.



By **Anirban Ghoshal**
Senior Writer, InfoWorld | APR 25, 2023 8:01 AM PDT

Power Apps

Announcing a next-generation AI Copilot in Microsoft Power Apps that will transform low-code development



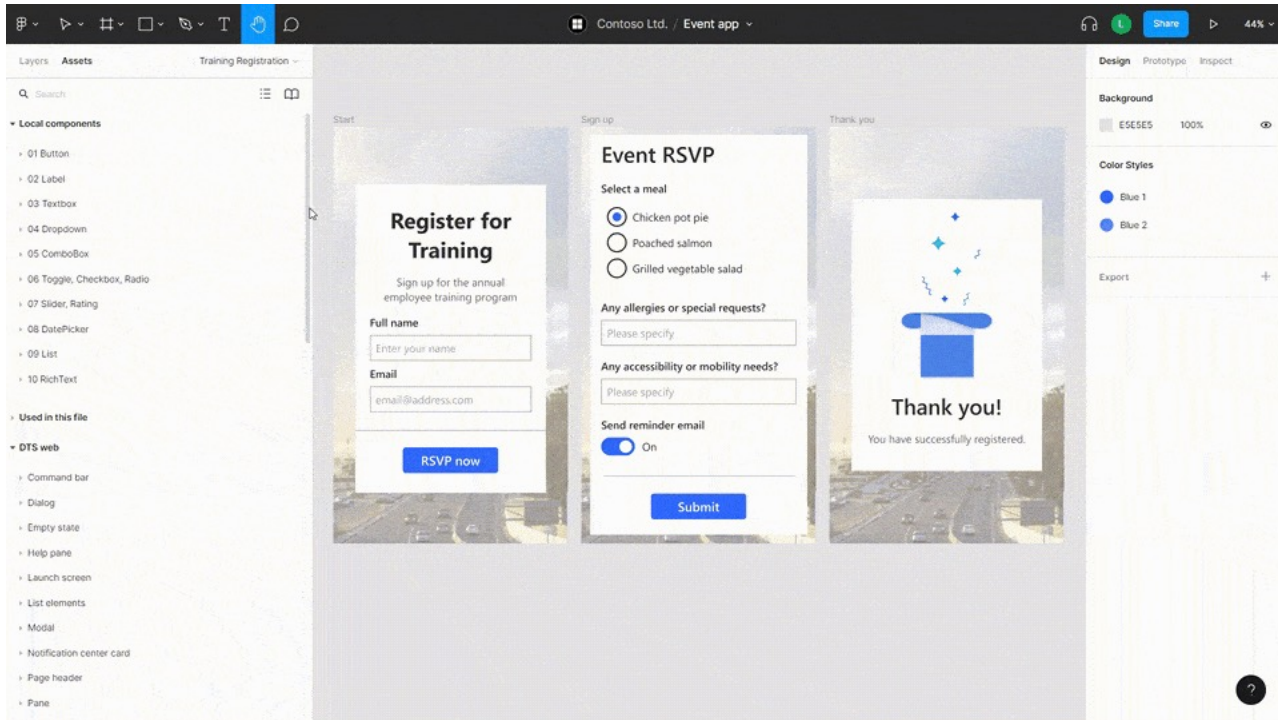
Ryan Cunningham, VP, Power Apps, 16 March 2023

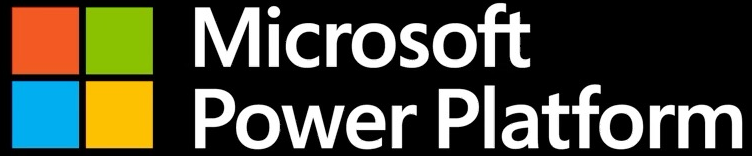


AI-empowered low-code development

- Early analysis¹ indicates that the market for generative AI for application development will be **\$4.1 billion by the end of 2023**, growing at approximately **32% a year**.

From design inputs to working low-code apps





Sketch a form
and turn it into an app
with Power Apps express design

Conversational AI

- Conversational AI is a type of **artificial intelligence (AI)** that can **simulate human conversation**.
- It works by using a combination of **natural language processing (NLP)** and **machine learning (ML)**.
- The model is **constantly learning** from its interactions and improving its response quality over time.
- Conversational AI has been receiving a lot of attention due to the latest breakthroughs in **large language models (LLMs)**.

I need to write a Python function called `count_common`. Here's a description: 'Write a function to count the most common words in a dictionary.'

Can you give me an example?

```
assert count_common(['one', 'two', 'three', 'four', 'five', 'one', 'two', 'one', 'three', 'one']) == [(('one', 4), ('two', 2), ('three', 2), ('four', 1))]
```

```
def count_common(list_of_words):
    dict = {}
    for word in list_of_words:
        if word in dict:
            dict[word] += 1
        else:
            dict[word] = 1
    return sorted(dict.items(), key=lambda x: x[1])
```



tests fail

Close! But can you sort the output in reverse (descending) order?

```
def count_common(list_of_words):
    dict = {}
    for word in list_of_words:
        if word in dict:
            dict[word] += 1
        else:
            dict[word] = 1
    return sorted(dict.items(), key=lambda x: x[1], reverse=True)
```



tests fail

Great. Now notice that the example only returns the top 4 results.

```
def count_common(list_of_words):
    dict = {}
    for word in list_of_words:
        if word in dict:
            dict[word] += 1
        else:
            dict[word] = 1
    return sorted(dict.items(), key=lambda x: x[1], reverse=True)[:4]
```



tests pass

Hooray! 🎉🎉🎉🎉

I need to write a Python function called `find_Max_Num`. Here's a description: 'Write a python function to find the largest number that can be formed with the given list of digits.'

Can you give me an example?

```
assert find_Max_Num([1,2,3]) == 321
```

```
def find_Max_Num(test_list):
    largest = 0
    for i in range(0, len(test_list)):
        current = test_list[i]
        for j in range(0, len(test_list)):
            if (current == test_list[j]):
                current = current * 10
            else:
                break
        if (largest < current):
            largest = current
    return largest
```



tests fail

This function should sort the list in decreasing order and then combine the digits into an integer.

```
def find_Max_Num(test_list):
    return sorted(test_list)[-1]
```



tests fail

Join the string representations of the values in the sorted list.

```
def find_Max_Num(test_list):
    return ".join(map(str, sorted(test_list)))
```



tests fail

Sort the list from highest to lowest, not lowest to highest.

```
def find_Max_Num(test_list):
    return ".join(map(str, sorted(test_list, reverse=True)))
```



tests fail

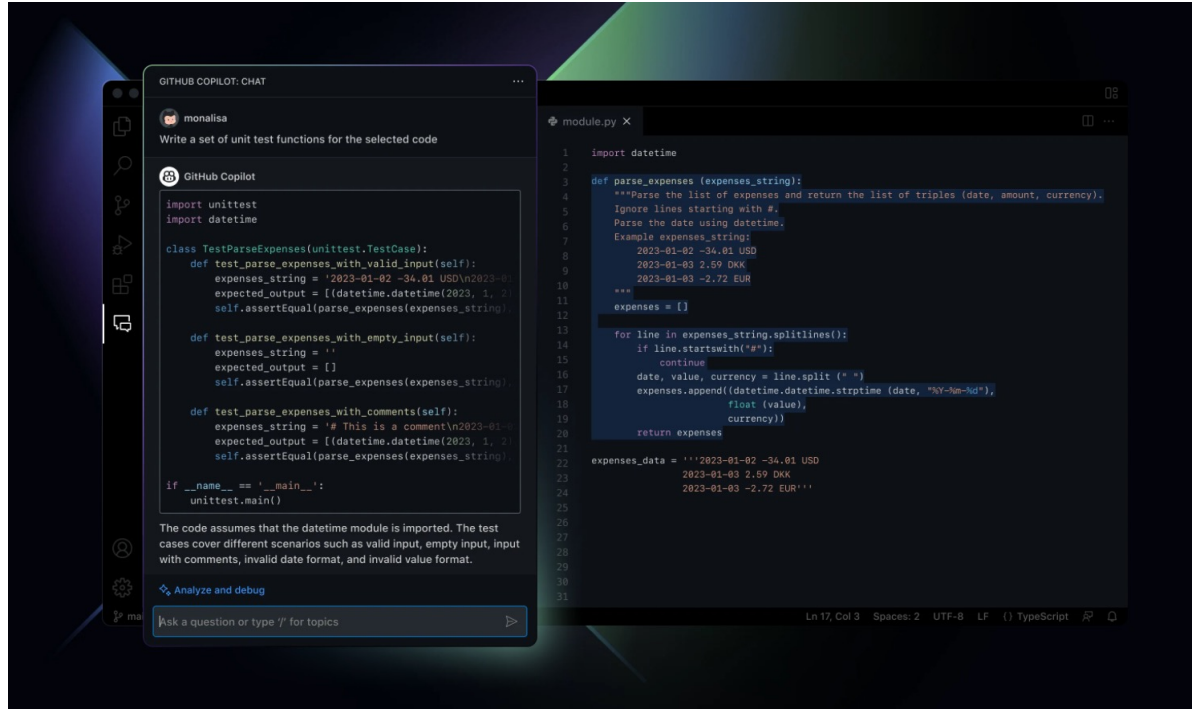
Return the int of the string.

```
def find_Max_Num(test_list):
    return int(".join(map(str, sorted(test_list, reverse=True))))
```

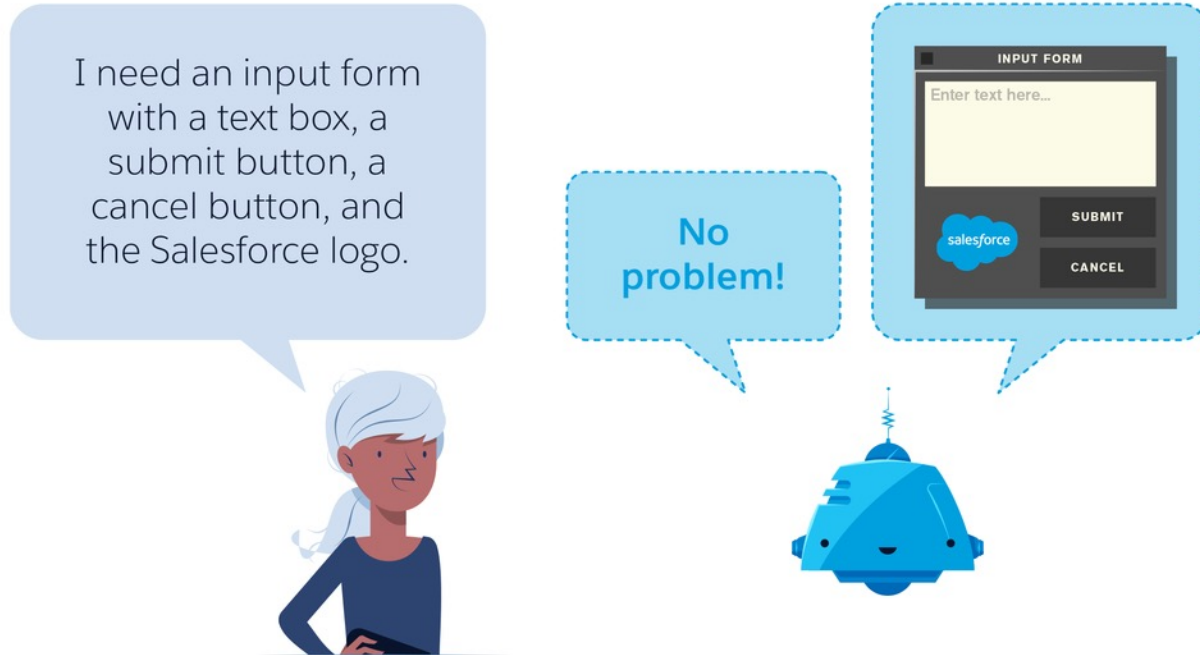


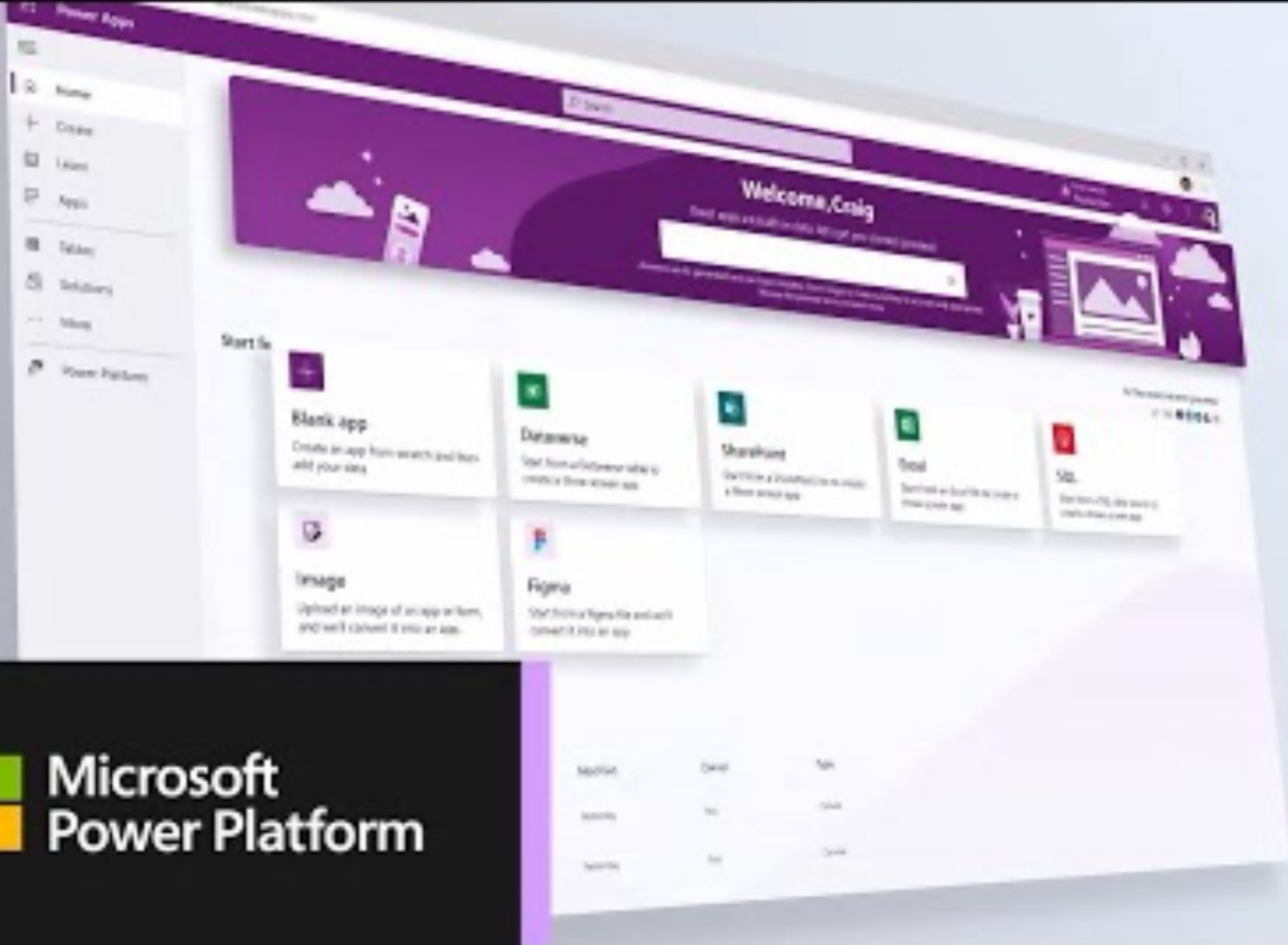
tests pass

GitHub Copilot X



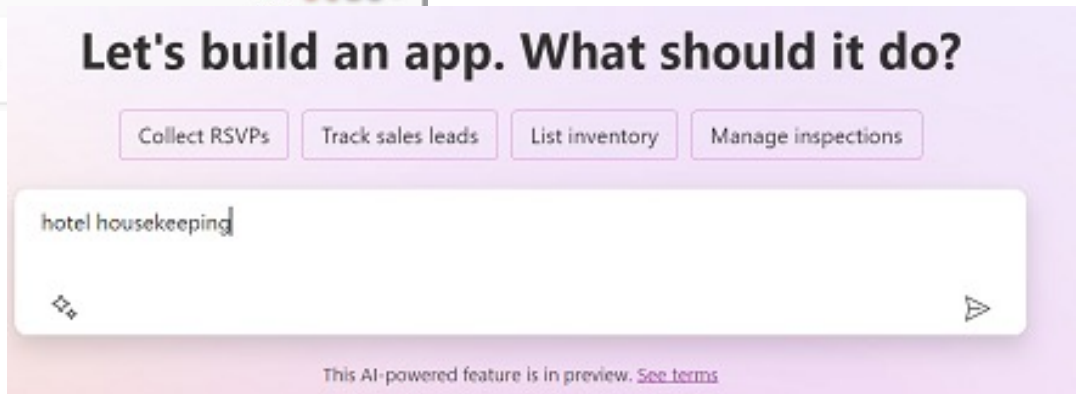
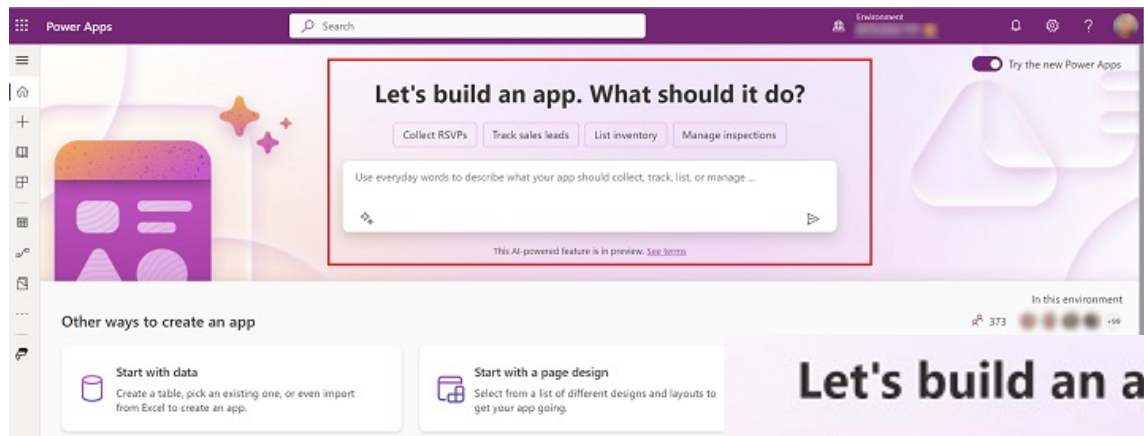
Low-code: Can we just ask for the program?





Microsoft
Power Platform

Low-code and conversational AI



Low-code and conversational AI

Create an app PREVIEW

Here's a table for your app

Why a table? Because when you see your app in a moment, the data in this table is what you'll see on the screen. There will also be a form that your users can fill out to add information to the table.

Need anything changed?
Just tell Copilot what you need, and it will make the changes for you.

Suggestions

- Change ... to ...
- Add a column for ...
- Add a row for ...
- Remove the ... row.
- Change the data type for the ... column to ...
- Refresh the data.
- Give me suggestions.

Edit table properties

Hotel Housekeeping

ID	Name	Room Number
0001	John	101
0002	Mary	102
0003	Ada	103
0004	Bob	104
0005	Alice	105

Copilot

hotel housekeeping
6 minutes ago

Here is a table for hotel housekeeping
6 minutes ago

Describe what you want changed...

Make sure AI-generated content is accurate and appropriate before using. [See terms](#)

Create app

Cancel

Low-code and conversational AI

Edit table properties

Hotel Housekeeping

ID	Name	Room Number
0001	John	101
0002	Mary	102
0003	Ada	103
0004	Bob	104
0005	Alice	105

Cancel

Create app

Copilot

hotel housekeeping

20 minutes ago

Here is a table for hotel housekeeping

19 minutes ago

Add columns to track start and end time

Make sure AI-generated content is accurate and appropriate before using. [See terms](#)

Cancel

Create app

Edit table properties

Hotel Housekeeping

End Date	Start Time	End Time
5/21/2020	5/20/2020 9:00 AM	5/21/2020 5:00 PM
4/19/2010	4/18/2010 10:00 AM	4/19/2010 6:00 PM
9/13/2015	9/12/2015 11:00 AM	9/13/2015 7:00 PM
6/16/2020	6/15/2020 12:00 PM	6/16/2020 8:00 PM
7/21/2020	7/20/2020 1:00 PM	7/21/2020 9:00 PM

Cancel

Create app

Copilot

hotel housekeeping

23 minutes ago

Here is a table for hotel housekeeping

22 minutes ago

Add columns to track start and end time

Just now

The Hotel Housekeeping table has been updated

Just now

Describe what you want changed...

Make sure AI-generated content is accurate and appropriate before using. [See terms](#)

Cancel

Create app

But...

Garbage in, garbage out has never
been more true.

So...

Prompt Engineering (asking the right question) has never been so important.

Prompt Engineering

Prompt engineering is the process of designing and optimizing prompts to LLMs for a wide variety of applications and research topics. Prompts are short pieces of text that are used to guide the LM's output. They can be used to specify the task that the LM is supposed to accomplish, provide additional information that the LM needs to complete the task, or simply make the task easier for the LM to understand.

Source: Google Bard

Prompt Engineering

- Prompt engineering is enabled by ***in-context learning***, which is a model's ability to **temporarily learn** from prompts.
 - ❖ In contrast to **training** and **fine tuning** for specific tasks, which are **not temporary**.
- ***In-context learning*** has been discovered as an ***emergent ability*** of LLMs (Wei et al., 2022) and it's a form of **meta-learning** or "learning to learn".
- At present, **the mechanisms of *in-context learning* are not well understood** and remain mostly an **intuition** (vonOswald et al., 2023; Garg et al., 2023).

One of the reasons we should care about the **prompt**: LLMs are **non-deterministic**!

*“Code generation models have achieved impressive performance. However, **they tend to be brittle as slight edits to a prompt could lead to very different generations.**”*
(Wang et al., 2022)

What is 1 + 1?

What's 1 + 1?

>> 1+1=2

What's 1 + 1?

>> 1 + 1 = 2.

What's 1 + 1?

>> 1 + 1 = 2.

What's 1 + 1?

>> I understand that you are asking me what is 1 + 1. The answer is 2.

Chat with Google Bard

Engineering the Question

What's 1+1 stated as an integer?

>> 1+1 stated as an integer is 2.

An integer is a whole number that can be positive, negative, or zero. It is no

Chat with Google Bard
(even **worse**)

Engineering the Question

What's 1+1? Only return the numeric answer.

>> 2

What's 1+1? Only return the numeric answer.

>> 2

What's 1+1? Only return the numeric answer.

>> 2

Chat with Google Bard
(**better**, more deterministic)

Prompt Engineering Techniques

Prompt Engineering Techniques

- Zero-Shot Prompting
 - Few-Shot Prompting
 - One-Shot Prompting
 - Chain-of-Thought (CoT) Prompting
 - Zero-shot COT Prompting
 - Generated Knowledge Prompting
- Least-to-most prompting
 - Self-consistency decoding
 - Complexity-based prompting
 - Self-refine
 - Tree of Thoughts
 - Maieutic prompting
 - Retrieval Augmented Generation
 - Automatic Reasoning and Tool-use
 - Automatic Prompt Engineer
 - Active-Prompt
 - Directional Stimulus Prompting
 - ReAct
 - Multimodal CoT
 - Graph Prompting
 - ...

Zero-Shot Prompting

- In **zero-shot prompting**, **no examples are provided** to the model, and **only the task request** is provided.

Prompt:

```
Classify the text into neutral, negative or positive.  
Text: I think the vacation is okay.  
Sentiment:
```

Output:

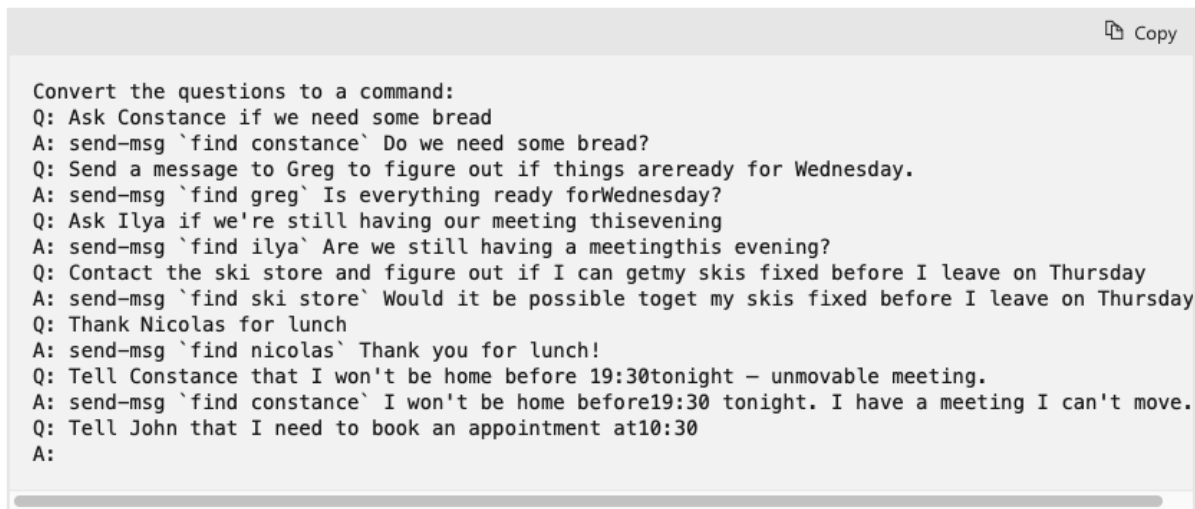
```
Neutral
```

Few-Shot Prompting

- While LLMs demonstrate remarkable zero-shot capabilities, **they still fall short on more complex tasks** when using the zero-shot setting.
- **Few-shot prompting** can be used as a technique to enable ***in-context learning*** where we provide **demonstrations in the prompt** to steer the model to better performance.
- The demonstrations serve as **conditioning** for subsequent examples where we would like the model to generate a response.

Few-Shot Prompting

- The user includes **several examples** in the prompt that demonstrate the **expected answer format and content**.



```
Convert the questions to a command:
Q: Ask Constance if we need some bread
A: send-msg `find constance` Do we need some bread?
Q: Send a message to Greg to figure out if things are ready for Wednesday.
A: send-msg `find greg` Is everything ready for Wednesday?
Q: Ask Ilya if we're still having our meeting this evening
A: send-msg `find ilya` Are we still having a meeting this evening?
Q: Contact the ski store and figure out if I can get my skis fixed before I leave on Thursday
A: send-msg `find ski store` Would it be possible to get my skis fixed before I leave on Thursday
Q: Thank Nicolas for lunch
A: send-msg `find nicolas` Thank you for lunch!
Q: Tell Constance that I won't be home before 19:30 tonight – unmovable meeting.
A: send-msg `find constance` I won't be home before 19:30 tonight. I have a meeting I can't move.
Q: Tell John that I need to book an appointment at 10:30
A:
```

One-Shot Prompting

- This case is **the same as the few-shot** approach except **only one example is provided**.



```
Convert the questions to a command:  
Q: Ask Constance if we need some bread  
A: send-msg `find constance` Do we need some bread?  
Q: Send a message to Greg to figure out if things are ready for Wednesday.  
A:
```

Zero-Shot Prompting

 Copy

```
Convert the question to a command:  
Q: Ask Constance if we need some bread  
A:
```

Chain-of-Thought (CoT) Prompting (Wei et al., 2022)

- Chain-of-Thought (CoT) prompting enables complex reasoning capabilities through **intermediate reasoning steps**.
- Can be **combined with few-shot prompting** to get better results on more complex tasks that require **reasoning before responding**.

Chain-of-Thought (CoT) Prompting (Wei et al., 2022)

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅

Zero-shot COT Prompting (Kojima et al., 2022)

- Essentially involves adding “**Let’s think step by step**” to the original prompt.

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

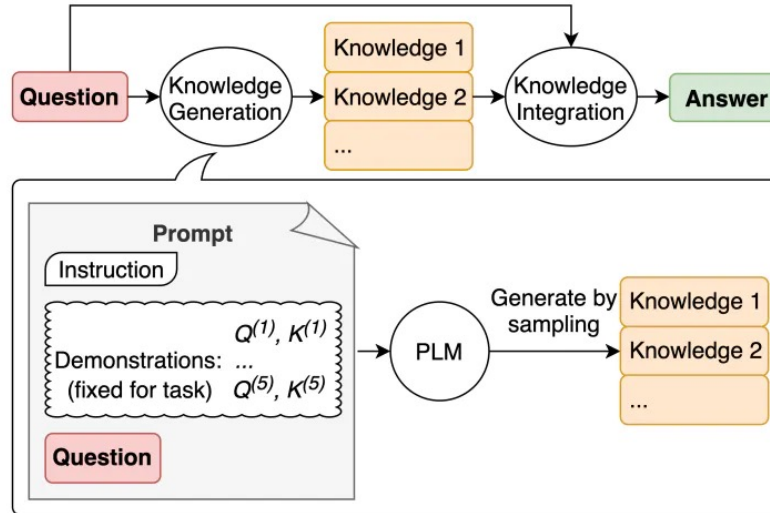
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let's think step by step.**

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Generated Knowledge Prompting (Liu et al., 2021)

- Generated Knowledge Prompting involves **incorporating knowledge or information** to help the model make more accurate predictions.



LLM Prompting for Programming Tasks

LLM prompting for programming

An example from Martin Fowler and Xu Hao, Thoughtworks.

Source: <https://martinfowler.com/articles/2023-chatgpt-xu-hao.html>

- The objective is to build **Self-Testing Code**.
- It starts with a prompt that **sets the context for the application** and **how you want the code to be structured**.
- Even though the final goal is to generate code and its tests, ChatGPT is initially instructed: **“Don’t generate code”**.

The current system is an online whiteboard system. Tech stack: typescript, react, redux, konvajs and react-konva. And vitest, react testing library for model, view model and related hooks, cypress component tests for view.

All codes should be written in the tech stack mentioned above. Requirements should be implemented as react components in the MVVM architecture pattern.

There are 2 types of view model in the system.

1. Shared view model. View model that represents states shared among local and remote users.
2. Local view model. View model that represents states only applicable to local user

Here are the common implementation strategy:

1. Shared view model is implemented as Redux store slice. Tested in vitest.
2. Local view model is implemented as React component props or states(by useState hook), unless for global local view model, which is also implemented as Redux store slice. Tested in vitest.
3. Hooks are used as the major view helpers to retrieve data from shared view model. For most the case, it will use 'createSelector' and 'useSelector' for memorization. Tested in vitest and react testing library.
4. Don't dispatch action directly to change the states of shared view model, use an encapsulated view model interface instead. In the interface, each redux action is mapped to a method. Tested in vitest.

5. View is consist of konva shapes, and implemented as react component via react-konva. Tested in cypress component tests

Here are certain patterns should be followed when implement and test the component

1. When write test, use describe instead of test
2. Data-driven tests are preferred.
3. When test the view component, fake view model via the view model interface

Awareness Layer

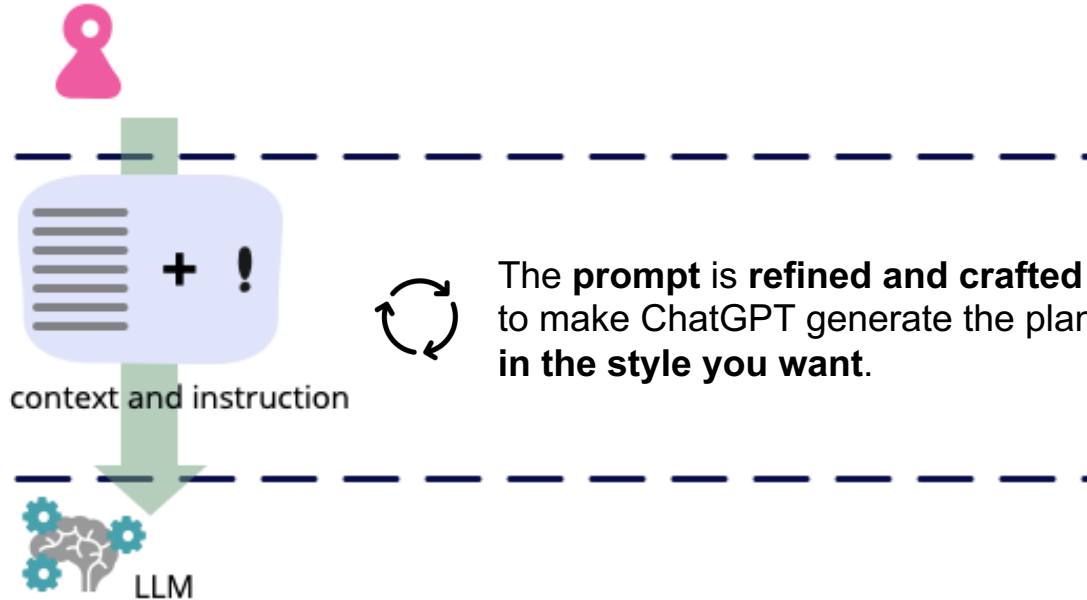
Requirement:

Display other users' awareness info(cursor, name and online information) on the whiteboard.

AC1: Don't display local user

AC2: When remote user changes cursor location, display the change in animation.

Provide an overall solution following the guidance mentioned above. Hint, keep all awareness information in a Konva layer, and an awareness info component to render cursor, and name. Don't generate code. Describe the solution, and breaking the solution down as a task list based on the guidance mentioned above. And we will refer this task list as our master plan.



LLM prompting for programming

An example from Martin Fowler and Xu Hao, Thoughtworks.

Source: <https://martinfowler.com/articles/2023-chatgpt-xu-hao.html>

- Initial prompt primes the LLM with an **implementation strategy** and encourages the model to **explain its reasoning** (*chain of thought prompting*).
- Prompt initially asks for an **implementation plan rather than code** (*generated knowledge prompting*).
- The plan is used later to **refine the implementation** and **generate useful sections of code**.

LLM prompting for programming

An example from Martin Fowler and Xu Hao, Thoughtworks.

Source: <https://martinfowler.com/articles/2023-chatgpt-xu-hao.html>

- This prompt technique is called **Generated Knowledge**¹.
 - ❖ First, we ask the LLM to **generate some useful information about the problem**, and then
 - ❖ We **feed that information back** into the LLM to generate the final product.

Solution:

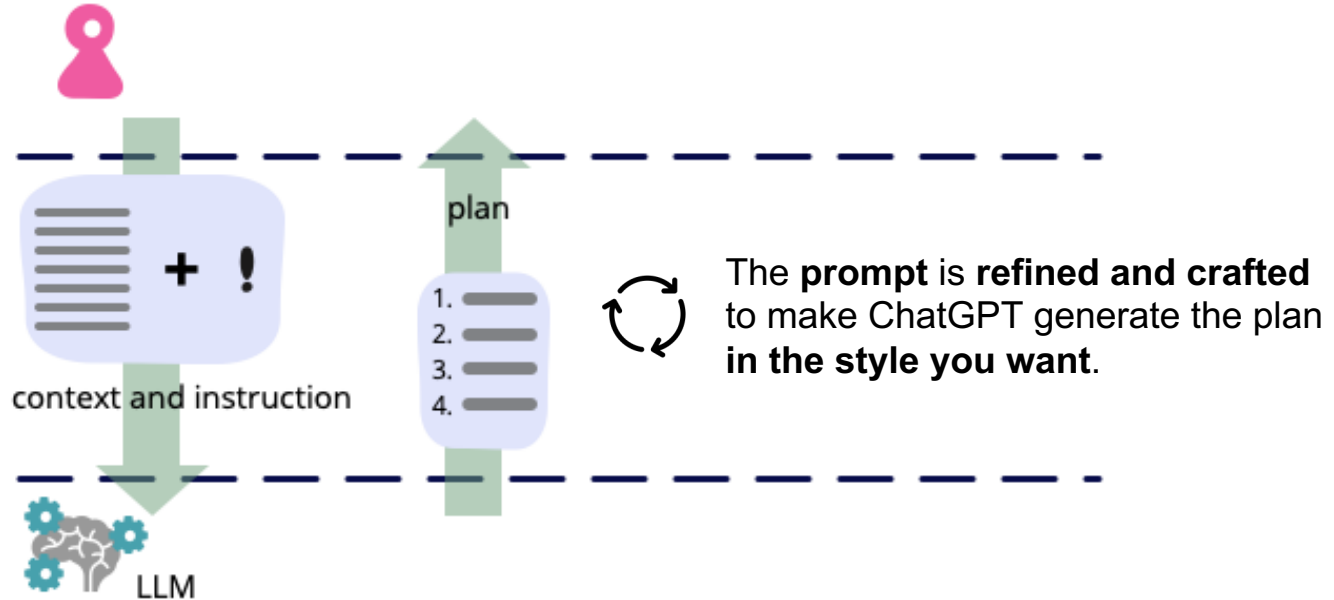
To implement the awareness layer feature, we can follow the guidelines mentioned in the problem statement and use the existing tech stack.

Task List:

1. Create a new component named "AwarenessInfo" to render cursor and name for remote users.
2. Create a new Konva layer to hold all the "AwarenessInfo" components.
3. Add the Konva layer to the main whiteboard component.
4. Create a new Redux slice to store the awareness information for remote users.
5. Create an encapsulated view model interface for the awareness Redux slice.
6. Implement the "AwarenessInfo" component to retrieve and display the awareness information for each remote user from the view model.
7. Implement a method in the encapsulated view model interface to update the awareness information for each remote user.
8. When a remote user changes the cursor location, dispatch the update method in the encapsulated view model interface to update the corresponding awareness information.
9. In the "AwarenessInfo" component, use the new location to animate the cursor change.
10. Write tests for the new "AwarenessInfo" component, including data-driven tests to ensure it properly displays the awareness information for all remote users.
11. Write tests for the encapsulated view model interface to ensure it properly updates the awareness information for remote users.
12. Write end-to-end Cypress tests to ensure the awareness layer is properly displayed and updated when remote users change their cursor location.

By following these tasks, we can implement the awareness layer feature in the online whiteboard system.

The key element of the response is an **ordered task list, with numbers for each step**. We can use these numbers to refer to these tasks for the rest of the session.

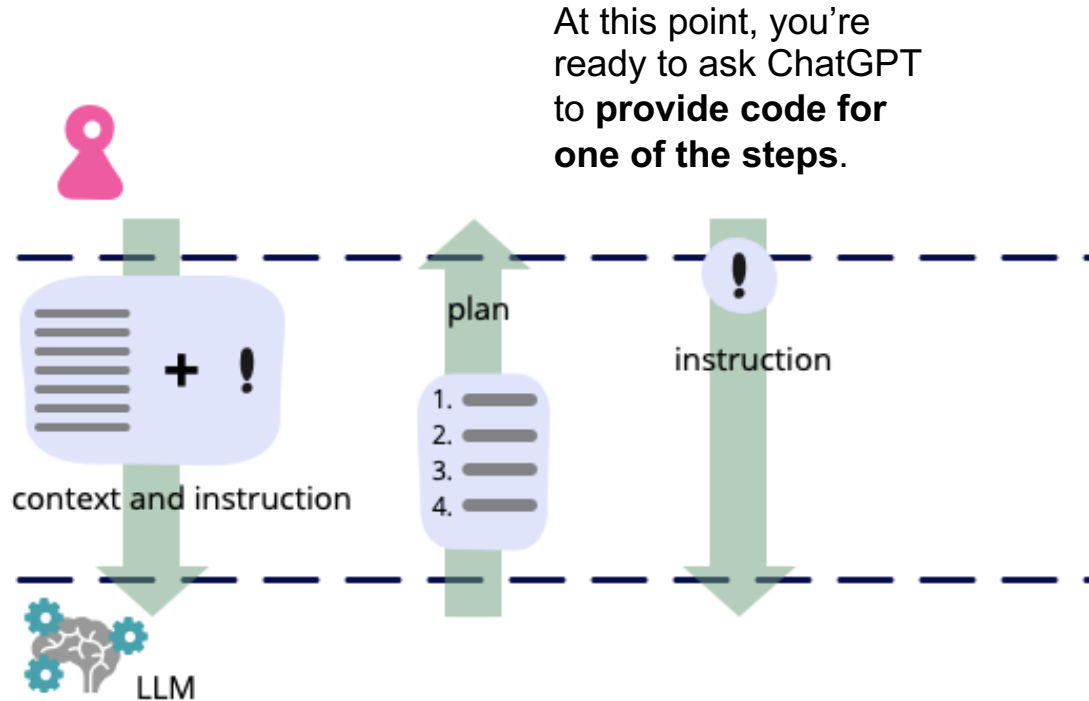


Rewrite the master plan, this time with detailed component names, methods, and props

Sure, here's a revised master plan with detailed component names, methods, and props:

Task List:

1. Create a new component named "AwarenessInfo" to render cursor and name for remote users. This component will receive props such as user name, cursor position, and online status.
2. Create a new Konva layer named "AwarenessLayer" to hold all the "AwarenessInfo" components.
3. Add the "AwarenessLayer" to the main whiteboard component.
4. Create a new Redux slice named "awareness" to store the awareness information for remote users.
5. Create an encapsulated view model interface named "AwarenessViewModel" for the "awareness" Redux slice. This interface will contain a method named "updateUserAwarenessInfo" to update the awareness information for each remote user.
6. Implement the "AwarenessInfo" component to retrieve and display the awareness information for each remote user from the view model. The component will use the "useSelector" hook to retrieve the relevant information.
7. Implement the "updateUserAwarenessInfo" method in the "AwarenessViewModel" interface to dispatch a Redux action to update the awareness information for each remote user. This method will be called by the "MainWhiteboard" component.
8. When a remote user changes the cursor location, dispatch the "updateUserAwarenessInfo" method in the "AwarenessViewModel" interface to update the corresponding awareness information.
9. In the "AwarenessInfo" component, use the new location to animate the cursor change. This can be accomplished using the Konva "Tween" animation.
10. Write tests for the new "AwarenessInfo" component, including data-driven tests to ensure it properly displays the awareness information for all remote users. The tests will be written using the React Testing Library.
11. Write tests for the encapsulated view model interface to ensure it properly updates the awareness information for remote users. These tests will use the Vitest framework.
12. Write end-to-end Cypress tests to ensure the awareness layer is properly displayed and updated when remote users change their cursor location. The tests will simulate user interactions with the whiteboard and verify the state of the "AwarenessLayer" and "AwarenessInfo" components.



provide me example tests and implementation for the awareness slice and AwarenessViewModel. Test first.

Sure, let's start with writing tests for the awareness slice and AwarenessViewModel:

Test for Awareness Slice:

```
describe('awareness slice', () => {
  let store: MockStore;
  beforeEach(() => {
    store = configureMockStore()(initialState);
  });

  it('should set the cursor location', () => {
    store.dispatch(setCursorLocation({ x: 100, y: 200 }));
    const state = store.getState().awareness;
    expect(state.cursorLocation).toEqual({ x: 100, y: 200 });
  });

  it('should set the name', () => {
    store.dispatch(setName('John Doe'));
    const state = store.getState().awareness;
    expect(state.name).toBe('John Doe');
  });

  it('should set the online status', () => {
    store.dispatch(setOnline(true));
    const state = store.getState().awareness;
    expect(state.isOnline).toBe(true);
  });
});

Test for AwarenessViewModel:
```

```
describe('AwarenessViewModel', () => {
  let store: MockStore;
  let awarenessViewModel: AwarenessViewModel;
  beforeEach(() => {
    store = configureMockStore()(initialState);
    awarenessViewModel = new AwarenessViewModel(store);
  });

  it('should set the cursor location', () => {
    awarenessViewModel.setCursorLocation({ x: 100, y: 200 });
    const state = store.getState().awareness;
    expect(state.cursorLocation).toEqual({ x: 100, y: 200 });
  });

  it('should set the name', () => {
    awarenessViewModel.setName('John Doe');
    const state = store.getState().awareness;
    expect(state.name).toBe('John Doe');
  });

  it('should set the online status', () => {
    awarenessViewModel.setOnline(true);
    const state = store.getState().awareness;
    expect(state.isOnline).toBe(true);
  });
});

Now, let's implement the Awareness slice and AwarenessViewModel:
```

Awareness slice:

```
interface AwarenessState {
  cursorLocation: { x: number; y: number };
  name: string;
  isOnline: boolean;
}
```

```
const initialState: AwarenessState = {
  cursorLocation: { x: 0, y: 0 },
  name: '',
  isOnline: false,
};
```

```
const awarenessSlice = createSlice({
  name: 'awareness',
  initialState,
  reducers: {
    setCursorLocation: (state, action: PayloadAction<{ x: number; y: number }>) => {
      state.cursorLocation = action.payload;
    },
    setName: (state, action: PayloadAction) => {
      state.name = action.payload;
    },
    setOnline: (state, action: PayloadAction) => {
      state.isOnline = action.payload;
    },
  },
});
```

```
export const { setCursorLocation, setName, setOnline } = awarenessSlice.actions;
```

```
export default awarenessSlice.reducer;
AwarenessViewModel:
```

```
class AwarenessViewModel {
  private store: Store;

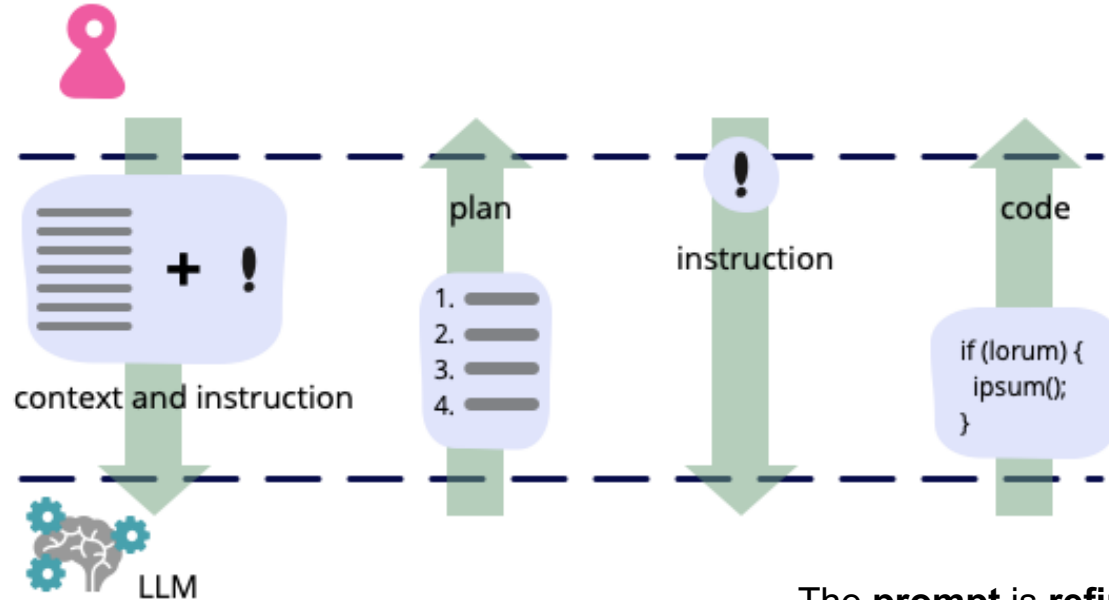
  constructor(store: Store) {
    this.store = store;
  }

  setCursorLocation(cursorLocation: { x: number; y: number }) {
    this.store.dispatch(setCursorLocation(cursorLocation));
  }

  setName(name: string) {
    this.store.dispatch(setName(name));
  }

  setOnline(isOnline: boolean) {
    this.store.dispatch(setOnline(isOnline));
  }
}
```

Note: In this implementation, I assumed that the AwarenessViewModel is used to set the values in the store. If it is also used to retrieve the values from the store, we can add corresponding methods to the AwarenessViewModel class



The **prompt** is **refined and crafted** to make ChatGPT generate the code **in the style you want**.

Sure, here's an example implementation of the tests for the AwarenessViewModel and the awareness slice without using redux-mock-store:

```
import { createAwarenessSlice, AwarenessState, updateAwareness } from './awarenessSlice';
import { AwarenessViewModel } from './AwarenessViewModel';

describe('AwarenessViewModel', () => {
  const mockDispatch = jest.fn();
  const initialState: AwarenessState = {
    users: {
      'user1': {
        name: 'User 1',
        cursorPosition: { x: 0, y: 0 },
        online: true,
      },
      'user2': {
        name: 'User 2',
        cursorPosition: { x: 100, y: 100 },
        online: true,
      },
    },
  };
  let viewModel: AwarenessViewModel;

  beforeEach(() => {
    viewModel = new AwarenessViewModel(initialState, mockDispatch);
  });

  describe('getUserNames', () => {
    it('returns the list of user names', () => {
      const names = viewModel.getUserNames();
      expect(names).toEqual(['User 1', 'User 2']);
    });
  });

  describe('getCursorPosition', () => {
    it('returns the cursor position for a given user ID', () => {
      const cursorPosition = viewModel.getCursorPosition('user1');
      expect(cursorPosition).toEqual({ x: 0, y: 0 });
    });

    it('returns undefined for an unknown user ID', () => {
      const cursorPosition = viewModel.getCursorPosition('unknown');
      expect(cursorPosition).toBeUndefined();
    });
  });

  describe('updateCursorPosition', () => {
    it('dispatches the updateAwareness action with the new cursor position', () => {
      viewModel.updateCursorPosition('user1', { x: 50, y: 50 });
      expect(mockDispatch).toHaveBeenCalledWith(updateAwareness({
        user: 'user1',
        cursorPosition: { x: 50, y: 50 },
      }));
    });
  });
});
```

And here's an example implementation of the awarenessSlice without using redux-mock-store:

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';

export type CursorPosition = { x: number, y: number };

export type UserAwareness = {
  name: string;
  cursorPosition: CursorPosition;
  online: boolean;
};

export type AwarenessState = {
  users: Record<string, UserAwareness>;
};

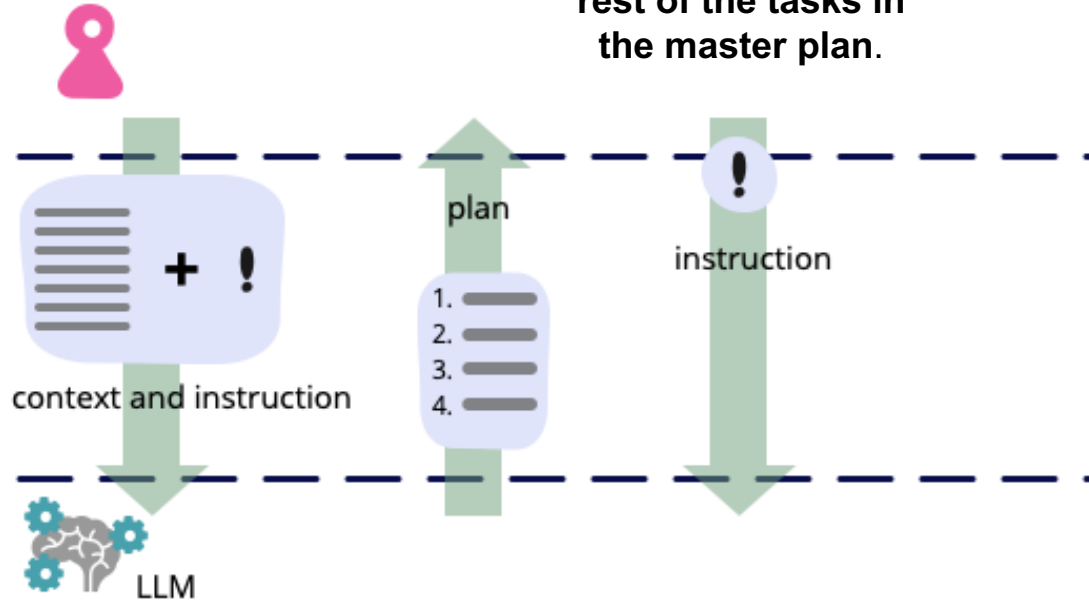
const initialState: AwarenessState = {
  users: {},
};

const awarenessSlice = createSlice({
  name: 'awareness',
  initialState,
  reducers: {
    updateAwareness: (state, action: PayloadAction<{ user: string, cursorPosition: CursorPosition }>) => {
      const { user, cursorPosition } = action.payload;
      state.users[user].cursorPosition = cursorPosition;
    },
    setOnline: (state, action: PayloadAction<{ user: string, online: boolean }>) => {
      const { user, online } = action.payload;
      state.users[user].online = online;
    },
  },
});

export const { updateAwareness, setOnline } = awarenessSlice.actions;
export const awarenessReducer = awarenessSlice.reducer;
```

Note that since we're not using redux-mock-store, we need to pass a mock dispatch function to the view model constructor, and in the test for updateCursorPosition we need to check that the correct action was dispatched using jest.fn().

Now you can **repeat**
the process for the
rest of the tasks in
the master plan.



LLM prompting for programming

An example from Martin Fowler and Xu Hao, Thoughtworks.

Source: <https://martinfowler.com/articles/2023-chatgpt-xu-hao.html>

- You can have problems with the **limited context window** (also called “**token limit**”).
 - ❖ ChatGPT may stop generating content: just say something like “**you are not finished**” or “**go on**”.
 - ❖ If the overall prompt is too big, then ChatGPT will simply return a fatal error and we have to **start a new conversation**.
 - ❖ ChatGPT may start forgetting and losing context. This is where the approach of getting a **master plan** and **breaking down the task** into separate elements helps.

Other examples: GPT Engineer

<https://github.com/AntonOsika/gpt-engineer>

```
(base) → gpt_engineer git:(main) x
```

Other examples: GPT PILOT

<https://github.com/Pythagora-io/gpt-pilot>

```
✕ -zsh
(env) zvonimirsabljic@Zvonimirs-MBP pilot %
```

Low-Code Programming using Traditional vs LLM Support

LCP and LLM-based LCP (Liu et al., 2024)

- Empirical study of both **traditional LCP** and **LLM-based LCP** analyzing developers' discussions on Stack Overflow (SO) over the past three years.
- The authors sought to answer:
 - ❖ What **application domains** do discussions on traditional LCP and LLM-based LCP focus on?
 - ❖ To which **software development tasks** do traditional LCP and LLM-based LCP contribute?
 - ❖ What are the **limitations** associated with traditional LCP and LLM-based LCP?

Application domains (Liu et al., 2024)

- **LLM-based LCP** encompasses a **wider array of application domains** compared to **traditional LCP**, which are **more narrowly focused on web development**.
- The main discussions about LLM-based LCP and traditional LCP revolve around **general usage issues**. The former predominantly involves **general programming challenges**, whereas the latter centers on **API integration** issues.
- Users predominantly focus on **similar application areas** regardless of the LLM-based LCP or traditional LCP, i.e., **web frontend, web backend and data management**.

Software development tasks (Liu et al., 2024)

- Discussion posts on usage of both LLM-based LCP and traditional LCP are **prevalent across the software development life cycle**, with a **particular emphasis on the implementation-related tasks**.
- **LLM-based LCP** discussions show a heightened **focus on deployment-related tasks** compared to traditional LCP, reflecting the different needs and concerns of users in this phase of the software development life cycle.

Limitations (Liu et al., 2024)

- Users of both LLM-based and traditional LCP express **concerns over reliability**, with a **higher degree** of apprehension observed in **LLM-based LCP due to the uncertainties associated with LLMs**.
- In order to **implement advanced functions**, LLM-based and traditional LCP **users need professional programming knowledge**, with **traditional LCP users needing more**.
- Both LLM-based LCP and traditional LCP **users face version-related issues**, with the former stemming from the **outdated data and hallucination of LLMs**, and the latter due to **conflicts during component upgrades**.

Limitations (Liu et al., 2024)

- **LLM-based LCP** appears to **surpass** traditional LCP's **API limitations**, enhancing **flexibility** and **adaptability**.
- **Traditional LCP** poses **data migration challenges** due to its **closed nature**, while **LLM-based LCP** offers **greater flexibility** in this regard.
- Compared to traditional LCP, **LLM-based LCP exhibits greater uncertainty**, necessitating improvements to meet user expectations more reliably.

Further Reading

- Yongkun Liu, Jiachi Chen, Tingting Bi, John Grundy, Yanlin Wang, Jianxing Yu, Ting Chen, Yutian Tang, Zibin Zheng (2024), ***“An Empirical Study on Low Code Programming using Traditional vs Large Language Model Support”***, Submitted to IEEE Transactions on Software Engineering, arxiv.org/abs/2402.01156.

Low-Code Development and Generative AI

Thiago Rocha Silva (trsi@mmmi.sdu.dk)
Associate Professor