生成AI Web開発体験1

bolt.new

https://bolt.new/

■ プロンプト

1. 計算機アプリを作ってください。説明は日本語で。
2. このデザインで計算機を作ってください。
3. 計算履歴を表示してください。
4. デプロイ（実際に使えるようにする）
5. このURLをQRコード表示する画面を追加してください。

■ エラーの場合

- 画面に表示される「Attempt fix」ボタンを実行
- エラーメッセージをコピー&ペースト

生成AI Web開発体験2

■ プロンプト

1. 三目並べゲーム・アプリを作ってください。説明は日本語で。
2. このデザインで三目並べゲームを作ってください。
3. 対戦相手が手を打つようにしてください。
4. 勝者には派手に花火を上げて、敗者は画面を暗くして残念なムードにして
5. 二人同時に別のブラウザで対戦する三目並べゲームにしてください。
6. デプロイ（実際に使えるようにする）
7. このURLをQRコード表示する画面を追加してください。
8. この三目並べゲームを作るためのプロンプトを教えてください。

Cursorでのチャットボット開発体験

■ プロンプト

1. 何を聞かれても、「はいよ」と回答するStreamlitチャットボットを作成してください。
2. 「へいへいほー」と言われた場合は「へいへいほー」と答える。「とんとんとん」の場合は「とんとんとん」と答える。それ以外は「与作～与作～」と答えるようにして。
3. Anthropic API 連携バージョン

■ Anthropic API 連携バージョンのプロンプト

```
# タスク: AWS Bedrock と Claude を使用した Streamlit チャットボットアプリケーションの開発

## 目的
AWS Bedrock上のClaude AIを利用した、インタラクティブなチャットボットアプリケーションを作成する。

## 要件
1. Streamlitを使用したウェブインターフェース
2. .envファイルからAWS認証情報を読み込む
3. モデルID「anthropic.claude-3-5-sonnet-20240620-v1:0」との対話

## 実装詳細
- ページタイトルは「Claude Chat Bot 🤖」
- .envファイルからAWS認証情報を読み込む（python-dotenvを使用）

## 使用ライブラリ（インストール済み）
- streamlit
- anthropic
- python-dotenv

## .env ファイルの構成
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_SESSION_TOKEN
- AWS_DEFAULT_REGION
```



```
## セキュリティ考慮事項
- 一時的な認証情報の安全な管理
- 認証情報の定期的なローテーション

```


■ サンプルコード

```
import streamlit as st
from anthropic import AnthropicBedrock
import os
from dotenv import load_dotenv

# 環境変数の読み込み
load_dotenv()

# ページ設定
st.set_page_config(
    page_title="Claude Chat Bot",
    page_icon="🤖"
)

# タイトルの設定
st.title("Claude Chat Bot 🤖")

# Bedrockクライアントの初期化
@st.cache_resource
def get_bedrock_client():
    return AnthropicBedrock(
        aws_access_key=os.getenv("AWS_ACCESS_KEY_ID"),
        aws_secret_key=os.getenv("AWS_SECRET_ACCESS_KEY"),
        aws_session_token=os.getenv("AWS_SESSION_TOKEN"),
        aws_region=os.getenv("AWS_REGION")
    )

# セッション状態の初期化
if "messages" not in st.session_state:
    st.session_state.messages = []

# チャット履歴の表示
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# ユーザー入力
if prompt := st.chat_input("メッセージを入力してください"):
    # ユーザーメッセージの表示
    st.chat_message("user").markdown(prompt)
```

```
    st.session_state.messages.append({"role": "user", "content": prompt})

    # Claudeからの応答を取得
    client = get_bedrock_client()
    with st.chat_message("assistant"):
        with st.spinner("考え中..."):
            response = client.messages.create(
                model="anthropic.claude-3-5-sonnet-20240620-v1:0",
                max_tokens=1000,
                messages=[{"role": m["role"], "content": m["content"]} for m in st.session_st
ate.messages]
            )
            assistant_response = response.content[0].text
            st.markdown(assistant_response)
            st.session_state.messages.append({"role": "assistant", "content": assistant_respo
nse})

```

■ .envファイル

```
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_SESSION_TOKEN=
AWS_DEFAULT_REGION=us-east-1
```


■ Rules for AI

```
always respond in japanese
```

あなたは高度な問題解決能力を持つAIアシスタントで、特にコード生成と修正に特化しています。以下の指示に従って、効率的かつ正確にタスクを遂行してください。

まず、ユーザーからの指示を確認します：

<ユーザー指示>
{{instructions}}
</ユーザー指示>

この指示を元に、以下のプロセスに従って作業を進めてください：

1. 指示の分析と計画
タスク分析をタグ内に記述してください：
- 指示の主要なポイントの簡潔な要約
- タスクの重要な要件と制約の特定
- ユーザーの指示から抽出した重要な情報のリスト
- 潜在的な課題とその解決策の列挙
- タスク実行のための具体的なステップの詳細な列挙
- それらのステップの最適な実行順序の決定
- 必要となる可能性のあるツールやリソースの考慮
- タスクの成功基準の明確化
- コード変更に関する具体的な指示の確認と、変更範囲の明確化
- 変更が必要な具体的なコードセクションの特定
- コード変更の潜在的な副作用の考慮
- 修正されたコードのユニットテストの計画

各項目について具体的に記述し、必要に応じて箇条書きや番号付きリストを使用して整理してください。この分析は後続のプロセス全体を導くものなので、十分に詳細かつ包括的に行ってください。長くなっても構いません。

2. タスクの実行
- 特定したステップを一つずつ実行し、各ステップの完了後に簡潔に進捗を報告してください。
- 実行中に問題や疑問が生じた場合は、即座に報告し、対応策を提案してください。
- コードの生成や変更を行う際は、ユーザーが明示的に指示した内容のみを実行し、その他の部分は変更しないでください。

3. 品質管理
- 各タスクの実行結果を迅速に検証してください。
- エラーや不整合を発見した場合は、直ちに修正アクションを実施してください。
- コマンドを実行する場合は、必ず標準出力を確認し、結果を報告してください。

4. 最終確認
- すべてのタスクが完了したら、成果物全体を評価してください。
- 当初の指示内容との整合性を確認し、必要に応じて調整を行ってください。
- コードの変更が指示された箇所のみに限定されていることを再確認してください。

5. 結果報告
以下のフォーマットで最終的な結果を報告してください：

```markdown
# 実行結果報告

## 概要
[全体の要約を簡潔に記述]

## 実行ステップ
1. [ステップ1の説明と結果]
2. [ステップ2の説明と結果]
...

## 最終成果物
[成果物の詳細や、該当する場合はリンクなど]

## 注意点・改善提案
- [気づいた点や改善提案があれば記述]
- [コード変更を行った箇所と理由を明確に説明]
```

重要な注意事項：
- 不明点がある場合は、作業開始前に必ず確認を取ってください。
- 重要な判断が必要な場合は、その都度報告し、承認を得てください。
- 予期せぬ問題が発生した場合は、即座に報告し、対応策を提案してください。
- コードの生成や変更は、ユーザーが明示的に指示した内容のみに厳密に限定してください。指示されていない部分は絶対に変更しないでください。

このプロセスに従って、効率的かつ正確にタスクを遂行してください。作業を開始する前に、必ずタスク分析タグ内で詳細な分析と計画を行ってください。
```
```

生成AI Web開発体験1

bolt.new

https://bolt.new/

■ プロンプト

1. 計算機アプリを作ってください。説明は日本語で。
2. このデザインで計算機を作ってください。
3. 計算履歴を表示してください。
4. デプロイ（実際に使えるようにする）
5. このURLをQRコード表示する画面を追加してください。

■ エラーの場合

- 画面に表示される「Attempt fix」ボタンを実行
- エラーメッセージをコピー&ペースト

生成AI Web開発体験2

■ プロンプト

1. 三目並べゲーム・アプリを作ってください。説明は日本語で。
2. このデザインで三目並べゲームを作ってください。
3. 対戦相手が手を打つようにしてください。
4. 勝者には派手に花火を上げて、敗者は画面を暗くして残念なムードにして
5. 二人同時に別のブラウザで対戦する三目並べゲームにしてください。
6. デプロイ（実際に使えるようにする）
7. このURLをQRコード表示する画面を追加してください。
8. この三目並べゲームを作るためのプロンプトを教えてください。

VSCode のインストール

1. 画面下部の検索欄で「PowerShell」と検索してください。
2. 「PowerShell」または「Windows PowerShell」を起動してください。
3. 下記の内容を貼り付けてください。もしここで「実行ポリシーの変更」という表示が出た場合は「Y」を入力してください。

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
scoop install git
scoop bucket add extras
scoop install nodejs vscode github
```

VScodeのGitHub Copilotのチャットに以下を記入して下さい
javascriptとHTMLとCSSで電卓アプリをつくってください

Cursorでのチャットボット開発体験

■ プロンプト

1. 何を聞かれても、「はいよ」と回答するStreamlitチャットボットを作成してください。
2. 「へいへいほー」と言われた場合は「へいへいほー」と答える。「とんとんとん」の場合は「とんとんとん」と答える。それ以外は「与作～与作～」と答えるようにして。
3. Anthropic API 連携バージョン

■ Anthropic API 連携バージョンのプロンプト

```
# タスク: AWS Bedrock と Claude を使用した Streamlit チャットボットアプリケーションの開発

## 目的
AWS Bedrock上のClaude AIを利用した、インタラクティブなチャットボットアプリケーションを作成する。

## 要件
1. Streamlitを使用したウェブインターフェース
2. .envファイルからAWS認証情報を読み込む
3. モデルID「anthropic.claude-3-5-sonnet-20240620-v1:0」との対話

## 実装詳細
- ページタイトルは「Claude Chat Bot 🤖」
- .envファイルからAWS認証情報を読み込む（python-dotenvを使用）

## 使用ライブラリ（インストール済み）
- streamlit
- anthropic
- python-dotenv

## .env ファイルの構成
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_SESSION_TOKEN
- AWS_DEFAULT_REGION
```



```
## セキュリティ考慮事項
- 一時的な認証情報の安全な管理
- 認証情報の定期的なローテーション

```

■ サンプルコード
```
import streamlit as st
from anthropic import AnthropicBedrock
import os
from dotenv import load_dotenv

# 環境変数の読み込み
load_dotenv()

# ページ設定
st.set_page_config(
    page_title="Claude Chat Bot",
    page_icon="🤖"
)

# タイトルの設定
st.title("Claude Chat Bot 🤖")

# Bedrockクライアントの初期化
@st.cache_resource
def get_bedrock_client():
    return AnthropicBedrock(
        aws_access_key=os.getenv("AWS_ACCESS_KEY_ID"),
        aws_secret_key=os.getenv("AWS_SECRET_ACCESS_KEY"),
        aws_session_token=os.getenv("AWS_SESSION_TOKEN"),
        aws_region=os.getenv("AWS_REGION")
    )

# セッション状態の初期化
if "messages" not in st.session_state:
    st.session_state.messages = []

# チャット履歴の表示
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# ユーザー入力
if prompt := st.chat_input("メッセージを入力してください"):
    # ユーザーメッセージの表示
    st.chat_message("user").markdown(prompt)
```



```
    st.session_state.messages.append({"role": "user", "content": prompt})

    # Claudeからの応答を取得
    client = get_bedrock_client()
    with st.chat_message("assistant"):
        with st.spinner("考え中..."):
            response = client.messages.create(
                model="anthropic.claude-3-5-sonnet-20240620-v1:0",
                max_tokens=1000,
                messages=[{"role": m["role"], "content": m["content"]} for m in st.session_st
ate.messages]
            )
            assistant_response = response.content[0].text
            st.markdown(assistant_response)
            st.session_state.messages.append({"role": "assistant", "content": assistant_respo
nse})
```

■ .envファイル
```
AWS_ACCESS_KEY_ID=ASIAUXFFUNS3MKNASGFN
AWS_SECRET_ACCESS_KEY=DoFqneuWn/4/uOtihje/kgwaHv4uO6ZXnKQW+i6b
AWS_SESSION_TOKEN=FwoGZXIvYXdzEJf/////////wEaDMRiNejzfghtSqOHAiKCAffOYnwOeFEjwvSMS7tkSXZAkoH
XpgQaEbzkKRLdK8JY8+qNeEjeVQ4g7KMJVyU2t587EoSOHVcgv/uOIAkyxOrBDaOiWV8dhmVp97FqVzy1GnCFFDfq1UxX
YxViRhJWecfSJMJ/TEJ9pHScQ7m8W8ZZPCKv5xMp60fCYwx8UFkVAS4o67TPugYyKJcQ6K7Lb2gv9Mh9xwh8RJ8p9xJeT
LngxjMcInY8QrKyyGyYjuCWHyw=
AWS_DEFAULT_REGION=us-east-1
```


■ Rules for AI
```
always respond in japanese
あなたは高度な問題解決能力を持つAIアシスタントで、特にコード生成と修正に特化しています。以下
の指示に従って、効率的かつ正確にタスクを遂行してください。

まず、ユーザーからの指示を確認します：

<ユーザー指示>
{{instructions}}
</ユーザー指示>

この指示を元に、以下のプロセスに従って作業を進めてください：

1. 指示の分析と計画
タスク分析をタグ内に記述してください：
- 指示の主要なポイントの簡潔な要約
- タスクの重要な要件と制約の特定
- ユーザーの指示から抽出した重要な情報のリスト
- 潜在的な課題とその解決策の列挙
- タスク実行のための具体的なステップの詳細な列挙
- それらのステップの最適な実行順序の決定
- 必要となる可能性のあるツールやリソースの考慮
- タスクの成功基準の明確化
- コード変更に関する具体的な指示の確認と、変更範囲の明確化
- 変更が必要な具体的なコードセクションの特定
- コード変更の潜在的な副作用の考慮
- 修正されたコードのユニットテストの計画

各項目について具体的に記述し、必要に応じて箇条書きや番号付きリストを使用して整理してくださ
い。この分析は後続のプロセス全体を導くものなので、十分に詳細かつ包括的に行ってください。長く
なっても構いません。

2. タスクの実行
- 特定したステップを一つずつ実行し、各ステップの完了後に簡潔に進捗を報告してください。
- 実行中に問題や疑問が生じた場合は、即座に報告し、対応策を提案してください。
- コードの生成や変更を行う際は、ユーザーが明示的に指示した内容のみを実行し、その他の部分は変
更しないでください。

3. 品質管理
- 各タスクの実行結果を迅速に検証してください。
- エラーや不整合を発見した場合は、直ちに修正アクションを実施してください。
- コマンドを実行する場合は、必ず標準出力を確認し、結果を報告してください。
```



```
4. 最終確認
- すべてのタスクが完了したら、成果物全体を評価してください。
- 当初の指示内容との整合性を確認し、必要に応じて調整を行ってください。
- コードの変更が指示された箇所のみに限定されていることを再確認してください。

5. 結果報告
以下のフォーマットで最終的な結果を報告してください：

```markdown
# 実行結果報告

## 概要
[全体の要約を簡潔に記述]

## 実行ステップ
1. [ステップ1の説明と結果]
2. [ステップ2の説明と結果]
...

## 最終成果物
[成果物の詳細や、該当する場合はリンクなど]

## 注意点・改善提案
- [気づいた点や改善提案があれば記述]
- [コード変更を行った箇所と理由を明確に説明]
```

重要な注意事項：
- 不明点がある場合は、作業開始前に必ず確認を取ってください。
- 重要な判断が必要な場合は、その都度報告し、承認を得てください。
- 予期せぬ問題が発生した場合は、即座に報告し、対応策を提案してください。
- コードの生成や変更は、ユーザーが明示的に指示した内容のみに厳密に限定してください。指示されて
いない部分は絶対に変更しないでください。

このプロセスに従って、効率的かつ正確にタスクを遂行してください。作業を開始する前に、必ずタス
ク分析タグ内で詳細な分析と計画を行ってください。
```