

Pythonによるブロック チェーン開発入門

～NFTチケットサービスを作ってみよう～

ステップバイステップで実践

株式会社SECURE4D/ Who am I

- システムインテグレータにてネットワークやサーバ構築などに従事し、その後、セキュリティ製品の開発事業に10年以上携わる
- セキュリティオペレーションセンターの立上げに参画
- OWASP Sendaiのメンバー
 - Meet Up:脆弱性放置の危険性
 - 「USB無線マウスからパソコンへ侵入」など
- 株式会社SECURE4Dを設立
- 近年はブロックチェーン界隈の開発にも携わる
- 主な使用言語: Python



Yosuke Sato (佐藤 陽亮)

OWASP: アメリカメリーランド州を本部とするWebアプリケーションセキュリティに関する情報や文書、知識の共有、普及を目的としたオープンで非営利のオープンソース・ソフトウェアコミュニティであるOWASP Foundationの **仙台支部** です。

今回の演習の内容

- ブロックチェーンを使ったチケット配布システム構築の基礎を学びます。
- ブロックチェーンの基本的な仕組みと役割を確認します。
- チケットの発行・配布・利用確認の流れを、API開発を通して体験します。
- 認証やデータ連携など、実運用に必要な周辺技術の基本も学びます。

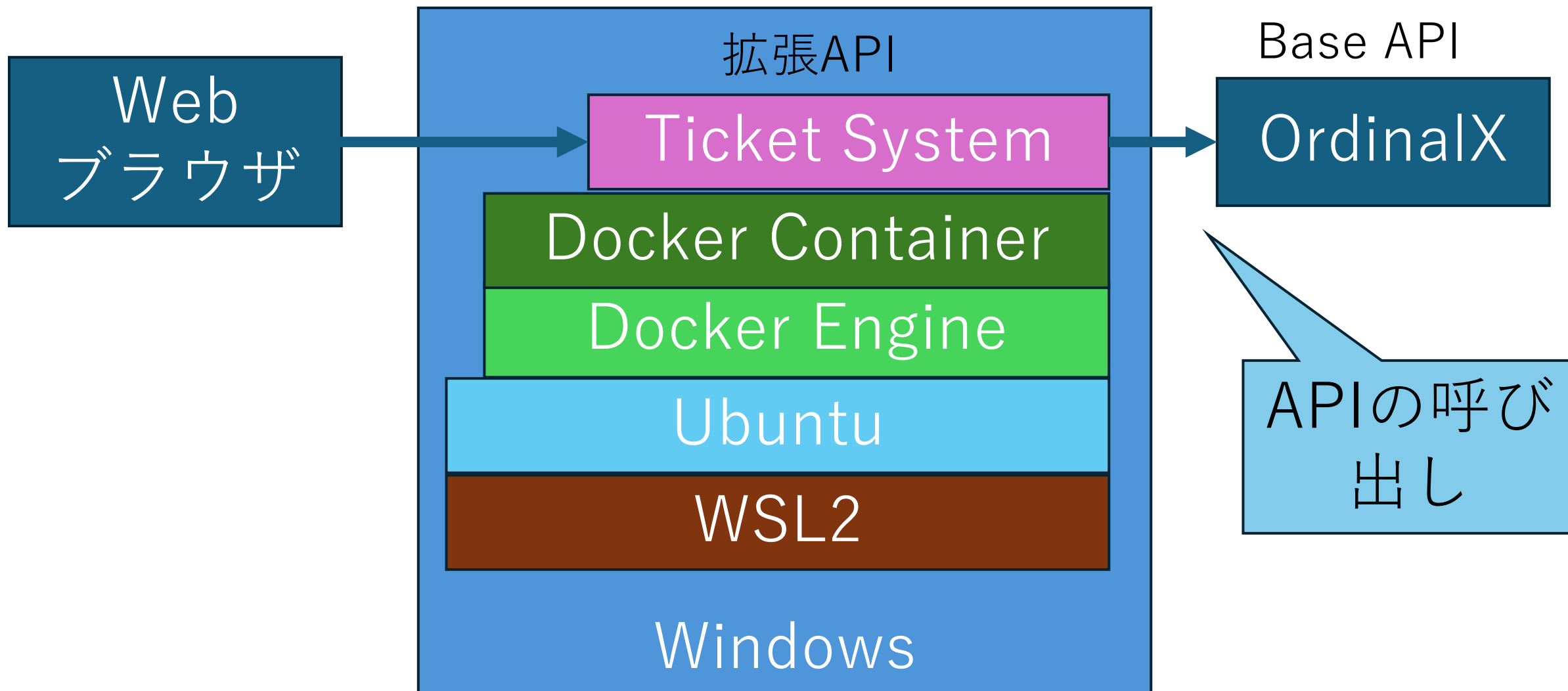
演習の進め方

- 「説明」→「手を動かす」→「動作確認」という流れを繰り返しながら、演習を進めていきます
- 使用するコマンドは配布するgitの documents フォルダのファイルにまとめてありますので参考にしながら実行していきます。
- - Stepごとの成功条件を満たしたら次へ
※厳密ではなくてOKです。
- まずはブラウザで以下を開いてください。

<https://github.com/voyager1708/ticketsystem>

全体像のお話①

<http://127.0.0.1:8001/>



開発環境の前提 (WSL/Mac/IDE)

- - WindowsはWSL内で作業用フォルダを使います
 - - MacはホストOS上の作業フォルダを利用します
 - - Ubuntu内で `git clone` してプロジェクトを取得します
 - - IDEはWindsurfまたはCursorを使用します
 - - 受講前準備: スムーズに進行できるようWSL(Ubuntu)とDockerを事前に導入してください
-
- 注釈: WSLはWindows上でLinuxを動かす仕組み
 - 注釈: IDEは開発機能をまとめた統合開発環境
 - 注釈: 今回Windsurf/Cursorは無料枠で利用します
 - 注釈: AIへの質問では無料枠消費に注意
 - 注釈: 無料枠でのクレジット入力は不要です

参照md: `documents/001\_setup\_v1\_start.md`

これからのステップ

- 環境構築

①Ubuntuのインストール

未

WIN

②Ubuntu起動

未

WIN

③管理者権限

未

WIN

④Dockerの導入

未

WIN

MAC

⑤プロジェクト取得（GIT）と起動

未

WIN

MAC

⑥IDE(統合開発環境) インストール

未

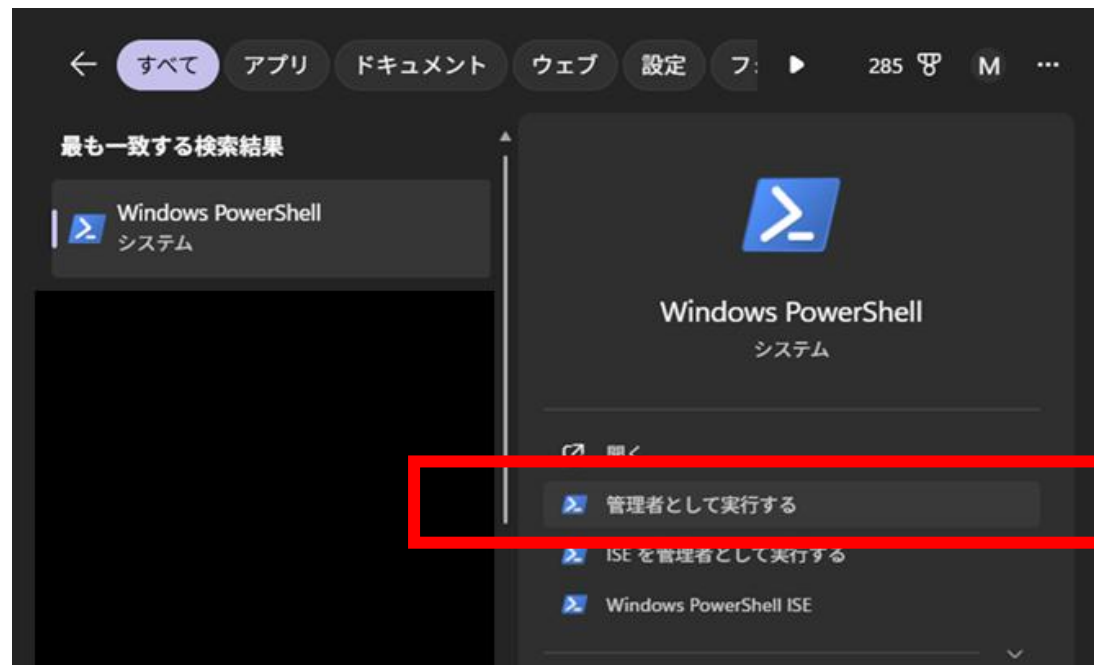
WIN

MAC

WSLが無効の場合 Linux 用 Windows サブシステムを有効にする

Power Shell を管理者として開き、以下を実行します。

`dism /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all`



```
PS C:\windows\system32> dism /online /enable-feature  
/featurename:Microsoft-Windows-Subsystem-Linux /all
```

展開イメージのサービスと管理ツール
バージョン: 10.0.26100.5074

イメージのバージョン: 10.0.26200.8037

機能を有効にしています

[=====100.0%=====]
=====]

操作は正常に完了しました。

Windows を再起動してこの操作を完了してください。
今すぐコンピューターを再起動しますか? (Y/N)

(参考)

<https://learn.microsoft.com/ja-jp/windows/wsl/install-manual>

①Ubuntuインストール

WIN

演習環境は Ubuntu OS上 で構築します。新たにPowerShellを開き、以下のコマンドでUbuntuをインストールします。 バージョンは 24.04を指定します。

```
PS C:¥Users¥ms> wsl.exe --install -d Ubuntu-24.04
```

一般ユーザ
権限



ダウンロード中: Ubuntu 24.04 LTS

インストール中: Ubuntu 24.04 LTS

ディストリビューションが正常にインストールされました。'wsl.exe -d Ubuntu-24.04' を使用して起動できます

②Ubuntu起動

WIN

以下のコマンドでUbuntuを起動します。

```
PS C:¥Users¥ms> wsl.exe -d Ubuntu-24.04
```

一般ユーザ
権限

アカウント名（任意アカウント名）の入力 と パスワードの設定をします。

```
Provisioning the new WSL instance Ubuntu-24.04
```

```
This might take a while...
```

```
Create a default Unix user account: XX
```

```
New password:
```

```
Retype new password:
```

```
passwd: password updated successfully
```

③管理者権限

WIN

以下はds9 というアカウントで作成した一般ユーザの**Ubuntuのプロンプト**です。

```
ds9@snapdragon:/mnt/c/Users/ms$
```



sudo -s というコマンドで**管理者権限**に昇格します。

```
sudo -s
```

ハンズオンでは管理者権限のまま使います



先程のパスワードを入力し管理者になりました。**プロンプトが # になっています。**

```
[sudo] password for ds9:  
root@snapdragon:/mnt/c/Users/ms#
```

④Dockerの導入

演習環境はDocker CEで管理します。

WIN

001_Setup_v1_start.mdの「Docker が未インストールの場合」の以下の青枠の部分をコピーでプロンプトに入力しDockerをインストールします。

⑤ Dockerの導入

以下の手順で進んでください。

Docker が未インストールの場合（方法1: 公式リポジトリで入れる・推奨）

`docker` コマンドが見つからない場合は、まず Docker をインストールします。**`docker compose` (Compos

1. **公式リポジトリの追加とインストール** (WSL2 / Ubuntu 系)

```
```bash
sudo apt update
sudo apt install -y ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/keyrings/docker.asc > /dev/
null
sudo chmod a+r /etc/apt/keyrings/docker.asc
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.
docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /
dev/null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

gitの  
documentsフォルダに  
あります。

## ④Dockerの導入

WIN

インストールできたことを確認します。docker -v コマンド

```
docker -v
```

```
Docker version 29.3.0, build 5927d80
```

(一般ユーザで作業する場合) 現在ユーザーを `docker` グループに追加します

```
sudo groupadd -f docker
sudo usermod -aG docker "$USER"
```

その後、WSLは再ログインだけで反映されないことがあるので、反映するには:

```
wsl.exe --terminate Ubuntu-24.04
wsl.exe -d Ubuntu-24.04
```

## ④Dockerの導入

WIN

### Dockerサービスを起動します

```
sudo service docker start
```

\* Starting Docker: docker

[ OK ]

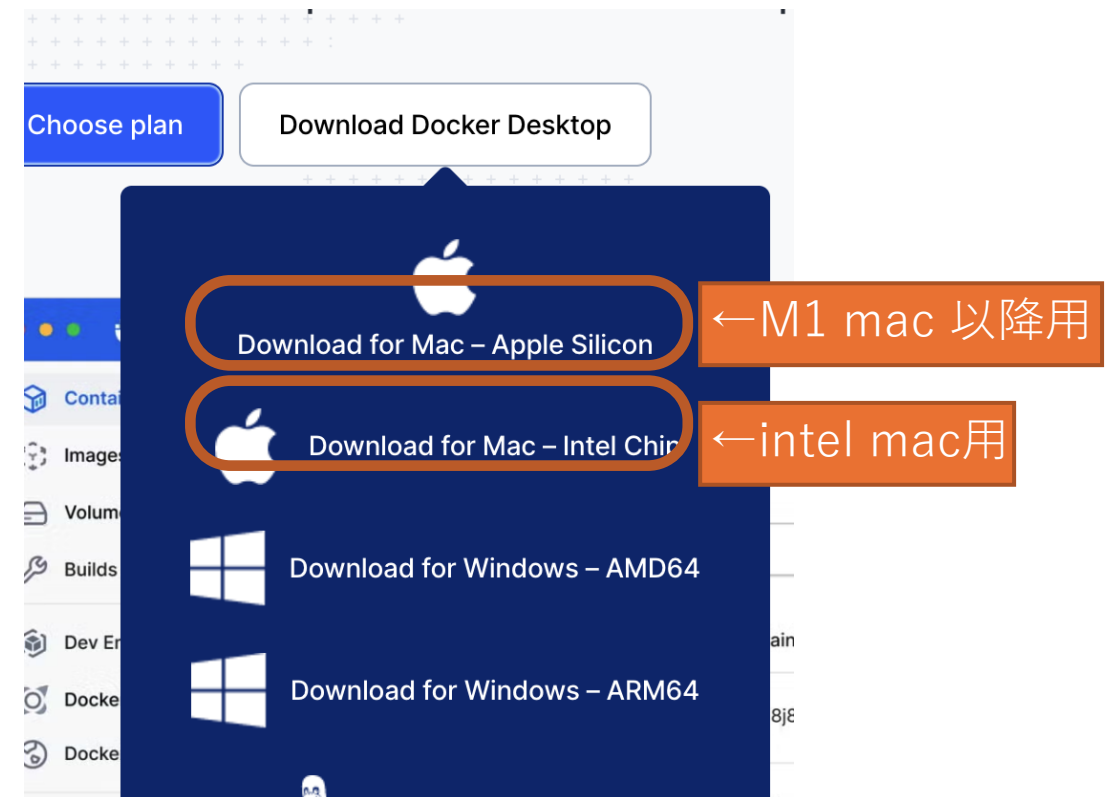
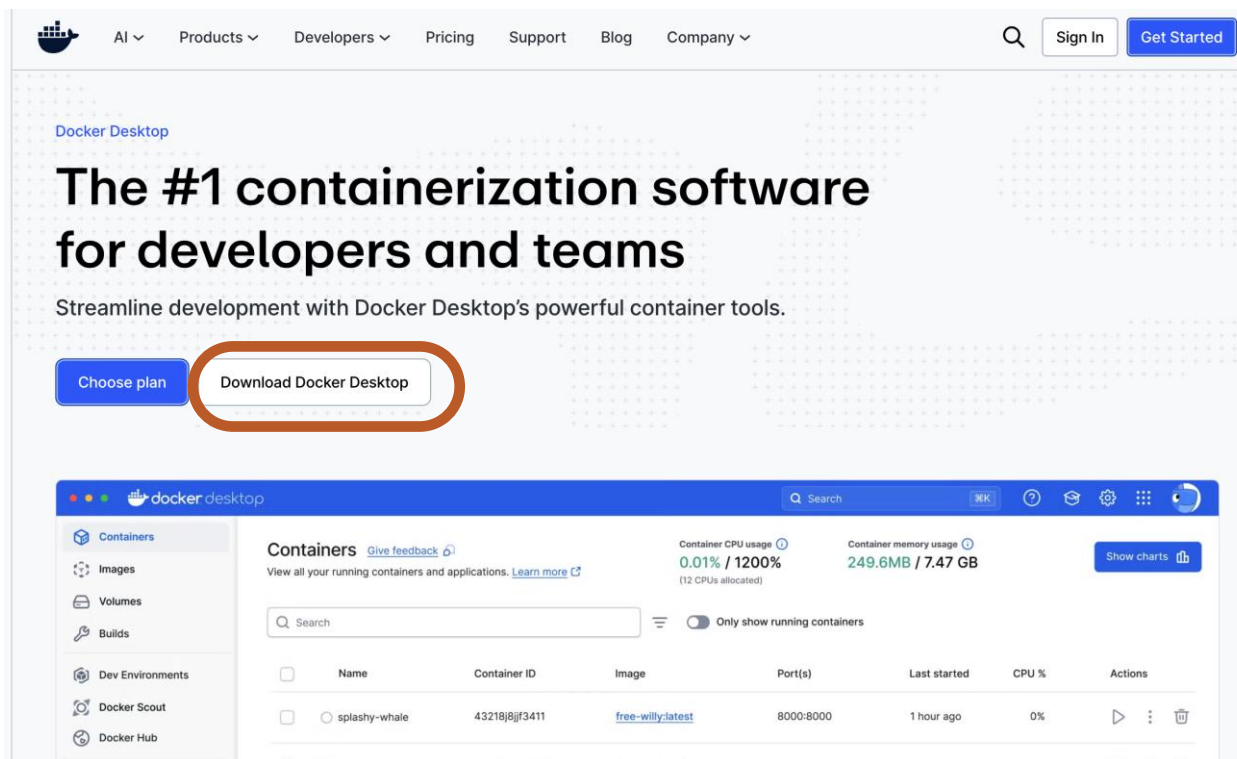
# ④Dockerの導入

Mac

## Docker Desktop のダウンロードとインストール

以下から Docker Desktop for Mac をダウンロードし、インストールしてください。

<https://www.docker.com/products/docker-desktop/>



# ④Dockerの導入

Mac

## Docker Desktop のダウンロードとインストール

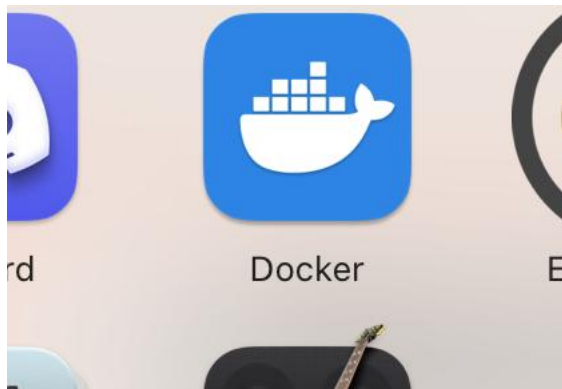
ダウンロードしたdocker fileを開く

アプリケーションフォルダーにドラック



Docker.dmg  
651.2 MB

アプリケーションフォルダーから開く



# ④Dockerの導入(トラブルシューティング)

Mac

最新版のmacOSを使用していない場合、Docker desktopを起動しようとするときエラーが表示されることがあります。

その場合は、AI（Claude、ChatGPT、Gemini など）にご自身のMacのOSバージョンを伝え、対応しているDockerの公式ダウンロード直リンクを案内してもらってください。

13.7.8だとどのドッカーが使えるのか

結論から言うと、

**\*\*macOS 13.7.8 (Ventura) で使える Docker Desktop は「4.25～4.43あたりまで」\*\*です。**

それより新しいバージョンは **macOS 14以上が必須**になります。 [Docker Document...](#) +1

## macOS 13.7.8 (Ventura) で使えるDocker

実務的には次の範囲です。

Docker Desktop	macOS対応
4.25.x	macOS 12 / 13 / 14
4.26.x	macOS 12 / 13 / 14
4.27～4.43	Venturaでも動作
4.44以降	Sonoma (macOS 14) 以上が推奨

例

- Docker Desktop 4.25.0
- Docker Desktop 4.26.1
- Docker Desktop 4.30.0

このあたりが Ventura で安定して動きます。 [GitHub](#)

この直リンクがあるか Docker Desktop 4.30

はい。Docker Desktop 4.30 の公式 CDN 直リンクは存在します。

## Docker Desktop 4.30 直リンク

### Intel Mac

<https://desktop.docker.com/mac/main/amd64/149282/Docker.dmg>



### Apple Silicon (M1 / M2 / M3)

<https://desktop.docker.com/mac/main/arm64/149282/Docker.dmg>



これは Docker の公式 CDN ( [desktop.docker.com](https://desktop.docker.com) ) の **build 149282** で、

**\*\*Docker Desktop 4.30.0 (2024-05-06 リリース) \*\***です。 [Gist](#) +1

# ④Dockerの導入

Mac

ターミナルでインストール確認

```
docker --version
docker compose version
```

## ⑤プロジェクト取得と起動

WIN

MAC

作業ディレクトリへ移動: `cd ~`  
GITコマンドでプロジェクトを取得します。

```
cd ~
```

```
git clone https://github.com/voyager1708/ticketsystem.git ticketsystem-start
```

```
Cloning into ticketsystem-start...
```

```
remote: Enumerating objects: 564, done.
(中略)
```

```
Resolving deltas: 100% (208/208), done.
```

```
ds9@snapdragon:~/ticketsystem-start$
```

のようになっていればOK

```
cd ticketsystem-start
```

```
git checkout handson/v1-start
```

```
root@snapdragon:/mnt/c/Users/ms/ticketsystem-start
はNGです
```

## ⑤プロジェクト取得と起動

WIN

MAC

### 1. `.env` の用意

```
cp .env.example .env
```

で `BASE_API_URL` を指定しています

`BASE_API_URL=https://bsvapi01.cds.tohoku.ac.jp`

`more .env` コマンドでファイルの中身を確認

# ⑤プロジェクト取得と起動

WIN

MAC

## 2.コンテナのビルド・起動

```
docker compose -f docker-compose.yml -f docker-compose.dev.yml
build --no-cache ticket_web
```

```
./bin/start-dev
```

内部では以下を実行：

```
docker compose -f docker-compose.yml -f docker-compose.dev.yml up -d --force-recreate
```

dockerって何？ とか  
分からないことは AI に  
聞いてみよう

# メモ①：一般ユーザで起動させてしまった場合

## **./venv のホスト側権限の問題が起きます。**

/home/app/venv は ./venv に bind mount されるので、ホストの venv が 別の所有となり app ユーザー (UID 1000) が書けず、**Permission denied** になり起動できません。

1) 起動しているか確認 (**Up**なら起動、**Exit**なら終了しています)

### **sudo docker ps -a**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
432038e069e1	ticketssystem-start-ticket_web	"/entrypoint.sh"	23 seconds ago	<b>Up 11 seconds</b>	0.0.0.0:8001-8001/tcp, [::]:8001->8001/tcp	ticketssystem-start-

2) 既存コンテナ停止

### **cd ~/ticketssystem-start**

### **sudo docker compose -f docker-compose.yml -f docker-compose.dev.yml down**

3) venv の権限を Docker のユーザー (ID) にする

**sudo chown -R 1000:1000 ./venv**

**sudo chmod -R u+rwX ./venv 2>/dev/null || true**

**sudo ./bin/start-dev**

# メモ②：ログの確認方法

何が起きている確認するにはログをみます。

1) CONTAINER ID を確認します。

**sudo docker ps -a**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
432038e069e1	ticketssystem-start-ticket_web	"/entrypoint.sh"	...	

2) ログの表示 CONTAINER ID を以下のように指定します。

**docker logs 432038e069e1**

## ⑤プロジェクト取得と起動

WIN

MAC

### 拡張API動作確認

```
curl -i -sS http://127.0.0.1:8001/healthz | head -5
```

```
HTTP/1.1 200 OK
```

```
Server: gunicorn
```

```
Date: Fri, 13 Mar 2026 06:18:00 GMT
```

```
Connection: close
```

```
Content-Type: text/html; charset=utf-8
```

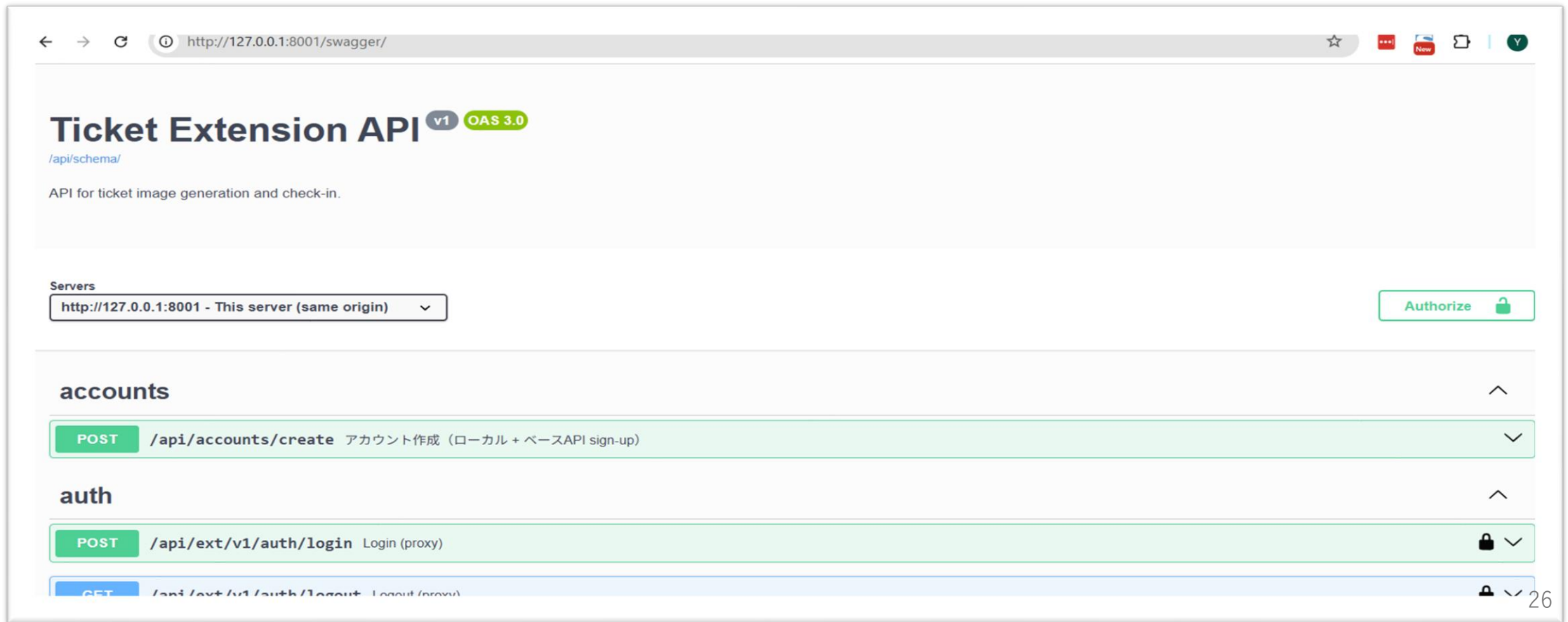
この1行をコピー  
で実行

ステータス200  
リクエストが正常に処理され成功

ブラウザで `http://127.0.0.1:8001/swagger/` を開く

## ⑤プロジェクト取得と起動

- - Swaggerトップ画面を確認します。
- - 画面のURLが `http://127.0.0.1:8001/swagger` か確認します



## ⑥ IDE(統合開発環境) インストール

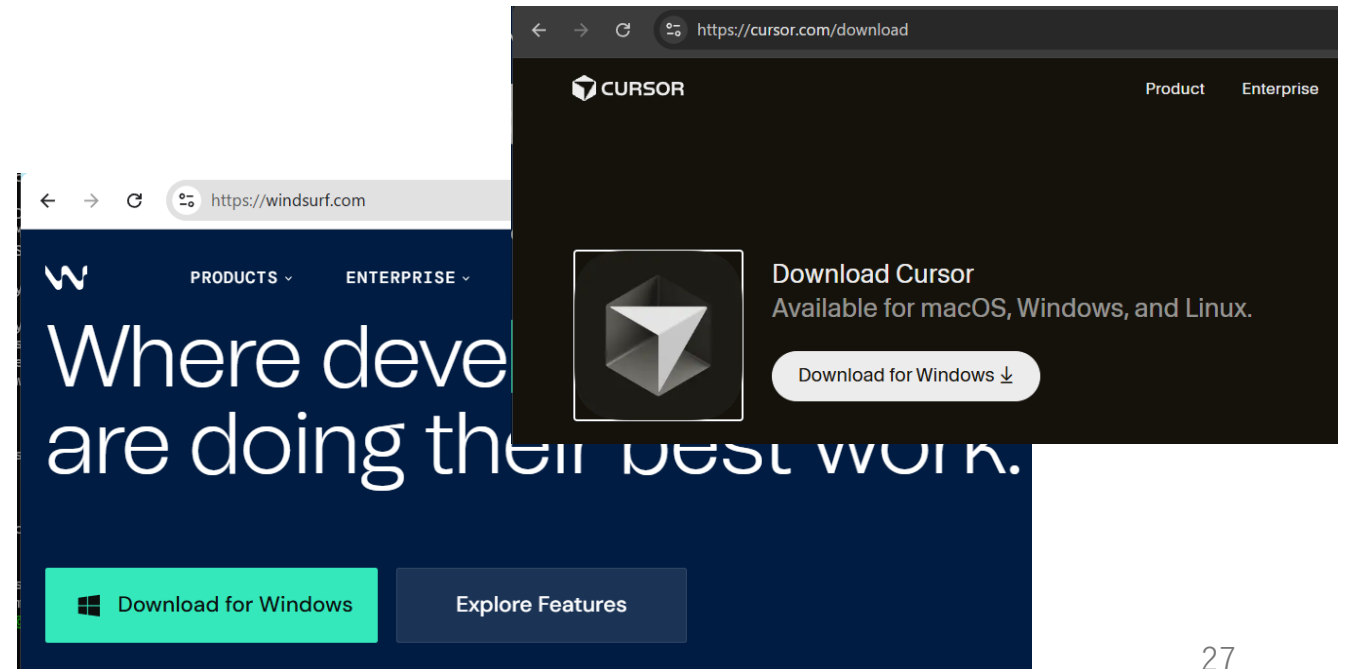
- Integrated Development Environment (統合開発環境) の略で、プログラムを開発するためのツールを一体化したソフトウェアのことです

今回はwindsurf（または cursor）を使います。※お好みで

<https://windsurf.com/>

<https://cursor.com/>

このページからダウンロード  
インストールしてください。

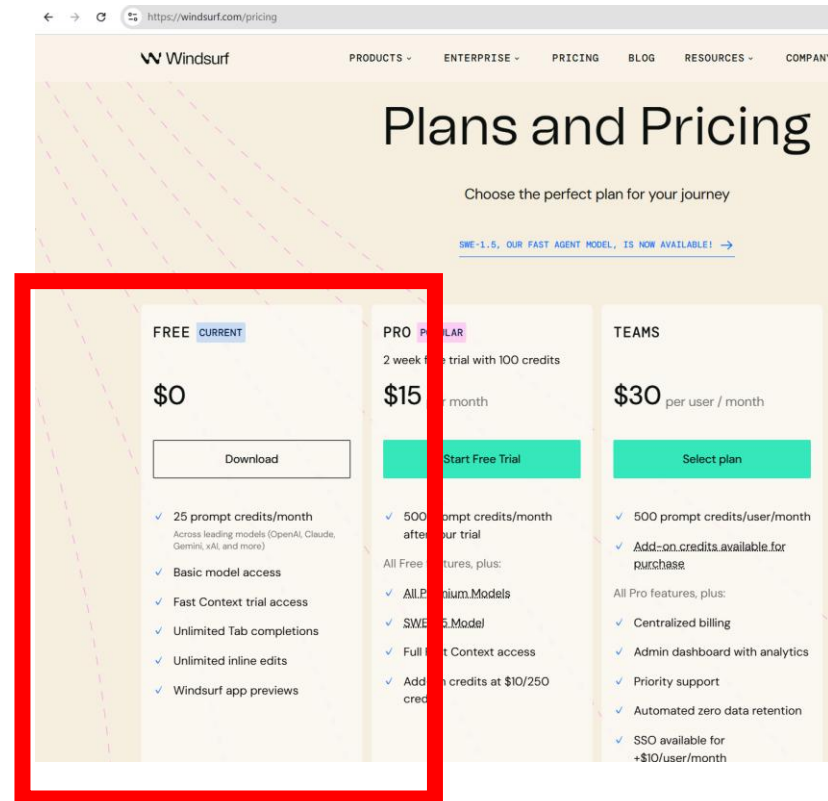
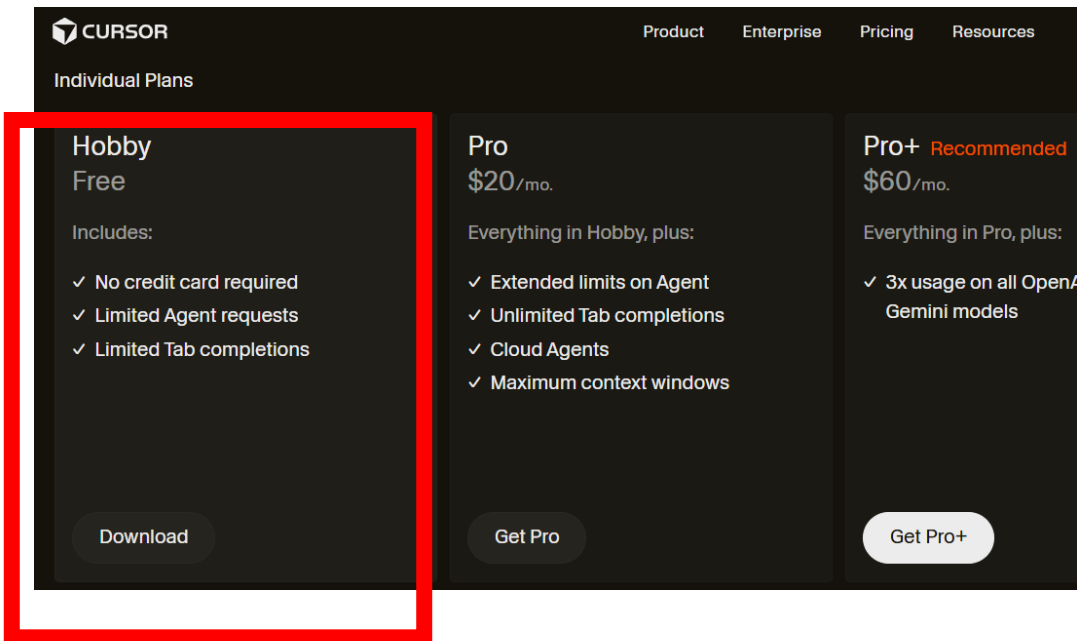


## ⑥ IDE(統合開発環境) インストール

Pricing のページからもダウンロードできます。

※お持ちのアカウントでログイン（またはサインアップ）

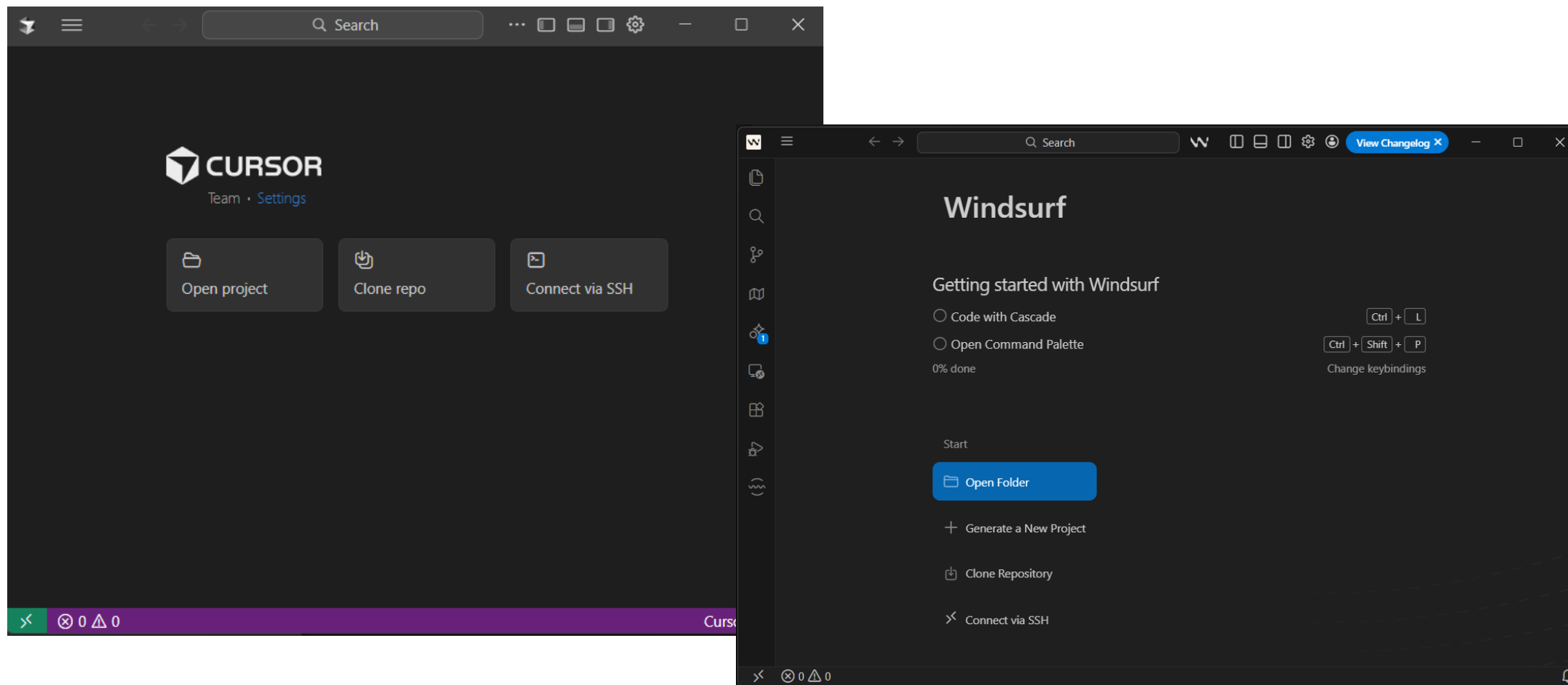
してダウンロードします。その後インストールに進んでください。



無料枠でのクレジット情報入力は不要です

# ⑥ IDE(統合開発環境) インストール (起動)

こんな感じの起動画面です。



# ここまでのステップ

- 環境構築

- |                    |   |
|--------------------|---|
| ①Ubuntuのインストール     | 済 |
| ②Ubuntu起動          | 済 |
| ③管理者権限             | 済 |
| ④Dockerの導入         | 済 |
| ⑤プロジェクト取得（GIT）     | 済 |
| ⑥IDE（統合開発環境）インストール | 済 |

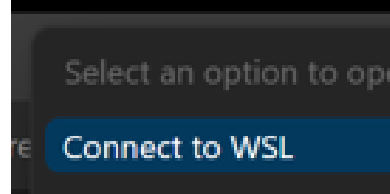
# IDEでプロジェクトフォルダを開く

WIN

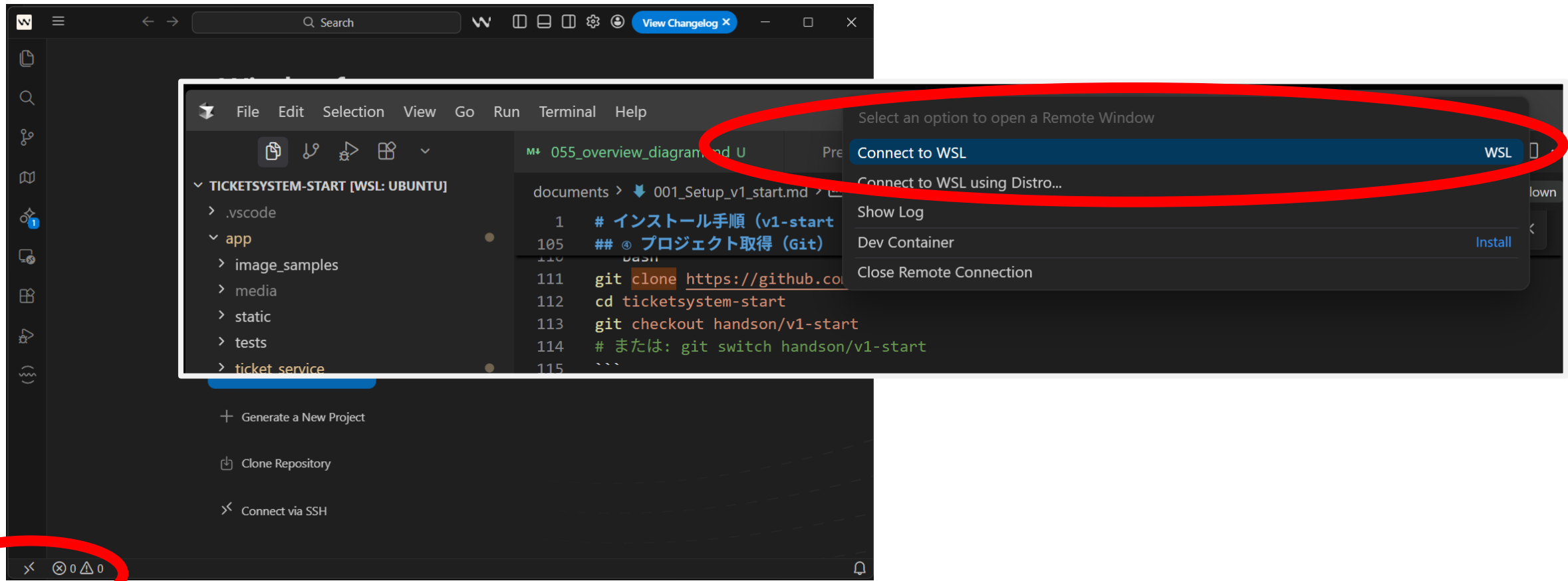
左下の



をクリックして



を選択します。

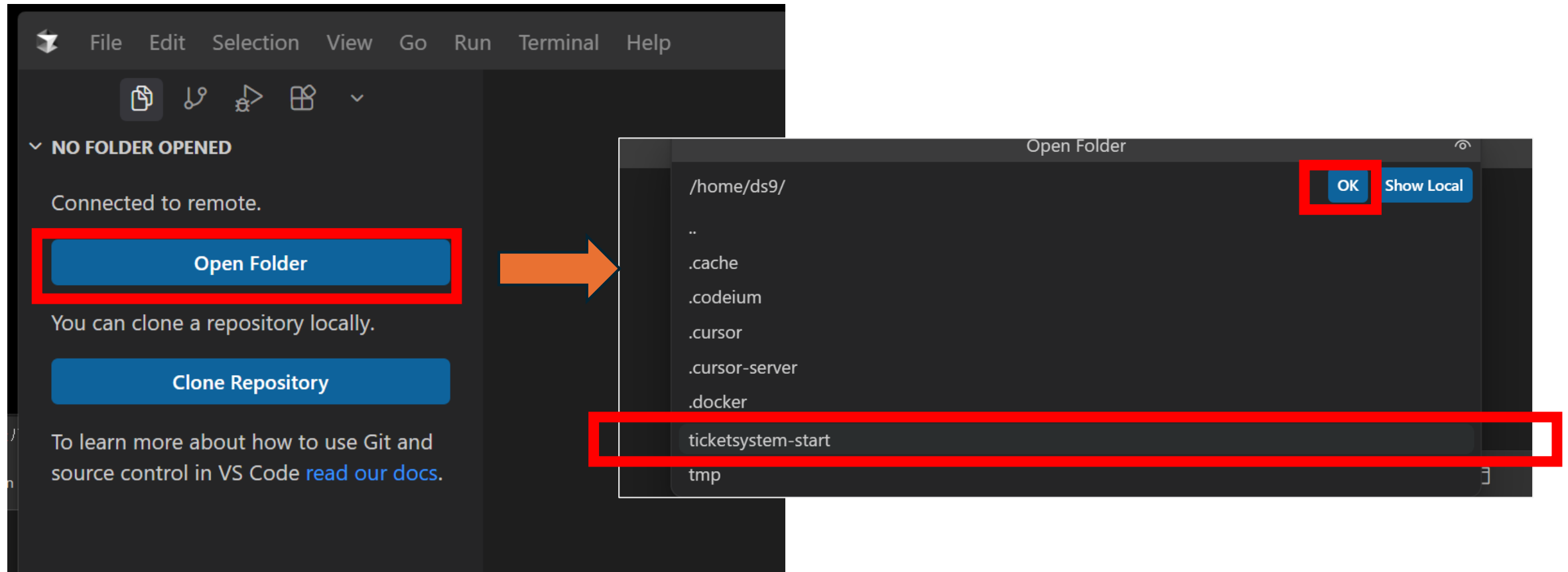


# IDEでプロジェクトフォルダを開く

WIN

MAC

Open Folderをクリック（またはファイルメニューからOpen Folderを選択）して、GITで取得したプロジェクトフォルダを開きます。



# IDEでプロジェクトフォルダを開く

以下、プロジェクトフォルダを開いた状態

ドキュメントフォルダ

AIプロンプト

信用してOK

Getting started with Windsurf

Do you trust the authors of the files in this folder?

Windsurf provides features that may automatically execute files in this folder.

If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See [our docs](#) to learn more.

~/ticketssystem-start [WSL: winds]

☒ Trust the authors of all files in the parent folder 'ms'

Yes, I trust the authors

No, I don't trust the authors

Trust folder and enable all features

Browse folder in restricted mode

# AIのモードとモデル選択

## Windsurf

Ask anything (Ctrl+L)

+ <> Code SWE-1.5

Codeを選択  
モデルはSWE1.5

# 全体像のお話

## Ticket System (構造)

```
ticketssystem-start/
├── app/
│ ├── ticket_service/ # 拡張API (チケット・チェックイン・報酬)
│ │ ├── views.py
│ │ ├── urls.py
│ │ ├── models.py
│ │ └── services/
│ │ ├── base_api_client.py
│ │ └── ticket_service.py
│ ├── ticket_system/ # Django プロジェクト設定
│ │ ├── settings.py
│ │ └── urls.py
│ ├── manage.py
│ ├── requirements.txt
│ └── Dockerfile
├── bin/
│ ├── start-dev # 開発用コンテナ起動
│ └── restart-dev
├── image_samples/
│ └── ticket_sample.html # チケットHTMLテンプレート例
├── .example
├── docker-compose.yml # Docker Compose 設定
└── docker-compose.dev.yml
```

今回はDjangoとい  
うフレームワークで  
構築

Dockerの設定

# urls.py や views.py を確認してみよう

```
urlpatterns = [
 path(
 'accounts/create',
 AccountCreateAPIView.as_view(),
 name='account-create',
),

```

ブラウザでのURLに対応して呼ばれる関数

```
class AccountCreateAPIView(APIView):
 """
 POST /api/accounts/create: ローカルにユーザーを作成し、
 同時にベースAPIの sign-up に転送する。
 先にベースAPIで登録し、成功したらローカルに保存。ベース
 APIが 400 の場合はローカルは作らない。
 """
 permission_classes = [AllowAny]
 authentication_classes = []
 parser_classes = [JSONParser]

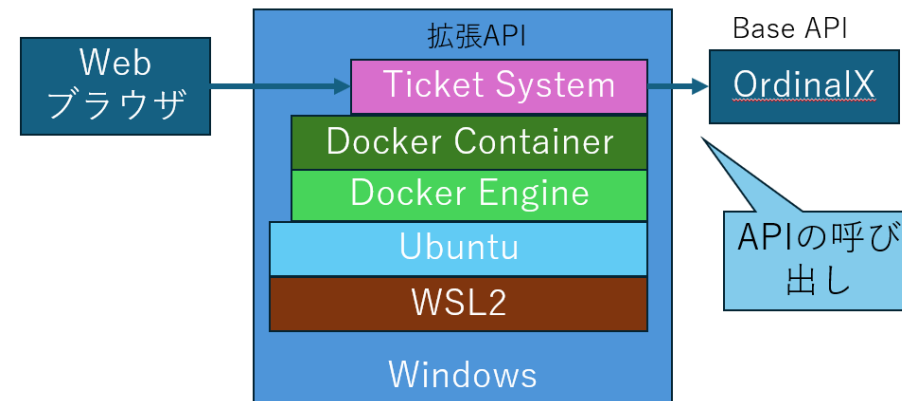
```

Swaggerの定義や実際の処理内容

# 全体アーキテクチャ

ちょっとおさらい

- - 構成:
  - 利用者操作 → 拡張API → Base API
- - 役割分担:
  - 拡張API: チケット作成とチェックイン機能を提供
  - Base API: アカウント管理とNFT基盤機能を提供
- - 認証/連携:
  - ユーザーは拡張APIへログインして各機能を実行します
  - 拡張APIはセッション付きでBase APIへ安全に中継します



# ここからのステップ

- 題材はチケットシステム

(アカウント作成、チケット生成、チェックイン/報酬送付)

※報酬はチケットの半券みたいなもの (記念)

Base APIの基礎理解

APIの型をつかむ

アカウント作成

ログイン

チケット生成

チェックイン

ここをAIを使って実装  
してみましょう

# Base APIの基礎理解

- - Base APIはNFTと認証を担う共通基盤です
- - 主な機能は認証、NFT発行、保有NFT取得です
- - 拡張APIはBase API機能を組み合わせて活用します
- - URL設定は `` の `BASE\_API\_URL` です
- - 読み込み場所は `app/ticket\_system/settings.py` です
- - 利用箇所は `services/base\_api\_client.py` です

URLは

<http://127.0.0.1:8001/swagger/>

# APIの型をつかむ

- - 未ログインで `GET /api/ext/v1/auth/user`
- - 期待: `401 Authentication credentials were not provided.`
- - 学ぶポイント:
  - - ステータスコード
  - - JSONレスポンス形
  - - 認証前提の有無

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** /api/ext/v1/auth/user
- Parameters:** No parameters
- Execute:** Button to execute the request
- Responses:** Section containing the request details and the response.
- Request URL:** http://127.0.0.1:8001/api/ext/v1/auth/user
- Server response:** A table showing the response details.

Code	Details
401	Error: Unauthorized

**Response body:**

```
{ "detail": "Authentication credentials were not provided."}
```

# アクター運用方針（今回は1アカウント）

- - 本来は主催者とユーザの2アクターで運用します
- - 今回演習は1つのアカウントで進めます
- - チケット発行時は主催者の立場で操作します
- - チェックイン時はユーザの立場で操作します
- - 役割を意識して操作目的を切り替えます

# アカウント作成

- - `POST /api/accounts/create`
- - 期待:
- - 成功: 201 (id, username)
- - 入力不正: 400
- - 役割:
- - ローカル保存 + Base API sign-up 転送

```
{
 "username": "ysato9",
 "email": "ysato9@secure4d.com",
 "password": "XXXXXX",
 "group": "string"
}
```

## accounts

**POST** /api/accounts/create

ベースAPIの sign-up に転送し、成功した

### Parameters

No parameters

**Request body** required

**Edit Value** | Schema

```
{
 "username": "ysato9",
 "email": "ysato9@secure4d.com",
 "password": "XXXXXX",
 "group": "string"
}
```

# アカウント作成

では各自、自分のアカウントを作成してください。

The screenshot shows a REST client interface for the `accounts` API. The selected endpoint is `POST /api/accounts/create` with the description "アカウント作成 (ローカル + ベースAPI sign-up)". Below the endpoint, there is a text block explaining that the user should be transferred to the base API sign-up, and upon success, an account should be created in the local environment. It specifies that `username`, `email`, and `password` are required, while `group` is optional for the base API. The interface includes sections for "Parameters" (showing "No parameters"), "Request body" (marked as "required" and set to `application/json`), and "Example Value" (showing a JSON schema for the request body).

**accounts**

**POST** `/api/accounts/create` アカウント作成 (ローカル + ベースAPI sign-up)

ベースAPIの sign-up に転送し、成功したらローカルにも Account を作成。username, email, password 必須。group は任意 (ベースAPI用)。

**Parameters** Try it out

No parameters

**Request body** required application/json

**Example Value** | Schema

```
{
 "username": "string",
 "email": "user@example.com",
 "password": "string",
 "group": "string"
}
```

Try it out を  
クリック

username,  
Email  
passwordを入力

※groupはそのまま  
OK

# ログイン

では、作成した自分のアカウントでログインしてください。

**auth**

**POST** `/api/ext/v1/auth/login` Login (proxy)

拡張API経由でベースAPIのログインを中継

**Parameters** Cancel Reset

No parameters

**Request body** required application/json

**Edit Value** | Schema

```
{
 "username": "ysato9",
 "password": "xxxxxxx"
}
```

# ログイン

では、作成した自分のアカウントでログインしてください。

**auth**

**POST** `/api/ext/v1/auth/login` Login (proxy)

拡張API経由でベースAPIのログインを中継

**Parameters** Cancel Reset

No parameters

**Request body** required application/json

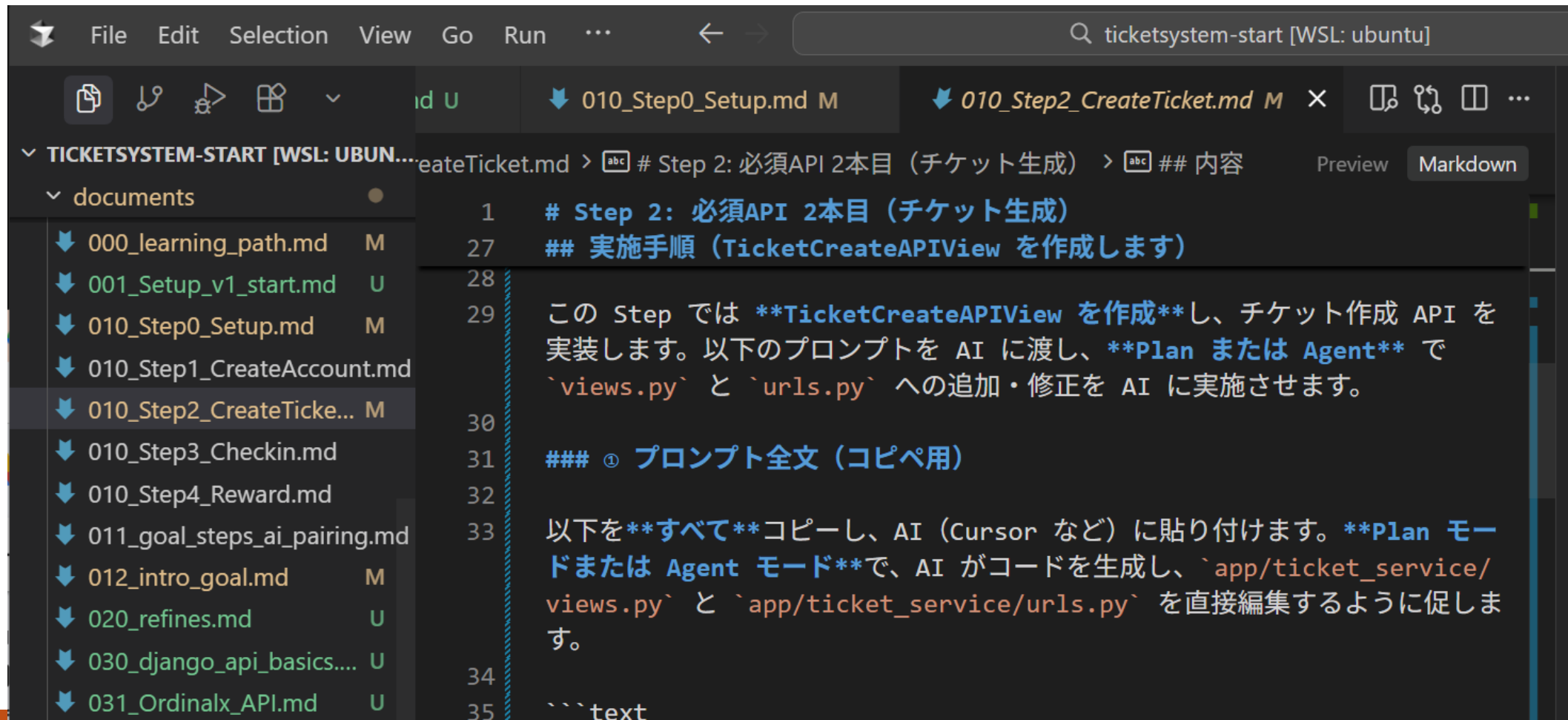
**Edit Value** | Schema

```
{
 "username": "ysato9",
 "password": "xxxxxxx"
}
```

# チケット生成

いよいよ、チケット作成にチャレンジします。

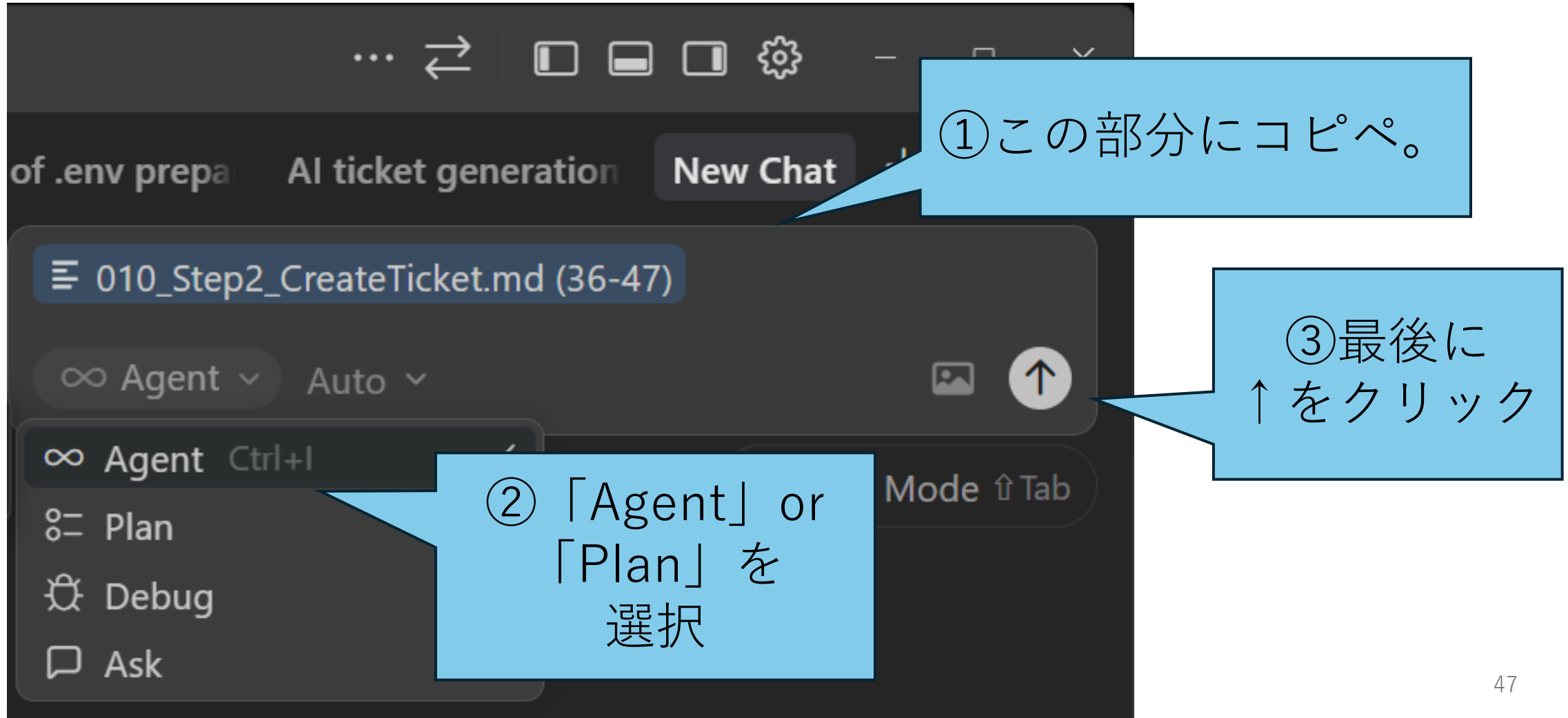
IDEの画面にて「010\_Step2\_CreateTicket.md」を見てください。



```
1 # Step 2: 必須API 2本目 (チケット生成)
27 ## 実施手順 (TicketCreateAPIView を作成します)
28
29 この Step では **TicketCreateAPIView を作成**し、チケット作成 API を
 実装します。以下のプロンプトを AI に渡し、**Plan または Agent** で
 `views.py` と `urls.py` への追加・修正を AI に実施させます。
30
31 ### ① プロンプト全文 (コピペ用)
32
33 以下を**すべて**コピーし、AI (Cursor など) に貼り付けます。**Plan モー
 ドまたは Agent モード**で、AI がコードを生成し、`app/ticket_service/
 views.py` と `app/ticket_service/urls.py` を直接編集するように促しま
 す。
34
35 ```text
```

# チケット生成

では実際に、AIのプロンプトに入力してチケット生成のプログラムを作ります。



# AIが作成したプログラムの確認

AIがプログラムを作成したら、**urls.py** や **views.py** を確認してみよう

ticketssystem-start/

├── app/

│ ├── ticket\_service/

│ │ ├── **views.py**

│ │ ├── **urls.py**

│ │ ├── models.py

│ │ └── services/

│ │ ├── base\_api\_client.py

# 拡張API (チケット・チェックイン・報酬)

# urls.py や views.py を確認してみよう

```
urlpatterns = [
 path(
 'accounts/create',
 AccountCreateAPIView.as_view(),
 name='account-create',
),

```

ブラウザでのURLに対応して呼ばれる関数

```
class AccountCreateAPIView(APIView):
 """
 POST /api/accounts/create: ローカルにユーザーを作成し、
 同時にベースAPIの sign-up に転送する。
 先にベースAPIで登録し、成功したらローカルに保存。ベース
 APIが 400 の場合はローカルは作らない。
 """
 permission_classes = [AllowAny]
 authentication_classes = []
 parser_classes = [JSONParser]

```

Swaggerの定義や実際の処理内容

# Dockerを再起動して反映させます

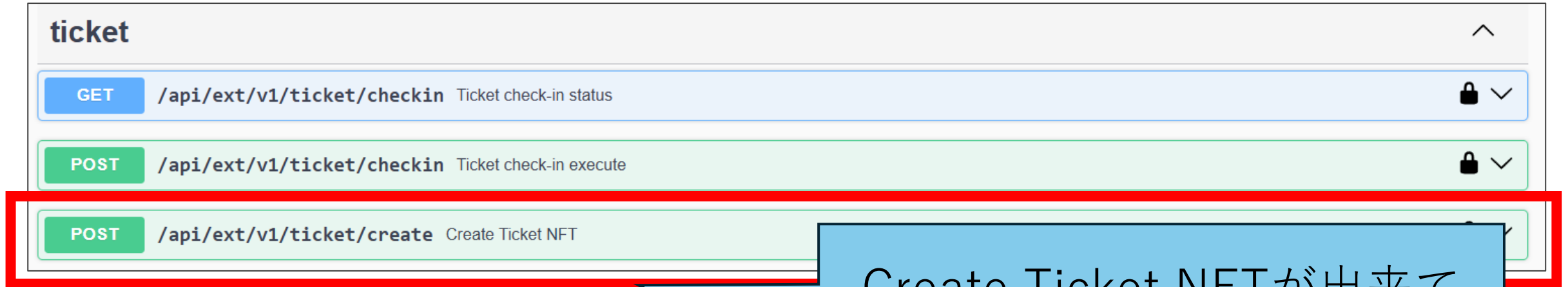
AIがプログラムを作成したら、  
Ubuntuのコンソール画面で、以下のコマンドを入力しDockerを再起動し反映させます。

```
root@snapdragon:/home/ds9/ticketsystem-start# ./bin/start-dev
```

```
[+] Running 1/1
```

```
✓ Container ticketsystem-start-ticket_web-1 Started 12.9s
```

# Swagger画面をリロードして確認します。

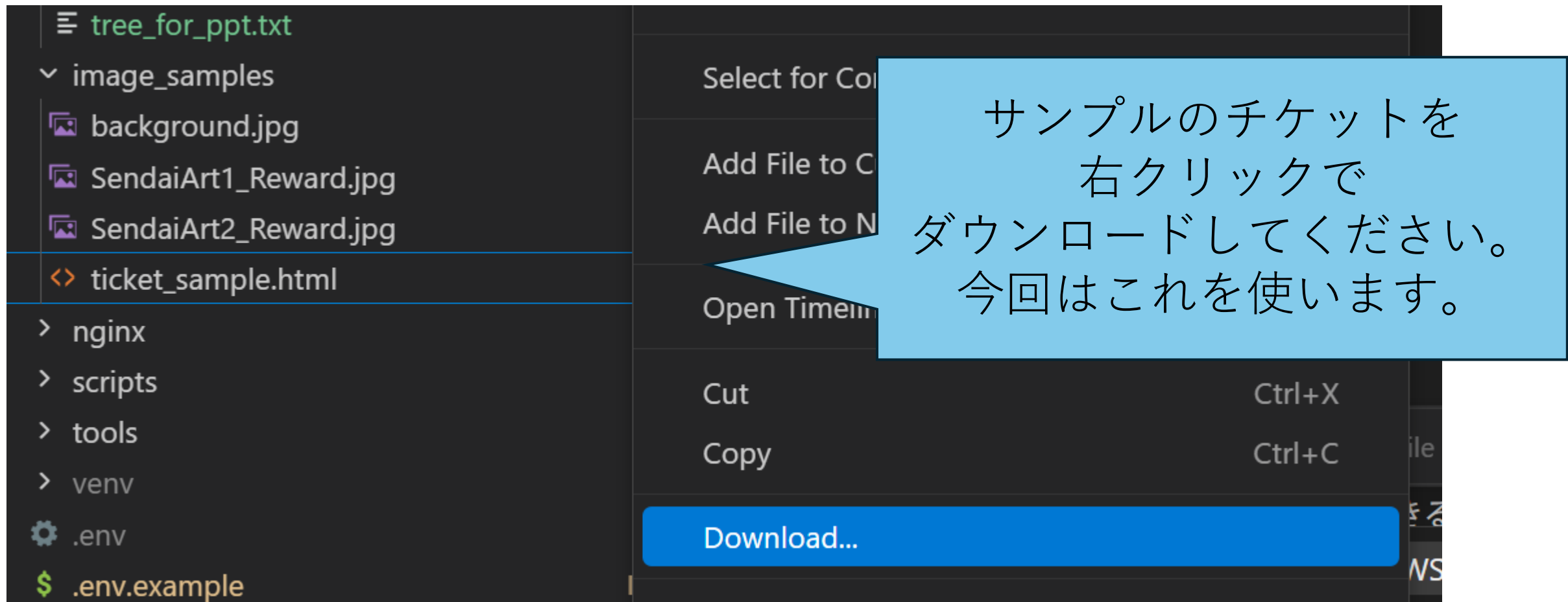


Create Ticket NFTが出来て  
いれば次に試してみます。

まずはチケットのデザインを  
作ります。

# チケットのデザイン

最初にチケットのサンプルデザインがあるのでダウンロードします。



# チケットを確認します

ブラウザで開くと以下のようなデザインになっています。



※背景画像はインラインイメージになっています。  
※余裕のある方はデザインを変えてみてください。

# では、チケットを作成してみます。

**POST** /api/ext/v1/ticket/create Create Ticket NFT

チケットNFTを作成します。event\_name, event\_date, ticket\_html 必須。アップロードした HTML にスタイルが含まれるため TicketDesign は使わな

**Parameters**

No parameters

**Request body**

<b>event_name</b> * required	イベント名 (必須)
string	Tohoku Festival 2026
<b>event_date</b> * required	イベント日時 (必須・ISO 8601推奨)
string	2026-04-01 18:00
<b>venue</b>	会場名・場所 (住所)
string	川内
	<input type="checkbox"/> Send empty value
<b>seat</b>	座席情報
string	A-10
	<input type="checkbox"/> Send empty value
<b>recipient_paymail</b>	受領者paymail (省略時は自分)
string	recipient_paymail
	<input checked="" type="checkbox"/> Send empty value
<b>ticket_html</b> * required	チケット用HTML (必須・スタイル含む)
string(\$binary)	Choose File  ticket_sample.html

**Execute**

もしファイルアップロードのボタンが無い場合は次のページを参考にAIへ指示して修正してください。

Paymail欄は空にしてください。

先程のサンプルチケットを指定 (選択) します。

# チケットのファイルアップロードが無い！？

チケットのファイルアップロードボタンが正しく作成されないことがあります。  
AIへの指示で以下の要点をおさえるようにします。

- 問題：  
（発生しているエラーや挙動を正確に書く）
- 再現条件：  
（いつ・どこで・どうすると起きるか）
- 原因の仮説：  
（自分の考え。なくてもOKだがあると精度UP）
- 目的：  
（最終的にどうなればOKか）
- 制約：
  - 変更していい範囲：（例：views.pyのみ）
  - 変更してはいけない：（例：既存のディレクトリ構成）
  - 実行環境：（例：Docker）
- 要求：
  - 修正は最小限にする
  - 差分形式で提示
  - なぜその修正で直るか簡潔に説明

# その後結果は . . .

↓ レスポンス201で成功しました。 **nft\_origin**や**checkin\_url**があります。

Code

Details

201

Response body

```
{
 "status": "success",
 "message": "Ticket NFT created successfully",
 "nft_origin": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0",
 "transaction_id": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e",
 "ticket_image_url": "/api/ext/v1/ticket/image/2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0",
 "checkin_url": "http://127.0.0.1:8001/api/ext/v1/ticket/checkin?token=2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0:1w1Lud:Q30t47oFJfEcQZcMIVfwFqYTxDUOSL1J0mS1xUpE",
 "nft_information": {
 "nft_id": 46,
 "nft_origin": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0",
 "current_nft_location": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0",
 "original_file_name": "ticket.html",
 "content_type": "text/html",
 "name": "Tohoku Festival 2026 Ticket",
 "metadata": {
 "MAP": {
 "app": "Ticket System",
 "name": "Tohoku Festival 2026 Ticket",
 "type": "ord",
 "subType": "collectionItem",
 "subTypeData": {
 "ticket": {
 "seat": "A-10",
 "venue": "川内",
 "created_at": "2026-03-14T10:07:58.983733Z",
 "event_date": "2026-04-01 18:00",
 "event_name": "Tohoku Festival 2026"
 }
 }
 }
 }
 }
}
```

後ほど使うので念のために  
メモ帳などにコピペしてお  
いてください。

download

# もしエラーなどが発生した場合は

AI のエージェントにエラーの内容をコピーし修正させます。

プロンプト入力例：

**以下のエラーが発生しました。ドキュメントの010\_Step2\_CreateTicket.mdを参考にして修正してください。**

**<エラーの内容をここにペーストします>**

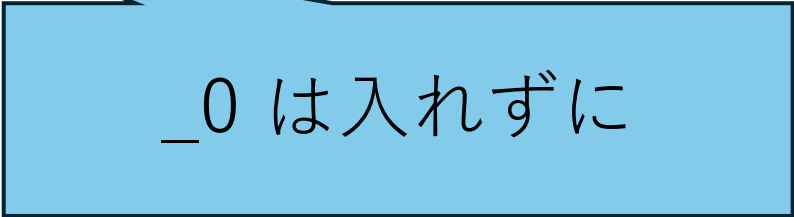
修正後は、先ほどと同様に `./bin/start-dev` で更新します。

# NFTがブロックチェーンにある？

では、本当にブロックチェーンに入ったか確認してみましょう。  
WhatsOnChainというサイトで確認してみます。  
先程のnft\_originの文字列から以下の赤字の部分をコピーします。

**"ticket\_image\_url":**

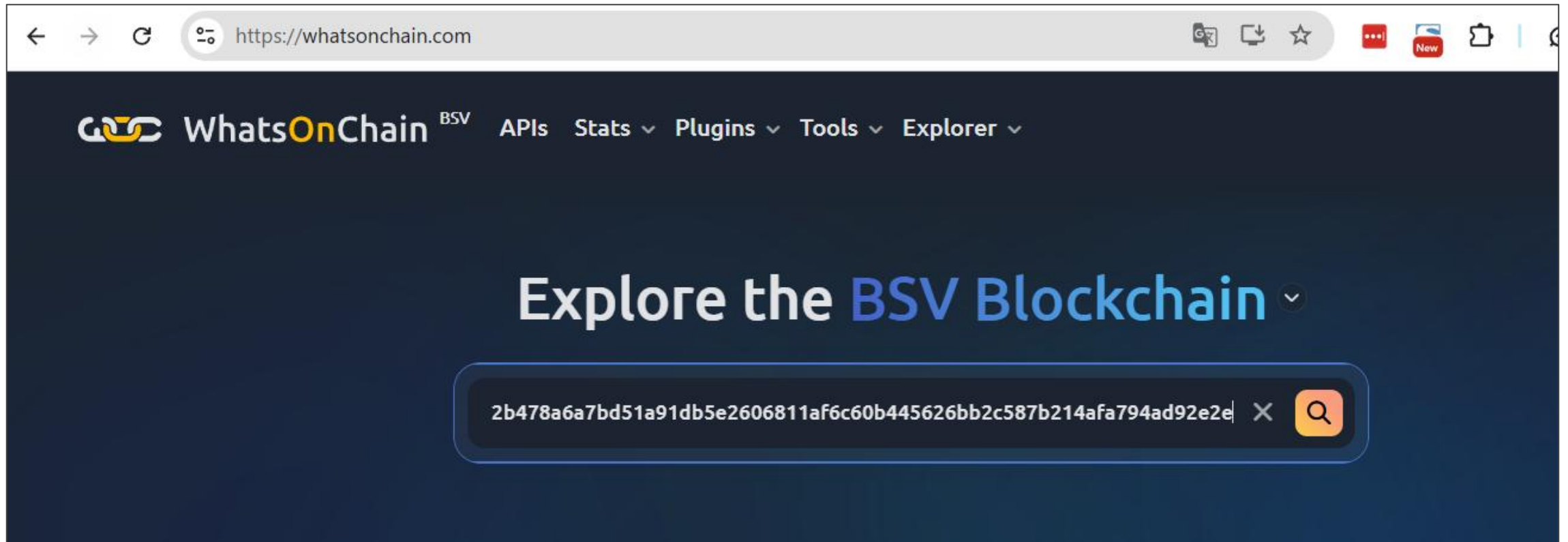
**"/api/ext/v1/ticket/image/2b478a6a7bd51a91db5e2606811af6c60b445626b  
b2c587b214afa794ad92e2e\_0",**



\_0 は入れずに

# NFTがブロックチェーンにある？

<https://whatsonchain.com/> をブラウザで開き、先ほどの文字列（トランザクションID）を入力し検索します。



# NFTがブロックチェーンにある？

トランザクションが見つかりました。右下の Outputに 1satOrdinalsがあります。

The screenshot shows the 'Transaction' page on the WhatsOnChain BSV explorer. The transaction ID is 2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e. The status is 'UNCONFIRMED'. The fee rate is 102.9 sat/KB. The transaction has 1 input and 2 outputs. The first output is labeled '1satOrdinals' and has a value of 0.00000001 BSV. The second output is labeled 'ScriptHash' and has a value of 0.00000001 BSV. The page also shows a search bar, navigation links, and a 'Details' tab.

Transaction ID: 2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e

Status: UNCONFIRMED

Fee Rate: 102.9 sat/KB

Version: 1

1 Input: Total Input: 0.00005411 BSV

2 Outputs: Total Output: 0.00003928 BSV

Output 1: 1satOrdinals, 0.00000001 BSV

Output 2: ScriptHash, 0.00000001 BSV

トランザクションID

InputとOutput

1satOrdinals

# NFTがブロックチェーンにある？

Decodeボタンをクリックするとチケットを見ることができます。

← → ↺ <https://whatsonchain.com/tx/2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e> 📄 📄 ☆ 📄 📄 📄

Default Decoder 📄 📄 ⓘ 📄

Plugins Lab Store

Plugins [learn more](#)

- WOC Default Decoder
- GaiaLog
- Bico decoder
- 3D Model Viewer
- STAS Viewer
- {OPUB} {OPUB} Decoder

**Tohoku Festival 2026**

日時 2026-04-01 18:00  
会場 川内  
座席 A-10  
所有者  
発行日 2026-03-14T10:07:58.982905Z

**TOHOKU UNIVERSITY**

これがNFT♥です

BSVだと簡単に実現できます

# チケットのイメージの確認

Swaggerでチケットのイメージも確認してみましょう。nft\_originをいれます。

"nft\_origin": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e\_0",

**GET** /api/ext/v1/ticket/image/{nft\_origin} Get ticket image

チケット画像（PNG）を返す。HTML テンプレートでレイアウトし QR を含む。

**Parameters**

Name	Description
nft_origin * required	
string	nft_origin

**Responses**

**Curl**  

```
curl -X 'GET' \
'http://127.0.0.1:8001/api/ext/v1/ticket/image/2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0' \
-H 'accept: application/json'
```

**Request URL**  

```
http://127.0.0.1:8001/api/ext/v1/ticket/image/2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0
```

**Server response**

Code	Details
200	<b>Response body</b>   The response body is a PNG image of a ticket for the Tohoku Festival 2026. The ticket has a dark blue background with a circular floral emblem in the center. Text on the ticket includes 'Tohoku Festival 2026', '日時 2026-04-01 18:00', '会場 川内', '座席 A-10', '所有者', '発行日 2026-03-14T10:07:58.982905Z', and 'TOHOKU UNIVERSITY'. A QR code is located in the top right corner of the ticket image.

※今回、QRコードのアドレス  
は127.0.0.1(ローカルアドレス)で、  
このブラウザからアクセスができます。  
(スマホからはアクセスできません)

# いままでの操作は主催者の操作でした。

ここからはお客さま（ユーザ）側のつもりで操作してください。

**ここは、会場の入場口です。**

チケットを提示し使用します。  
(※QRコードのURLにアクセスします。)

TOKENはcheckin\_urlにあります。  
これをコピーでswaggerへ

```
"checkin_url": "http://127.0.0.1:8001/api/ext/v1/ticket/checkin?token=2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0:1w1Lud:Q30t47oFJfEcQzcMIVfWwFqYTxUOSL1JDmSI1xUpE",
```

# お客さま（ユーザ）操作 — チェックイン

チェックインします！

**POST** `/api/ext/v1/ticket/checkin` Ticket check-in execute

token を検証し、未使用チケットを使用済みに更新する。

**Parameters**

No parameters

**Request body** required

Edit Value | Schema

```
{
 "token": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0:1w1Lud:Q30t47oFJfEcgQzcMIVfWwFqYTxU0SL1JDmS11xUpE"
}
```

ペースト

```
{
 "token": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0:1w1Lud:Q30t47oFJfEcgQzcMIVfWwFqYTxU0SL1JDmS11xUpE"
}
```

# お客さま（ユーザ）操作 — 報酬ゲット

レスポンスを確認します。“Checked in” → WOCで確認してみましょう。

Code

Details

200

Response body

```
{
 "used": true,
 "used_at": "2026-03-14T12:15:51.547125Z",
 "nft_origin": "2b478a6a7bd51a91db5e2606811af6c60b445626bb2c587b214afa794ad92e2e_0",
 "message": "Checked in",
 "reward_nft": {
 "created": true,
 "nft_origin": "9adffaa6d571bf48d10c45e214f9358b06d110b1d2e13237008728ebe9cf9559_0"
 }
}
```

報酬NFTをゲット！

9adffaa6d571bf48d10c45e214f9358b06d110b1d2e13237008728ebe9cf9559  
の部分をWOCで確認（\_0の部分は除いてコピペ）

# お客さま（ユーザ）操作 — 報酬ゲット

WOCで確認してみましょう。

The screenshot shows the WhatsOnChain (WOC) interface for a specific transaction. The transaction ID is 9adffaa6d571bf48d10c45e214f9358b06d110b1d2e13237008728ebe9cf9559. The page displays the following details:

- Transaction ID:** 9adffaa6d571bf48d10c45e214f9358b06d110b1d2e13237008728ebe9cf9559
- Timestamp (utc):** 2026-03-14 12:48:06
- Block #:** 940327
- Fee Paid:** 0.00001145 BSV (0.00005411 BSV - 0.00004266 BSV)
- Fee Rate:** 103 sat/KB
- Version:** 1
- Size:** 11,111 B
- Confirmations:** 13
- 1 Input:** Total Input: 0.00005411 BSV. Input #0: 14BP2jqvEeywvyq514L7pLPokh8MNxT SUA via 0b1e62 ... 17f6fb [2].
- 2 Outputs:** Total Output: 0.00004266 BSV. Output #0: 1SatOrdinals. ScriptHash: a85bf7e967f78417a0fbacbe05f71c96e80 b0313c3b364bfbdb42bf882da5dd34.

報酬NFTがありました！

# まとめ：本日の 5 API の全体像を確認

- 1) アカウント作成
- 2) チケット生成（テンプレート）
- 3) 配布（チケット画像取得）
- 4) チェックイン
- 5) チェックイン報酬送付

本日のゴールは、上記 5 API のつながりを理解し、`Step0～Step4` を通して「実装・検証・説明」まで一通り完了することでした。

ブロックチェーンと NFT を活用したチケット発行からチェックイン、報酬送付までの流れを、自分の手で通していただきましたがいかがでしたでしょうか？

# Appendix

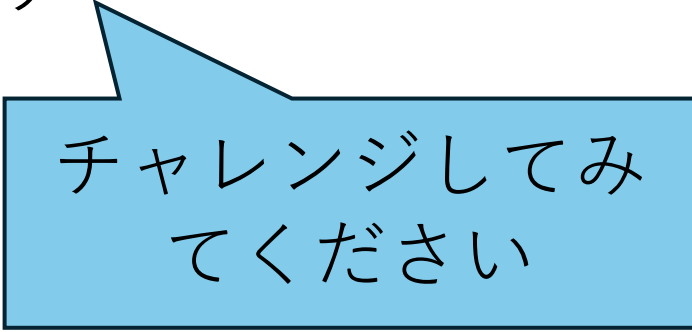
# AIのトークンがまだ使えそうなら・・・

例えば、

012\_Extra.md を用意していますのでトライしてみる

もしくは

追加したい機能があればAIに作らせてみましょう



チャレンジしてみ  
てください