

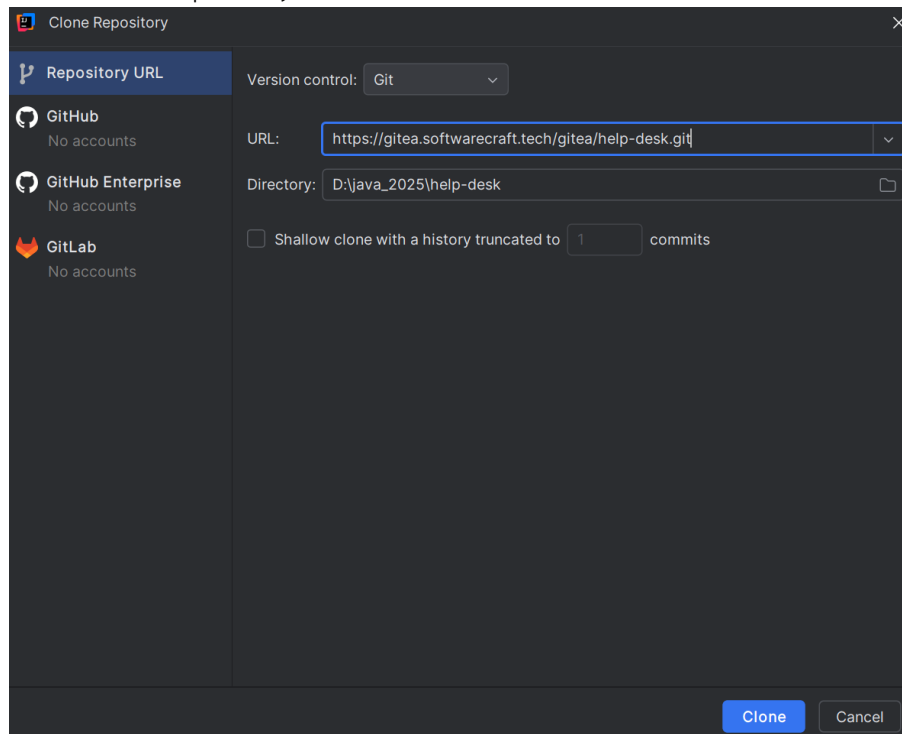
การรันโปรเจกต์ครั้งแรก

- เริ่มต้นโหลด (Pull/Clone) โปรเจกต์จาก Gitea มาใช้กับ IDE

คัดลอก HTTPS

<https://gitea.softwarecraft.tech/gitea/help-desk.git>

เปิด IDE และเลือก Clone Repository



ภาพที่ ... เลือก Clone Repository ใส่ URL ของโปรเจกต์และกำหนดที่เก็บโปรเจกต์ในเครื่อง

- กด Clone

- สิ่งที่ต้องตรวจสอบก่อนเริ่มพัฒนา

ตรวจสอบ branch

git branch

ต้องอยู่ใน main หรือ branch ที่ต้องการพัฒนา

ตรวจสอบ remote

git remote -v

เพื่อให้แน่ใจว่าสามารถ push กลับไปที่ Gitea ได้ เช่น

- สร้างไฟล์ .env ในโฟลเดอร์ infra แล้วระบุ Environment Variables ที่เกี่ยวข้องกับการใช้งาน Mailjet
ไฟล์ infra/.env

```
# Mailjet SMTP Credentials
MAILJET_SMTP_HOST=in-v3.mailjet.com
MAILJET_SMTP_PORT=587
MAILJET_API_KEY=<KEY>
MAILJET_SECRET_KEY=<KEY>
DEFAULT_FROM_EMAIL=<MAIL>
```

- รัน Infrastructure หลัก (DB, Cache, Storage) โดยไปที่โฟลเดอร์ infra และรันเฉพาะ Services ที่เป็น Infrastructure หลัก ซึ่ง Port ได้ถูกกำหนดไว้ใน ports แล้ว

```
cd infra
```

```
docker compose up -d cockroach-1 cockroach-2 cockroach-3 init-cluster redis minio
celery_worker flower
```

ตรวจสอบที่

<http://localhost:8080/>

<http://localhost:9001/>

<http://localhost:5555/>

- สร้างไฟล์ชื่อ **.env** ในโฟลเดอร์ backend/ ระบุ Environment Variables ที่เกี่ยวข้อง
ไฟล์ชื่อ backend/.env

```
# ฐานข้อมูล (เชื่อมต่อกับ Port ที่ถูก Publish)
```

```
DB_HOST=localhost
```

```
DB_PORT=26257
```

```
# Redis
```

```
REDIS_HOST=localhost
```

```
REDIS_PORT=6379
```

```
# Celery Broker URLs
```

```
CELERY_BROKER_URL=redis://127.0.0.1:6379/0
```

```
CELERY_RESULT_BACKEND=redis://127.0.0.1:6379/0
```

```
# MinIO (S3) Credentials
```

```
MINIO_ENDPOINT=http://localhost:9000
```

```
MINIO_ACCESS_KEY=minio_admin
```

MINIO_SECRET_KEY=minio_p@ssw0rd!

AI_SERVICE_INTERNAL_URL=http://localhost:8001

Mailjet SMTP Credentials

MAILJET SMTP_HOST=in-v3.mailjet.com

MAILJET SMTP_PORT=587

MAILJET_API_KEY=<KEY>

MAILJET_SECRET_KEY=<KEY>

DEFAULT_FROM_EMAIL=<MAIL>

- รัน Backend โดยในโฟลเดอร์ backend

python -m venv venv

venv\Scripts\activate

pip install -r requirements.txt

python create_db.py

python manage.py migrate

python manage.py createsuperuser

ระบุข้อมูล

Username: admin

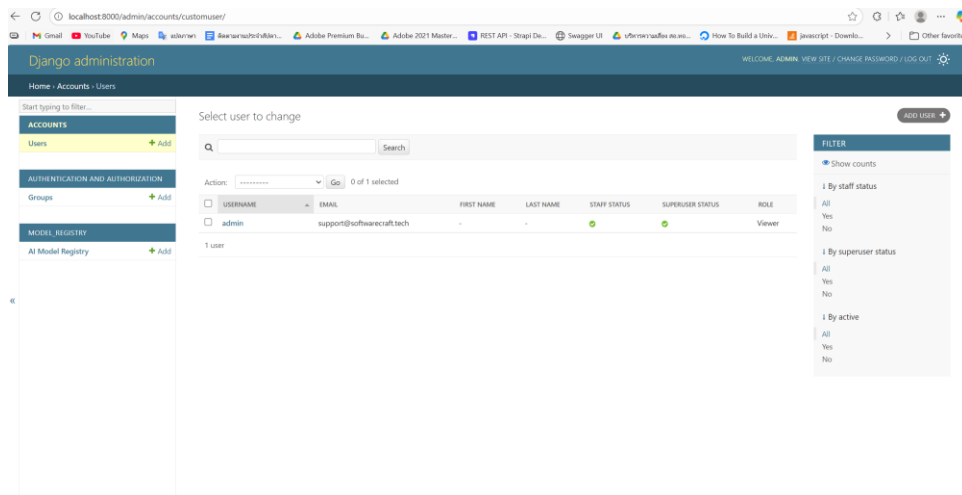
Email:support@softwarecraft.tech

Password: Str0ngp@ssword123-

python manage.py runserver 0.0.0.0:8000

ตรวจสอบที่

http://localhost:8000/admin/



ภาพที่ ... หน้า Django Admin

- สร้างไฟล์ชื่อ **.env** ในโฟลเดอร์ web/ ระบุ Environment Variables ที่เกี่ยวข้องกับไฟล์ชื่อ web/.env

VITE_API_BASE_URL=http://localhost:8000

- เข้าไปในโฟลเดอร์ web แล้วรันคำสั่ง

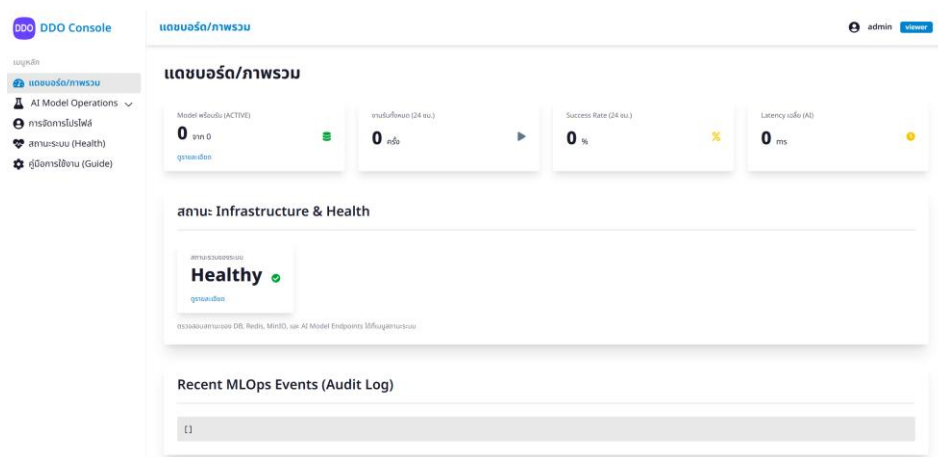
npm install

npm audit fix

npm run dev

จากนั้นเข้าไปที่

http://localhost:5173/



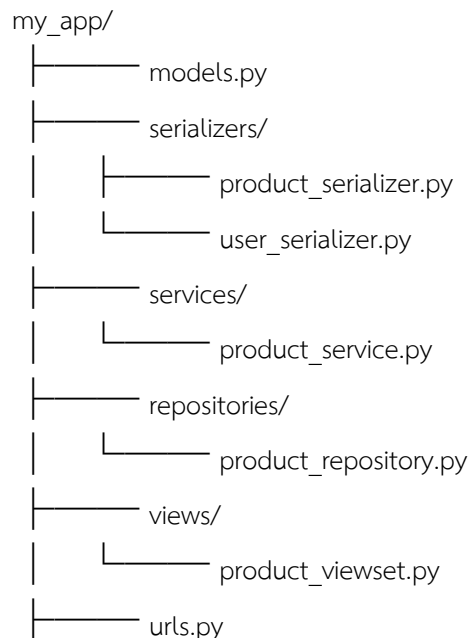
ภาพที่ ... สามารถล็อกอินได้

การเพิ่มแอปพลิเคชันที่จำเป็นใน Django Backend

แอปพลิเคชันที่จำเป็น ได้แก่

- helpdesk เพื่อจัดการ Logic หลักของระบบ (Ticket Model, Assignment, Status)
- chat เพื่อจัดการฟังก์ชันการสื่อสาร Real-time (Agent-to-Agent หรือ Agent-to-User) ซึ่งจะใช้ WebSockets, Django Channels, Redis Channel Layer
- สร้างแอปเพิ่มใน backend เพื่อแยก “ขอบเขตการทำงาน” (domain) ของระบบ
- แอป api ทำหน้าที่เป็น "ศูนย์รวม Logic ทั่วไป" และเป็นที่สำหรับ API ที่ไม่เข้าโดเมนของแอปอื่น ๆ
- รวม API Endpoint หลัก (Routing) ที่ไฟล์ core/urls.py

การพัฒนาแอปพลิเคชันจะจัดโครงสร้างตามมาตรฐานของการพัฒนา REST API ด้วย Django REST Framework (DRF) แบบ Clean Code / Clean Architecture



ภาพที่ ... โครงสร้าง DRF ระดับ “Clean Architecture”

ลำดับขั้นตอนของการพัฒนา REST API ด้วย Django REST Framework (DRF) ที่ สอดคล้องกับหลักการ Clean Code + Clean Architecture

1. สร้าง Model ในแอป

- “เก็บแค่โครงสร้างข้อมูลและกฎเชิงข้อมูล (data rules)”
- เป็นชั้น Entity Layer (Domain Model)

2. สร้าง Repository Layer ในแอป

- “แยกการเข้าถึงฐานข้อมูลออกจาก business logic”
- ลด coupling และทำให้เปลี่ยน ORM หรือ DB ได้ง่าย (เหมือน Spring’s Repository)

3. สร้าง Serializer Layer ในแอป

- “เป็นตัวกลางในการแปลง Model ↔ JSON / API Input”
- เป็นเหมือน DTO/Mapper ของ Spring Boot

4. สร้าง Service Layer ในแอป

- “เก็บ business logic ทั้งหมด”
- เช่น ตรวจสอบเงื่อนไข, คำนวณ, หรือเรียกหลาย repository พร้อมกัน

5. สร้าง ViewSet Layer ในแอป

- “เป็น Controller ชั้นบนสุด”
- ติดต่อกับ request/response เท่านั้น ไม่ควรมี logic ธุรกิจในนี้

6. กำหนด Routing ของแอปและของโปรเจกต์หลัก

- “เชื่อม ViewSet กับ URL path”
- เป็น Interface Layer ที่เปิดให้ client เข้ามาใช้งาน

การพัฒนา Page ใหม่/ปรับปรุง Page เดิม ของ Frontend (Web)

โปรเจกต์นี้ใช้สถาปัตยกรรม Redux Toolkit + React Hook Form + TanStack Query และระบบ Routing/Layouts ที่กำหนดไว้ (AuthRoutes, PrivateRoutes, MainLayout) สามารถสรุปขั้นตอนหลัก ๆ ได้ดังนี้

1 Page ใหม่เป็นส่วนของระบบที่ต้องล็อกอิน

- สร้าง Component และ Page ใน src/components และ src/pages
- เพิ่ม Route สำหรับหน้า Page ใหม่ ในไฟล์ src/routes/pageRoutes.jsx เพื่อให้ผู้ใช้สามารถเข้าถึงหน้า Page ผ่าน URL ที่ต้องการได้ (การจัดการเส้นทางภายใต้ /dashboard/*)
- ปรับปรุงเมนู เช่น ชื่อเมนู หรือสิทธิ์ในการเข้าถึง เป็นต้น ในไฟล์ src/config/sidebarRoutes.jsx (สำหรับเมนู Sidebar)

2 การจัดการข้อมูล (Data Fetching & State)

- สร้าง Hook ใหม่ โดยสร้างไฟล์ Service API (TanStack Query) ใน src/services/ เช่น reportApi.js เป็นต้น เพื่อกำหนด Queries (สำหรับดึงข้อมูล) และ Mutations (สำหรับส่งข้อมูล/สร้าง/อัปเดต/ลบ) โดยใช้ useQuery หรือ useMutation
- ใช้ axiosClient.js ที่มี Interceptor สำหรับแนบ Token อัตโนมัติ (สำคัญมากสำหรับ API ที่ต้องล็อกอิน)
- ใช้ Redux Slice (สำหรับ Global State) โดยให้สร้าง Slice ใหม่ในโฟลเดอร์ src/features/ เช่น src/features/reports/reportSlice.js เป็นต้น หากต้องการจัดการสถานะที่เป็น **Global** และไม่เกี่ยวข้องกับการดึงข้อมูลโดยตรง เช่น สถานะ UI, การแจ้งเตือน, ข้อมูลที่ไม่เปลี่ยนแปลงบ่อย เป็นต้น
- เพิ่ม Reducer ใหม่ในไฟล์ src/app/store.js

3. การจัดการฟอร์มและการตรวจสอบ (Form & Validation)

- หาก Page ใหม่มีแบบฟอร์ม (Form) ให้ใช้ React Hook Form ร่วมกับ Yup
- สร้างไฟล์ใหม่ใน src/schemas/ เช่น reportSchema.js เป็นต้น เพื่อกำหนดกฎการตรวจสอบข้อมูลด้วย Yup

- ใน Component Page เช่น NewReportPage.jsx ให้ใช้ useForm และ yupResolver เพราะการใช้ useForm คู่กับ yupResolver ทำให้ React Hook Form สามารถรับ Schema ที่กำหนดไว้ใน src/schemas/ มาใช้ตรวจสอบข้อมูลฟอร์มได้โดยอัตโนมัติ

4 การใช้งาน Toast Component เพื่อแสดงข้อความ

- เป็นการใช้ Redux Toolkit ในการส่ง (Dispatch) Action เพื่อเพิ่มข้อความแจ้งเตือนเข้าไปใน Global State โดยที่ Component หลัก ได้แก่ Toast Component จะคอยตรวจสอบ State และแสดงผลแจ้งเตือนออกมา
- ใน Component ที่ต้องการเรียกใช้ Toast Component เพื่อแสดงข้อความ ต้องมีการ import { useDispatch } from 'react-redux'; และ Import Action เช่น import { addToast } from '../features/toast/toastSlice';
- ใช้ dispatch(...) เมื่อเกิดเหตุการณ์ที่ต้องการแจ้งเตือน (เช่น API สำเร็จหรือล้มเหลว) โดยกำหนดอย่างน้อย 2 fields ได้แก่ message ข้อความที่ต้องการแสดงใน Toast Notification กับ type ประเภทของ Toast ซึ่งกำหนดสไตล์ (เช่น success, error, warning)

5 การใช้งาน Custom Confirmation Dialog เพื่อแสดงข้อความ (Modal Component)

- Component แม่ จะทำหน้าที่เป็น **Controller** โดยการกำหนดสถานะ (State) สำหรับควบคุม Modal และเก็บข้อมูลที่เกี่ยวข้อง
- สร้าง Handler Function ใน Component แม่ เพื่อกำหนดฟังก์ชันใน Component แม่ สำหรับจัดการเหตุการณ์ต่าง ๆ
- ส่งผ่าน Handler ไปยัง Component ลูก เพื่อให้ Component ลูกเรียกใช้ ซึ่งจะถูกเรียกใช้เมื่อถึงเวลาที่เหมาะสม เช่น กดปุ่ม เป็นต้น
- Component แม่ จะทำการ Render Custom Confirmation Dialog และเชื่อมต่อ Props เพื่อควบคุมการทำงาน