МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
ХЭРЭГЛЭЭНИЙ ШИНЖЛЭХ УХААН, ИНЖЕНЕРЧЛЭЛИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ

Веб програм хөгжүүлэлт
(Full-stack web development)

Програм хангамж(D061302)
Үйлдвэрийн дадлагын тайлан

Удирдагч: _____ Д. Эрдэнэбаяр

Хамтран удирдагч: _____ Н. Оюун-Эрдэнэ

Гүйцэтгэсэн: _____ Д. Балжинням (20B1NUM0563)

Улаанбаатар

2023 оны 9 сар

Зохиогчийн баталгаа

Миний бие Даянгийн Балжинням ”Веб програм хөгжүүлэлт” сэдэвтэй судалгааны ажлыг гүйцэтгэсэн болохыг зарлаж дараах зүйлсийг баталж байна:

- Бусдын хийсэн ажлаас хуулбарлаагүй, ашигласан бол ишлэл, зүүлт хийсэн.
- Ажлыг би өөрөө (хамтарч) хийсэн ба миний хийсэн ажил, үзүүлсэн дэмжлэгийг тайлангийн ажилд тодорхой тусгасан.
- Ажилд тусалсан бүх эх сурвалжид талархаж байна.

Гарын үсэг: _____

Огноо: _____

ГАРЧИГ

ЗУРГИЙН ЖАГСААЛТ

ХҮСНЭГТИЙН ЖАГСААЛТ

Кодын жагсаалт

УДИРТГАЛ

Миний бие Даянгийн Балжинням ”Веб програм хөгжүүлэлт” сэдэвтэй үйлдвэрийн дадлагын ажлыг Dentsu Data Artist Mongol компани дээр гүйцэтгэсэн. Энэхүү үйлдвэрийн дадлагын хүрээнд Python, болон Javascript програмчиллын хэлнүүд дээр түлхүү ажилсан.

Энэхүү дадлагын ажлын хүрээнд Python-гийн FAST-API framework, Javascript-н Vuejs дээр ажилсан билээ. Хийх ажлын гол зорилго нь, гараас хэрэглэгчийн оруулсан текстийг PNG файл болгон **машин сургалтын** аргаар, шошго үүсгэхэд ашиглах.

Table 1: Дадлагын төлөвлөгөө

№	**Гүйцэтгэх ажил**	**Хугацаа**	**Биелэлт**	**Дадлагын удирдагчийн үнэлгээ**
1	Гараас оруулсан текстийн дагуу PNG үүсгэхэд шаардлагтай технологийг судлах	06/07 - 06/09		
2	Front-End дээр үүсгэсэн PNG file-г үзэх хэсгийг хэрэглэгчид хялбар байдлаар хийх	06/09 - 06/15		
3	Back-End дээр динамик байдлаар хэрэглэгчийн оруулсан текстийг Фонтын хэмжээ, өнгө, чимэглэлийн дагуу үүсгэх	06/15 - 06/23		
4	Ашиглаж болохуйц End-Point үүсгэх	06/23 - 06/24		
5	Үүсгэсэн файлыг ашигласан тохиолдолд AWS ашиглах логик хэрэгжүүлэх	06/24 - 06/26		
6	Database дээр шинэ мөр нэмж migration хийх	06/26 - 06/27		

1. БАЙГУУЛЛАГЫН ТАНИЛЦУУЛГА

1.1 Товч танилцуулга

Dentsu Data Artist Mongol нь 2018 оны 6-р сард, Data Artist Inc.-ийн охин компани болж байгуулагдсан. Дэнцү группын гишүүний хувьд дэлхийн өнцөг булан бүрт байгаа группын компаниудад тоон маркетингийн чиглэлээр өгөгдлийн шинжилгээ, AI model, систем хөгжүүлэх үйлчилгээг үзүүлдэг. Дэнцү грүпп нь маркетингийн чиглэлээр дэлхийд тавд эрэмбэлэгддэг.

1.2 Ямар үйлчилгээ үзүүлдэг вэ?

Уг компани нь мэдээллийн технологийн чиглэлээр үйлчилгээ явуулдаг бөгөөд голчлон Японы компаниудад хиймэл оюун, ухаан машин сургалтын үйлчилгээ үзүүлдэг. Мөн шаардлагатай тохиолдолд систем хөгжүүлэлтийг хийдэг ба тухайн багт нь би байдаг.

2. СИСТЕМИЙН ШААРДЛАГА

2.1 Танилцуулга

Миний дадлагын хугацаанд ажилсан систем нь хиймэл оюун ухаан, машин сургалтын технологийг ашиглан бүх төрлийн шошгыг хэрэглэгчийн оруулсан мэдээллийг ашиглан үүсгэдэг систем юм. Ингэснээр дизайнер хүмүүсийн ажлыг хөнгөвчилж байгаа билээ. Энэхүү систем нь recommendation model ашиглан ямар төрлийн хүмүүст зориулснаар нь ялгаж өөр төрлийн хэв маягийн өнгө, зураг, дизайн сонгодог ба тэрхүү сонгодогсон матералуудыг LAYOUT-GAN++ гэх model ашиглан layout-г нь тохируулж эцсийн бүтээгдэхүүнийг үүсгэдэг.

2.2 Функционал шаардлагууд

- **Хэрэглэгчийн бүртгэл**: Вэбсайт нь хэрэглэгчдэд цахим шуудангийн хаягаа ашиглан бүртгүүлэх, бүртгэл үүсгэх боломжийг олгох ёстой.
- **Хэрэглэгчийн нэвтрэлт**: Бүртгэгдсэн хэрэглэгчид бүртгүүлсэн цахим шуудан болон нууц үгээ ашиглан бүртгэлдээ нэвтрэх боломжтой байх ёстой.
- **Хэрэглэгчийн хяналтын самбар**: Нэвтэрсэний дараа хэрэглэгч өөрийн профайлыг удирдах, шошго үүсгэх функцэд хандах боломжтой хяналтын самбартай байх ёстой.
- **Зураг байршуулах**: Хэрэглэгчид шошго үүсгэхийг хүссэн зургаа байршуулах боломжтой байх ёстой (adobe illustrator file ашиглах ёстой).
- **Машин сургалтын model ашиглах endpoint**: Вэбсайт нь байршуулсан зураг дээр үндэслэн шошго үүсгэх боломжтой endpoint-уудтай байх ёстой.
- **Шошго үүсгэх**: Зургийг байршуулсны дараа вэбсайт нь машин сургалтын model ашиглан шошго үүсгэх ёстой.

- **Шошгоны дэлгэц**: Вэбсайт нь үүсгэсэн шошгыг хэрэглэгчдэд харуулах ёстой.
- **Шошго татаж авах**: Хэрэглэгчид үүсгэсэн шошгыг тохирох форматаар татаж авах боломжтой байх ёстой (AI эсвэл PNG).
- **Хэрэглэгчийн санал хүсэлт**: Хэрэглэгчид үүсгэсэн шошгоны нарийвчлалын талаар санал хүсэлт өгөх боломжтой байх ёстой.

2.3 Функционал бус шаардлагууд

- **Хурд**: Шошго үүсгэх үйл явц нь хамгийн бага хоцрогдолтой, хурдан бөгөөд үр дүнтэй байх ёстой.
- **Аюулгүй байдал**: Хэрэглэгчийн мэдээлэл, үүнд байршуулсан зураг, хамгаалагдсан байх ёстой.
- **Өргөтгөх чадвар (Scalibility)**: Вэбсайт нь хурд алдагдуулахгүйгээр олон тооны хэрэглэгчдэд үйлчлэх чадвартай байх ёстой.
- **Ашиглах боломж**: Вэбсайт нь хэрэглэгчдэд ээлтэй интерфэйстэй, ойлгомжтой зааварчилгаа, хялбар навигацтай байх ёстой.
- **Найдвартай байдал**: Шошго үүсгэхэд ашигладаг машин сургалтын model нь үнэн зөв, найдвартай үр дүнг өгөх ёстой.
- **Хүртээмжтэй байдал**: Вэбсайт нь өөр өөр хөтөч (Chrome, Firefox, Safari гэх мэт) болон төхөөрөмжүүдтэй (ширээний компьютер, гар утас, таблет) нийцтэй байх ёстой.
- **Maintainibility**: Вэбсайт болон түүний үндсэн код нь засвар үйлчилгээ хийх, шинэчлэхэд хялбар байх ёстой.
- **Дагаж мөрдөх**: Вэбсайт нь өгөгдөл хамгаалах хууль, дүрэм журамд нийцсэн байх ёстой.

3. АШИГЛАХ ТЕХНОЛОГИУД

3.1 Back-end талын технологиуд

3.1.1 Python, FastAPI

FastAPI нь python хэлний ASGI[1] framework ба үзүүлэлтийн хувьд nodejs эсвэл go зэргийн үзүүлдэг маш өндөр үзүүлэлтрүү дөхдөг билээ.

- Asynchronous байдлаар ажиллаж чаддаг байдал нь cpu-ны олон цөмийг ашиглах боломжийг олгодог ингэснээр илүү олон хандалт зэрэг авч чадна.
- Pydantic, Starlette гэсэн хоёр сан дээр суурилсан ба, starlette нь ASGI байдлаар ажиллах боломжийг олгох бол, Pydantic нь server дээр validation хийх боломжийг олгодог.
- Database migration хийхэд sqlalchemy ашигладаг ба энэ нь python хэлний ORM[2] ба давуу тал нь хөгжүүлэгч шууд database-тай харьцах биш python-г ашиглан харьцах боломжийг олгоно, ингэснээр database-н схемыг өөрчлөхэд хялбар болохоос гадна database injection зэргээс сэргийлэх давуу талтай.

3.2 Front-end талын технологиуд

3.2.1 Vuejs

Front-end талын хэсгийн технологи бол Vuejs-н progressive framework **Nuxt.js**[3] ба давуу тал нь SSR хийх боломжийг олгодгоос гадна бусад routing, local storage, гэх мэт хөгжүүлжэгчдийн өөрсдөө тохируулдаг зүйлсийг цаанаас нь шийдэж өгсөн байдаг.

[1] Asynchronous Server Gateway Interface

[2] Object Relational Mapping

[3] `https://nextjs.org/`

3.3 Бусад

3.3.1 Amazon S3

Amazon S3 нь Amazon-н cloud service ба энэ нь хэрэглэгчдийн өгөгдөл, зураг, видео, гэх мэт өгөгдөл хадгалах, хэрэглэгчдийн хандахад хялбар байдлаар хандах боломжийг олгодог.

3.3.2 ImageMagick

ImageMagick нь код бичих замаар зурагт засвар оруулдаг сан.

3.3.3 LAYOUTGAN++

LAYOUTGAN++ дээр fine-tune хийснээр, аль болох хэрэглэгчид таалагдахуйц байдлаар шошгон дээрх матералуудыг байршуулах боломжтой болно.

3.3.4 Dockerizing

Орчин үеийн нэгэн гайхалтай технологи бол контайнерчлах юм. Яагаад Docker чухал вэ гэвэл, ямар нэгэн систем хөгжүүлэгчийн компьютер аль эсвэл ямар сервер дээр ажиллаж байгаагаас үл хамааран програм нь өөрийн тусдаа орчинд ажиллах юм. Яг л Virtual machine шиг гэхдээ давуу тал нь Docker host system-ийнхээ цөмийг (kernel)-г ашигладаг учраас маш бага хэмжээгий зай, нөөц ашигладаг.

3.3.5 CI/CD

Мөн сүүлийн үед маш их өргөн түгж байгаа ойлголт бол Continius Integration/Continius Development. Энэ нь програм хангамж ямар ч нөхцөлд хөгжүүлэлт тасралтгүй явж байх орчноор хангадаг ба системд хэзээ ч тасалдал үүсгэхгүй мөн хүний оролцоог маш бага байлгах давуу талтай.

4. ХЭРЭГЖҮҮЛЭЛТ

4.1 Текстээс зураг үүсгэх

4.1.1 Database дээр зурагний мэдээлэл хадгалах table нэмэх

Database-н table дээр өөрчлөлт оруулахдаа бид sqlalchemy ашиглаж байгаа ба доор бичсэн моделийн дагуу бид migrate хийх юм. Үүний тулд back-end ажиллаж байгаа Docker container-лүү shell нээж төслийн root хэсгээс

```
alembic revision --autogenerate -m "your commit message"
```

гэсэн коммандын ашиглан өөрлөлт оруулах мэдээллиг үүсгэн. Дараа нь

```
alembic upgrade head
```

комманд хийснээр Датабазын модел бүрэн өөрчлөгдөнө.

```
class User(Base):
  # ...
  # Other table information

  text_title_string = Column(String(255), nullable=False, default="")
  text_title_color = Column(String(255), nullable=False, default="
     #000000")
  text_title_font_size = Column(Integer, nullable=False, default=0)
  text_title_font_name = Column(String(255), nullable=False, default=
     "")
  text_title_is_vertical = Column(Integer, nullable=False, default=0)
```

Код 4.1: Table-рүү оруулсан өөрчлөлт

4.1.2 API үүсгэх

Back-end дээр API үүсгэхэд анхаарах хэдэн зүйл бий.

1. Validation хийх ингэснээр хэрэглэгчээс ирсэн мэдээллийг шалгаж нэг ёсондоо ажиллахад ямар нэгэн асуудалгүй болгож байгаа юм. Хэрэв буруу мэдээлэл хүсэлт маягаар ирвэл **422 Unprocessable Entity** гэсэн хариу буцаана.

2. Dependency injection байдлаар user-н token-г шалгана ингэснээр хэрэглэгчийн нэвтрэлтийг шалгаж байгаа юм. Хэрэв хэрэглэгч нэвтрээгүй бол **401 Unauthorized**

```python
from fastapi import APIRouter, Depends, status
from sqlalchemy.orm import Session
from app import schemas
from app.feature.pillow import pillow_text_generator, preview_text
from app.models.user import User
from app.v1 import deps
router = APIRouter()


@router.post("/", status_code=status.HTTP_201_CREATED)
def pillow_text(
    *,
    obj_in: schemas.PillowBase,
    db: Session = Depends(deps.get_db),
    _: User = Depends(deps.get_current_user),
):
    return preview_text(
        text=obj_in.text,
        is_vertical=obj_in.is_vertical,
        color=obj_in.color,
```

```
            font_size=obj_in.font_size,
            strokewidth=obj_in.stroke_width,
            bordered=obj_in.bordered,
            PPI=obj_in.PPI,
        )
```

Код 4.2: Зураг буцаан user-лүү илгээх endpoint

4.1.3 Үүсгэсэн API дуудах

Энэхүү endpoint-руу user үүсгэх зурагнийхаа текст мэдээллийг JSON хэлбэрээр POST request явуулна. Frontend-с үүсгэсэн Endpointoo ашиглахдаа responseType-г нь arraybuffer болгож байгаа юм ингэснээр base64 image interneteer явуулснаас харьцангуй бага bandwidth ашиглах юм. Үүний дараагаар авсан датагаа decode хийж base64 img болгон хэрэглэгчид харуулна.

```
async preview_text_to_image({ _ }, { data }) {
  const result = await this.$axios.post(`text_image/`, data, {
    responseType: 'arraybuffer',
  })
  if (result.status === 200) {
    const b64 = btoa(String.fromCharCode(...new Uint8Array(result.
       data)))
    const imgData = 'data:' + result.headers['content-type'] + ';
       base64,' + b64
    return imgData
  } else {
    console.log('error')
  }
},
```

Код 4.3: Endpoint дуудах function

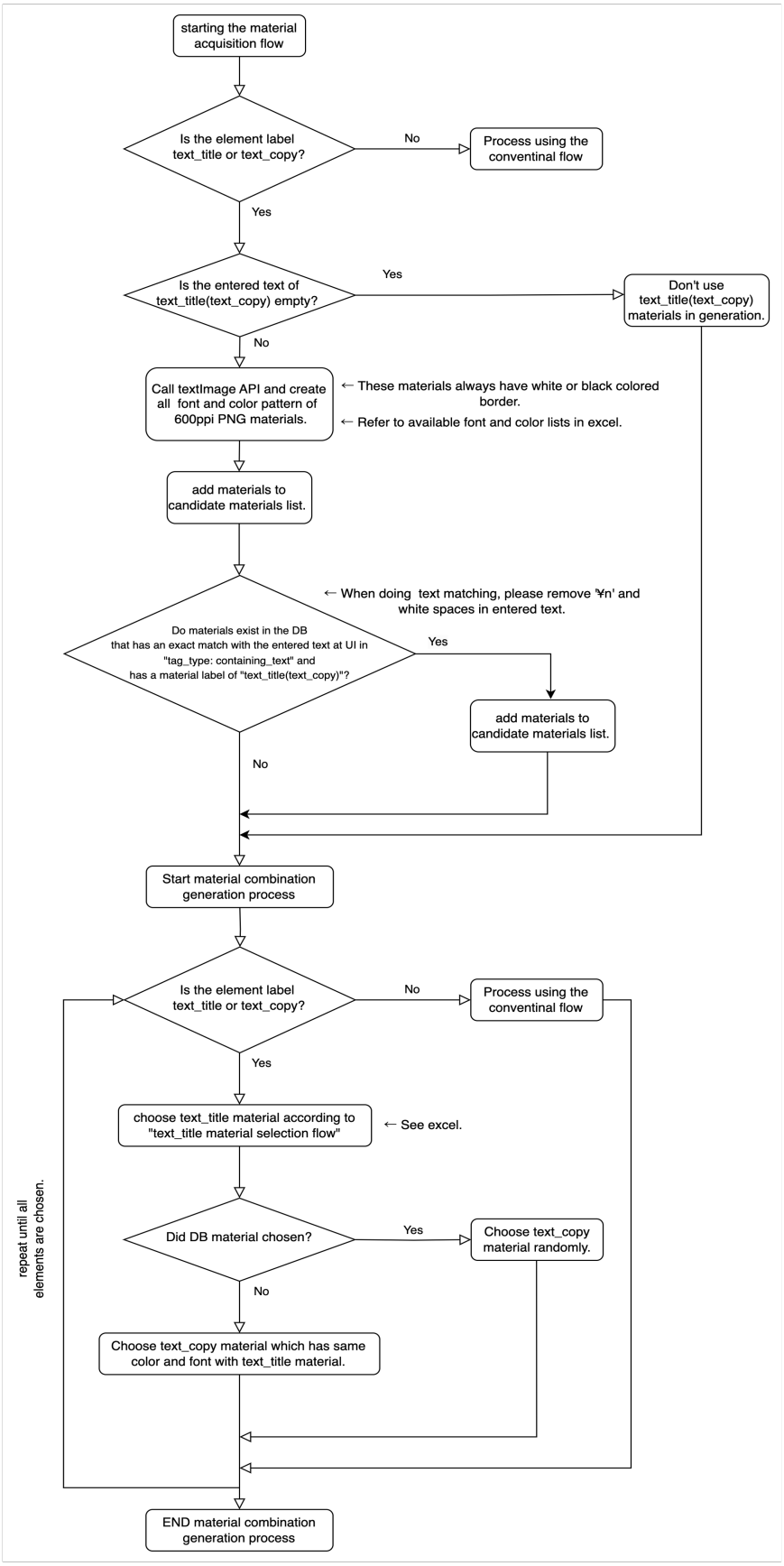
4.2 Business logic-г хийх

Ямар ч request client талаас ирж болох учраас үүнийг сервер дээр боловсруулахын өмнө, validation хийх нь зөв билээ. Үүний тулд Pydantic ашиглаж байгаа юм. Pydantic нь Python дээр зориулсан validation library юм. Үүнийг ашиглан бид үүсгэсэн PillowBase class-н мэдээллийг шалгаж байгаа юм. Хэрэв буруу мэдээлэл ирвэл 422 Unprocessable Entity гэсэн хариуг буцаана.

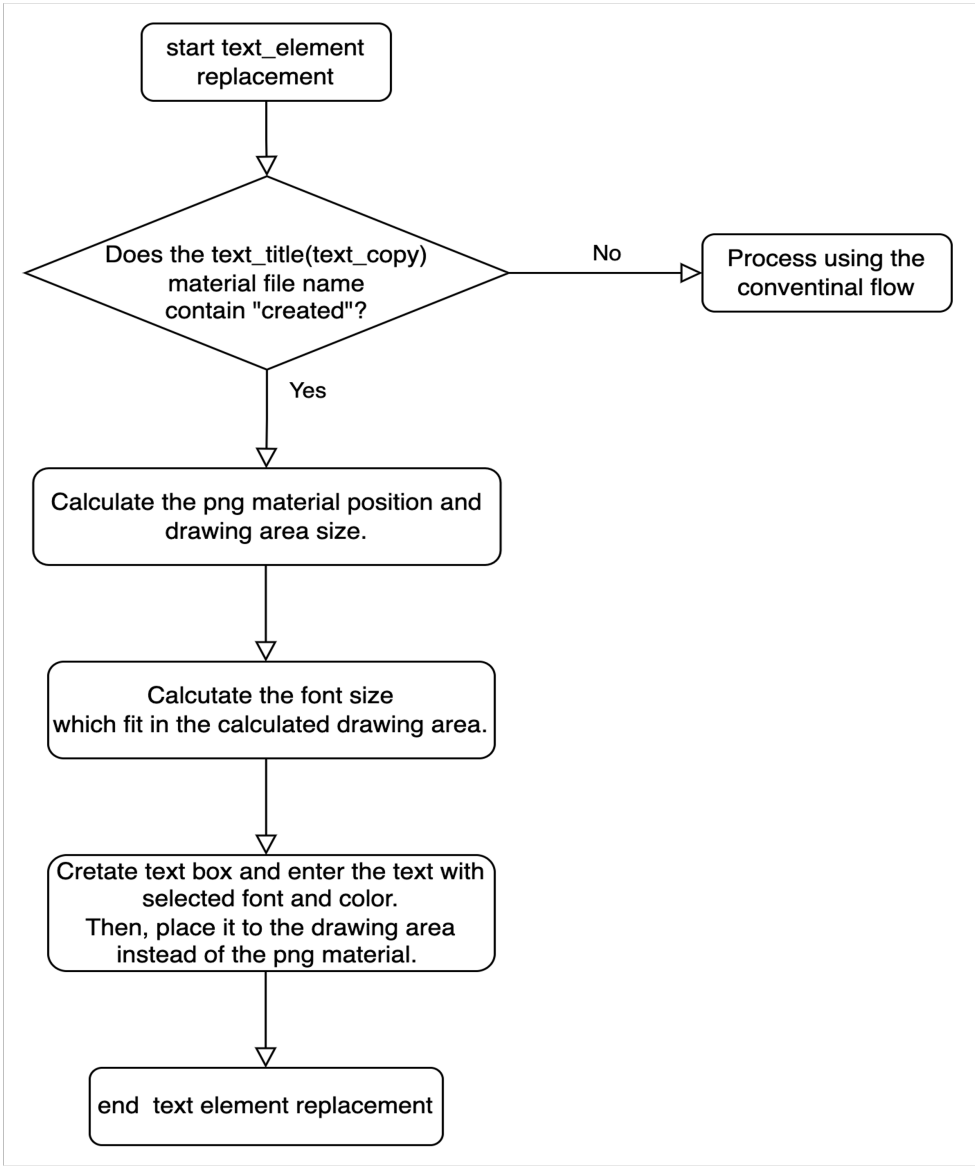
```
from pydantic import BaseModel
class PillowBase(BaseModel):
  text: str
  font_path: str
  is_vertical: bool = False
  font_size: Optional[int] = 90
  color: Optional[str] = "#000000"
  bordered: Optional[bool] = True
  stroke_color: Optional[str] = "#ffffff"
  stroke_width: Optional[int] = 3
  gaps: Optional[int] = 3
  PPI: Optional[int] = 300
```

Код 4.4: Pydantic Validator

Өмнөх API-г зөвхөн хэрэглэгч ямар хэлбэр дүрстэй зураг үүсгэснээ харах зорилготой байсан бол яг production орчинд хэрвээ тэрхүү үүсгэсэн зураг нь таалагдсан бол, текстээс үүсгэсэн зураг материалд ашиглахад тэс өөр логик ашиглах юм.



Зураг 4.1: Алгоритм



Зураг 4.2: Алгоритм-2

4.2.1 Алгоритмын тайлбар

Дээрх зурганд харуулсан Алгоритмыг ерөнхийд нь тайлбарлавал, хэрэглэгчийн зураг үүсгэхийг хүссэн тексттэй адилхан агуулгатай өгөгдөл database дээр байгаа эсэхийг шалгах юм. Хэрэв тийм өгөгдөл олдвол тухайн өгөгдлийн цаашид үүсгэх шошгонууд дээрээ жин өгч магадлал тооцон ашиглах юм.

5. ДҮГНЭЛТ

5.1 Дүгнэлтийн хэсэг

5.1.1 fr fr

Bibliography

[1] Inserting Images, Share LaTex, https://www.sharelatex.com/learn/Inserting_Images

[2] Paragraphs and new lines, Share LaTex, https://www.sharelatex.com/learn/Paragraphs_and_new_lines

[3] Bold, italics and underlining, Share LaTex, https://www.sharelatex.com/learn/Bold,_italics_and_underlining

[4] Lists, Share LaTex, https://www.sharelatex.com/learn/Lists

[5] Tables, Share LaTex, https://www.sharelatex.com/learn/Tables

А. ШИНЖИЛГЭЭ ЗОХИОМЖ

Хавсралтын агуулга

B. КОДЫН ХЭРЭГЖҮҮЛЭЛТ

B.1 Python

B.1.1 Гол script

```python
import io
import os
import random
import typing as t
import uuid

from fastapi.responses import StreamingResponse
from PIL import Image, ImageDraw, ImageFont

from app.core.config import settings
from app.schemas.materials import GeneratedMaterial
from app.utils.s3_handler import upload_file_object

color_dict = {
    "#000000": "#FFFFFF",  # black
    "#FFFFFF": "#000000",  # white
    "#B20019": "#FFFFFF",  # red
    "#270B65": "#FFFFFF",  # blue
    "#158D34": "#FFFFFF",  # green
    "#D38A15": "#FFFFFF",  # orange
    "#F5F105": "#000000",  # yellow
    "#B7AC8B": "#FFFFFF",  # gold
}


def get_items_from_child_dir(loc: str):
    current_directory = os.getcwd()
    current_directory = f"{current_directory}/{loc}"
    items = os.listdir(current_directory)
    return items


FONTS = get_items_from_child_dir("app/feature/fonts")


def vertical_dimensions(*, lines, font, font_size, gaps):
    lines.reverse()
    max_char_width = 0
    for line in lines:
        for char in line:
            max_char_width = max(max_char_width, get_text_dimensions(
                char, font)[2])

    # first max function gets height of every letter and gets the max,
```

```
    # second max function gets the max of the length of each line
    longest_text = max(len(line) for line in lines)
    total_line_height = (
        max(get_text_dimensions(line, font)[3] - get_text_dimensions(
            line, font)[1] for line in lines) * longest_text
    )
    padding = 4 * gaps
    width = (font_size + gaps) * len(lines) + padding
    height = (font_size + gaps) * longest_text + padding
    return width, height


def horizontal_dimensions(*, lines, font, font_size, gaps):
    padding = 4 * gaps
    width = max(get_text_dimensions(line, font)[2] for line in lines) +
        padding
    height = (font_size + gaps) * len(lines) + padding
    return width, height


def get_text_dimensions(text, font):
    draw = ImageDraw.Draw(Image.new("RGBA", (1, 1), color=(0, 0, 0, 0))
        )
    return draw.textbbox((0, 0), text, font)


def shrink_text(width, height, img, max_width, max_height):
    aspect_ratio = width / height

    if width > max_width or height > max_height:
        if width > height:
            new_width = max_width
            new_height = int(new_width / aspect_ratio)
        else:
            new_height = max_height
            new_width = int(new_height * aspect_ratio)

        img = img.resize((new_width, new_height), Image.ANTIALIAS)
    return img


def preview_text(
    text: str,
    is_vertical: bool,
    color: str,
    bordered: bool,
    PPI: int,
    font_size: int,
    strokewidth: int,
    gaps=3,
):
    height = 0
```

```
    width = 0
    if not bordered:
        strokewidth = 0
    current_directory = os.path.dirname(os.path.abspath(__file__)) + "/fonts"
    font_path = os.path.join(current_directory, FONTS[1])

    font = ImageFont.truetype(font_path, size=font_size)

    font_list = get_items_from_child_dir("app/feature/fonts")

    lines = text.split("\n")
    if is_vertical:
        gaps = gaps // 2
        width, height = vertical_dimensions(
            lines=lines,
            font=font,
            font_size=font_size,
            gaps=gaps,
        )
    else:
        width, height = horizontal_dimensions(lines=lines, font=font, font_size=font_size, gaps=gaps)

    img = Image.new("RGBA", (width, height), color=(0, 0, 0, 0))
    draw = ImageDraw.Draw(img)

    if is_vertical:
        write_vertical(
            lines=lines,
            draw=draw,
            font=font,
            font_size=font_size,
            gaps=gaps,
            color=color,
            outline_color="white",
            strokewidth=strokewidth,
        )
    else:
        write_horizontal(
            lines=lines,
            draw=draw,
            font=font,
            font_size=font_size,
            gaps=gaps,
            color=color,
            outline_color="white",
            strokewidth=strokewidth,
        )

    img_data = io.BytesIO()
    dpi = PPI / 2.54
```

```
    img.save(img_data, format="PNG")
    img_data.seek(0)

    return StreamingResponse(img_data, media_type="image/png")


def write_vertical(*, lines, draw, font, font_size, gaps, color, outline_color, strokewidth):
    padding = gaps * 2
    gaps = gaps // 2
    y = gaps
    x = padding
    # char_width = max(get_text_dimensions(char, font)[2] for line in lines for char in line)
    for idx, line in enumerate(lines):
        box_w = max(get_text_dimensions(x, font)[2] - get_text_dimensions(x, font)[0] for x in line)
        for char_idx, char in enumerate(line):
            if char == " ":
                char = " "
            alignment_val = (box_w - get_text_dimensions(char, font)[2]) / 2
            draw.text(
                (x + alignment_val, y),
                char,
                font=font,
                fill=color,
                stroke_fill=outline_color,
                stroke_width=strokewidth,
            )
            y += font_size
        x += font_size + gaps
        y = padding


def write_horizontal(*, lines, draw, font, font_size, gaps, color, outline_color, strokewidth):
    padding = gaps * 2
    y = gaps
    for line in lines:
        x = padding
        draw.text(
            (x, y),
            line,
            font=font,
            fill=color,
            spacing=gaps,
            stroke_fill=outline_color,
            stroke_width=strokewidth,
        )
        y += font_size + gaps

```

```python

def pillow_text_generator(
    text: str,
    is_vertical: bool = True,
    bordered: bool = True,
    PPI: int = 300,
    font_size: int = 90,
    strokewidth: int = 3,
    job_number: int = None,
    text_type: str = "text_title",
    gaps=3,
    element=None,
) -> t.List[str]:
    if text_type == "text_title":
        start_index = -23
    else:
        start_index = -63
    result_list = []
    for each_font in FONTS:
        for color, outline_color in color_dict.items():
            if not bordered:
                strokewidth = 0
            current_directory = f"{os.path.dirname(os.path.abspath(__file__))}/fonts"
            font_location = os.path.join(current_directory, each_font)
            font = ImageFont.truetype(font_location, size=font_size)

            lines = text.split("\n")
            width = height = 0
            if is_vertical:
                width, height = vertical_dimensions(
                    lines=lines,
                    font=font,
                    font_size=font_size,
                    gaps=gaps,
                )
            else:
                width, height = horizontal_dimensions(lines=lines, font=font, font_size=font_size, gaps=gaps)

            img = Image.new("RGBA", (width, height), color=(0, 0, 0, 0))
            draw = ImageDraw.Draw(img)

            if is_vertical:
                write_vertical(
                    lines=lines,
                    draw=draw,
                    font=font,
                    font_size=font_size,
                    gaps=gaps,
                    color=color,
```

```
                    outline_color=outline_color,
                    strokewidth=strokewidth,
                )
            else:
                write_horizontal(
                    lines=lines,
                    draw=draw,
                    font=font,
                    font_size=font_size,
                    gaps=gaps,
                    color=color,
                    outline_color=outline_color,
                    strokewidth=strokewidth,
                )

            saving_location = f"/jobs/{job_number}/input/png"
            text = text.replace("␣", "_")
            font_name = each_font.replace(".tff", "")
            file_name = f"{text_type}_created_{color}_{font_name}.png"

            if settings.IS_LOCAL:
                directory = f"/static{saving_location}"
                os.makedirs(directory, exist_ok=True)
                saved_location = f"{directory}/{file_name}"
                img.save(saved_location, "PNG", dpi=(PPI, PPI))
                material_file = GeneratedMaterial(
                    id=start_index,
                    png_width=img.width,
                    png_height=img.height,
                    index=job_number,
                    png_file_path=saved_location,
                    job_id=job_number,
                    element_id=element.id,
                    element_name=element.name,
                    element=element,
                    font_name=font_name,
                    color=color,
                )
                result_list.append(material_file)
            else:
                img_data = io.BytesIO()
                img.save(img_data, format="PNG")
                img_data.seek(0)
                saved_location = f"/static{saving_location}/{file_name}
                    "
                upload_file_object(img_data, saved_location[1:])
                material_file = GeneratedMaterial(
                    id=start_index,
                    png_width=img.width,
                    png_height=img.height,
                    index=job_number,
                    png_file_path=saved_location,
```

```
                        job_id=job_number,
                        element_id=element.id,
                        element_name=element.name,
                        element=element,
                        font_name=font_name,
                        color=color,
                    )
                    result_list.append(material_file)
                start_index -= 1

    return result_list


def randomly_choose_text(generated_materials: t.List[GeneratedMaterial
    ], registered_materials_from_db):
    black = "black"
    other = "other"
    db_item = "db_item"
    material_map = {
        black: [],
        other: [],
    }

    for item in generated_materials:
        if "#000000".lower() in str(item.png_file_path).lower():
            material_map[black].append(item)
        else:
            material_map[other].append(item)
    if registered_materials_from_db:
        material_map[db_item] = []
        for each_item in registered_materials_from_db:
            material_map[db_item].append(each_item)
    item_list = []
    item_weight = []
    if registered_materials_from_db:
        item_list.extend([black, other, db_item])
        item_weight.extend([0.7 * 1 / 3, 0.3 * 1 / 3, 1 / 3])
    else:
        item_list.extend([black, other])
        item_weight.extend([0.7, 0.3])

    chosen_item: GeneratedMaterial = random.choices(item_list, weights=
        item_weight)[0]
    chosen_value: GeneratedMaterial = random.choice(material_map[
        chosen_item])

    return chosen_value


def get_font_and_color(ids, titles):
    font, color = None, None
    for id in ids:
```

```
        for title in titles:
            if not isinstance(title, GeneratedMaterial):
                continue
            if id == title.id:
                font, color = title.font_name, title.color
                break
        if font is not None:
            break
    return font or "", color or ""
```

Код B.1: Text-ээс PNG зураг үүсгэдэг script

B.1.2 Business logic script

```
def create_combination_materials(db: Session, job_id: int,
    label_generation_number: int, layout_pattern_number: int):
    job = crud.jobs.get(db=db, id=job_id)
    materials = []
    info_materials = None

    # text to image process
    start_time = time.time()

    title_chosen: List[GeneratedMaterial] = []
    copy_chosen: List[GeneratedMaterial] = []

    generated_titles: List[GeneratedMaterial] = []
    generated_copies: List[GeneratedMaterial] = []
    registered_titles_from_db: List[RegisteredMaterials] = []
    registered_copies_from_db: List[RegisteredMaterials] = []

    unique_pattern_num = label_generation_number //
        layout_pattern_number

    if job.text_title_string:
        registered_titles_from_db = crud.registered_materials.
            search_registered_material_by_text(
            db=db, text=job.text_title_string
        )
        title_elm = crud.labels.get_by_name(db=db, name="text_title")
        generated_titles = pillow_text_generator(
            text=job.text_title_string,
            is_vertical=bool(job.text_title_is_vertical),
            job_number=job_id,
            text_type="text_title",
            element=title_elm,
        )

    if job.text_copy_string:
        registered_copies_from_db = crud.registered_materials.
            search_registered_material_by_text(
            db=db, text=job.text_copy_string
```

```
        )
        copy_elm = crud.labels.get_by_name(db=db, name="text_copy")
        generated_copies = pillow_text_generator(
            text=job.text_copy_string,
            is_vertical=bool(job.text_copy_is_vertical),
            text_type="text_copy",
            job_number=job_id,
            element=copy_elm,
        )

    logging.info("---␣execution␣time␣to␣text␣to␣image:␣%s␣seconds␣---"
       % (time.time() - start_time))

    if job.job_type == 0:
        materials = crud.materials.get_multi_by_job_id(db=db, job_id=
           job_id)
    elif job.job_type == 1:
        # execution start time
        start_time = time.time()

        # Get materials from tag ids
        job_tags: List[TagList] = []
        tag_id_records = crud.input_tag_record.get_multi_job_id(db=db,
           job_id=job.id)
        for tag_id_record in tag_id_records:
            tag = crud.tags.get(db=db, id=tag_id_record.tag_list_id)
            job_tags.append(tag)
        recommended_tags = crud.recommended_tag_record.get_multi_job_id
           (db=db, job_id=job.id)
        for tag_id_record in recommended_tags:
            tag = crud.tags.get(db=db, id=tag_id_record.tag_list_id)
            job_tags.append(tag)

        logging.info({"Combination␣tag␣ids": job_tags})
        tag_material_relations: List[TagMaterialRelation] = []
        for tag_id in job_tags:
            tag_material_relation = crud.tag_material_relation.
               get_multi_by_tag_id(db=db, tag_id=tag_id.id)
            tag_material_relations = tag_material_relations +
               tag_material_relation
        for tag_material_relation in tag_material_relations:
            material = crud.registered_materials.get(db=db, id=
               tag_material_relation.registered_material_id)
            materials.append(material)

        # Exclude duplicates
        materials = list(set(materials))

        logging.info("---␣execution␣time␣to␣fetch␣materials␣by␣tag:␣%s␣
           seconds␣---" % (time.time() - start_time))
        for material in materials:
            logging.info(
```

```
                [
                    material.id,
                    material.element.name,
                    [[tag.tag_list.tag_types.name, tag.tag_list.name]
                       for tag in material.tag_material_relation],
                ]
            )

        # Filter materials by tag relevancy
        start_time = time.time()
        materials = filter_materials_by_tag(materials, job_tags)
        logging.info("--- execution time to filter materials by tags: %
           s seconds ---" % (time.time() - start_time))

        # execution start time
        start_time = time.time()

        # Get info materials
        info_materials = get_info_materials(db=db, job=job)
        materials += info_materials

        logging.info({"materials before pixta": len(materials)})

        # Get images from pixta
        keywords = extract_keywords(job.text_title_string)
        pixta_back_images = get_images(db=db, job_id=job.id, keywords=
           keywords, is_background=True)
        pixta_main_images = get_images(db=db, job_id=job.id, keywords=
           keywords)
        materials += pixta_back_images
        materials += pixta_main_images

        logging.info({"last material": materials[-1]})
        if job.text_title_string:
            materials = [material for material in materials if material
               .element_id != title_elm.id]

        if job.text_copy_string:
            materials = [material for material in materials if material
               .element_id != copy_elm.id]

        logging.info({"materials count": len(materials)})

        logging.info("--- execution time to fetch material infos: %s
           seconds ---" % (time.time() - start_time))

        # execution start time
        start_time = time.time()

        logging.info([job_id, [[tag.tag_types.name, tag.name] for tag
           in job_tags]])
        for material in materials:
```

```
123             if material.id < 0:
124                 continue
125             logging.info(
126                 [
127                     material.id,
128                     material.element.name,
129                     [[tag.tag_list.tag_types.name, tag.tag_list.name]
                            for tag in material.tag_material_relation],
130                 ]
131             )
132 
133         logging.info({"Combination materials: ": materials})
134         if len(materials) < 1:
135             return
136         # Index materials in one label
137         materials_indexed = []
138         label_map = {}
139         for material in materials:
140             if material.element.name in label_map:
141                 label_map[material.element.name].append(material.
                        __dict__)
142             else:
143                 label_map[material.element.name] = [material.__dict__]
144 
145         for value in label_map.values():
146             for idx, item in enumerate(value):
147                 value[idx] = {**item, "index": idx + 1}
148             materials_indexed += value
149 
150         logging.info("--- execution time to index materials in one
                label: %s seconds ---" % (time.time() - start_time))
151         logging.info({"materials_indexed": materials_indexed})
152         # execution start time
153         start_time = time.time()
154 
155         # Copy material pngs to job input folder
156         png_folder = f"/static/jobs/{job_id}/input/png"
157         create_folder(png_folder)
158         for material in materials_indexed:
159             if material["id"] < -12:
160                 continue
161             file_name = f"{material['element'].name}_{material['index
                    ']}.png"
162             file_path = os.path.join(png_folder, file_name)
163 
164             if not settings.IS_LOCAL:
165                 try:
166                     s3 = boto3.resource("s3")
167                     copy_source = {
168                         "Bucket": bucket_name,
169                         "Key": material["png_file_path"][1:],
170                     }
```

```
                    bucket = s3.Bucket(bucket_name)
                    s3_file_key = file_path[1:]
                    bucket.copy(copy_source, s3_file_key)
                except Exception as e:
                    logging.info({"back␣png␣s3␣copy␣exception": e})
            else:
                shutil.copy2(material["png_file_path"], file_path)

            material["png_file_path"] = file_path

        logging.info("---␣execution␣time␣to␣copy␣materials:␣%s␣seconds␣
           ---" % (time.time() - start_time))

        if not settings.IS_LOCAL:
            os.rmdir(png_folder)

    # execution start time
    start_time = time.time()

    comb_map = {}
    align_map = {}
    for r in materials:
        if r.png_height > r.png_width:
            align_map[r.id] = False
        else:
            align_map[r.id] = True

        if r.element_id in comb_map:
            comb_map[r.element_id].append(r.id)
        else:
            comb_map[r.element_id] = [r.id]
    logging.info(comb_map)

    values = []
    for i in sorted(comb_map):
        values.append(comb_map[i][-10:])

    logging.info(values)
    material_combs = []
    max_combo_num = label_generation_number // layout_pattern_number +
       (
        label_generation_number % layout_pattern_number > 0
    )
    for _ in range(max_combo_num):
        random_title: GeneratedMaterial = None
        random_copy: GeneratedMaterial = None
        if job.text_title_string:
            random_title = randomly_choose_text(
                generated_materials=generated_titles,
                   registered_materials_from_db=
                   registered_titles_from_db
            )
```

```
219             title_chosen.append(random_title)
220             materials_indexed += [random_title.__dict__]
221         if job.text_copy_string:
222             random_copy = randomly_choose_text(
223                 generated_materials=generated_copies,
                        registered_materials_from_db=
                        registered_copies_from_db
224             )
225             copy_chosen.append(random_copy)
226             materials_indexed += [random_copy.__dict__]
227 
228         material_combs.append((random_product(*values, chosen_title=
                random_title, chosen_copy=random_copy)))
229 
230     logging.info({"comb_map": comb_map})
231     logging.info({"material_combs": material_combs})
232     logging.info({"len␣material_combs": len(material_combs)})
233 
234     # execution start time
235     start_time = time.time()
236 
237     labels = []
238     for matid in material_combs[0]:
239         if job.job_type == 0:
240             mat = crud.materials.get(db=db, id=matid)
241             labels.append(mat.element.name)
242         elif job.job_type == 1:
243             mat = crud.registered_materials.get(db=db, id=matid)
244             if mat is None:
245                 continue
246             labels.append(mat.element.name)
247     if info_materials is not None:
248         labels += [x.element.name for x in info_materials]
249     logging.info({"labels": labels})
250 
251     if ("info_barcode" in labels) and ("text_title" in labels):
252         tmp_mat_comb = []
253         barcode_idx = labels.index("info_barcode")
254         text_title_idx = labels.index("text_title")
255         for comb in material_combs:
256             text_long_width_flag = True
257             barcode_long_width_flag = True
258             # for matid in comb:
259             #     mat = crud.materials.get(db=db, id=matid)
260             #     if mat.label.name == "text_title":
261             #         if mat.height > mat.width:
262             #             text_long_width_flag = False
263             #     if mat.label.name == "info_barcode":
264             #         if mat.height > mat.width:
265             #             barcode_long_width_flag = False
266             barcode_matid = comb[barcode_idx]
267             barcode_long_width_flag = align_map[barcode_matid]
```

```
268             text_title_matid = comb[text_title_idx]
269             text_long_width_flag = align_map[text_title_matid]
270             if barcode_long_width_flag == text_long_width_flag:
271                 tmp_mat_comb.append(comb)
272         material_combs = tmp_mat_comb
273
274     # logging.info({"after barcode material_combs": material_combs})
275     logging.info({"after␣barcode␣len␣material_combs": len(
            material_combs)})
276
277     tmp_aligned_comb = []
278     tmp_not_aligned_comb = []
279     labels_to_align = ["text_title", "text_copy", "text_explain"]
280     active_labels_to_align = [lab for lab in labels_to_align if lab in
            labels]
281     active_labels_idx = [labels.index(lab) for lab in
            active_labels_to_align]
282     if len(active_labels_to_align) > 1:
283         for comb in material_combs:
284             rule_align_flags = []
285             for actv_idx in active_labels_idx:
286                 matid = comb[actv_idx]
287                 rule_align_flags.append(align_map[matid])
288             # for matid in comb:
289             # mat = crud.materials.get(db=db, id=matid)
290             # if mat.label.name in active_labels_to_align:
291             #     if mat.height > mat.width:
292             #         rule_align_flags.append(False)
293             #     else:
294             #         rule_align_flags.append(True)
295             if (sum(rule_align_flags) == 0) or (sum(rule_align_flags)
                    == len(active_labels_to_align)):
296                 tmp_aligned_comb.append(comb)
297             else:
298                 tmp_not_aligned_comb.append(comb)
299     else:
300         tmp_aligned_comb = material_combs
301         tmp_not_aligned_comb = []
302
303     # logging.info({"tmp_aligned_comb": tmp_aligned_comb})
304     # logging.info({"tmp_not_aligned_comb": tmp_not_aligned_comb})
305     logging.info({"len␣tmp_aligned_comb": len(tmp_aligned_comb)})
306     logging.info({"len␣tmp_not_aligned_comb": len(tmp_not_aligned_comb)
            })
307
308     aligned_rate = 1
309     aligned_num = int(label_generation_number * aligned_rate)
310
311     if len(tmp_aligned_comb) > aligned_num:
312         tmp_aligned_comb = random.sample(tmp_aligned_comb, aligned_num)
313     tmp_not_aligned_comb = random.sample(tmp_not_aligned_comb, len(
            tmp_not_aligned_comb))
```

```
314     tmp_aligned_comb.extend(tmp_not_aligned_comb)
315
316     material_combs = tmp_aligned_comb[:label_generation_number]
317     logging.info({"last material_combs": material_combs})
318
319     limit = min(len(material_combs), 100)
320     # rounding up the division if there is a remainder
321     combinations = [f"random_{i}" for i in range(1, limit + 1)]
322
323     for comb, mc in zip(combinations, material_combs):
324         # material combo dotor bga title copy 2g avaad font-g ni job
              combosruu hadgalnaa
325         text_title_font, text_title_color = "", ""
326         copy_font, copy_color = "", ""
327
328         if job.text_title_string:
329             text_title_font, text_title_color = get_font_and_color(mc,
                  title_chosen)
330
331         if job.text_copy_string:
332             copy_font, copy_color = get_font_and_color(mc, copy_chosen)
333
334         jc_in = schemas.JobCombinationsCreate(
335             job_id=job_id,
336             comb_name=comb,
337             text_title_font=text_title_font,
338             text_title_color=text_title_color,
339             text_copy_font=copy_font,
340             text_copy_color=copy_color,
341         )
342         crud.job_combinations.create(db=db, obj_in=jc_in)
343
344     job_combs = crud.job_combinations.get_multi_job_id(db=db, job_id=
          job_id)
345
346     data = []
347     combinations = []
348
349     for jc, material_ids in zip(job_combs, material_combs):
350         combination_materials = []
351         for material_id in material_ids:
352             if job.job_type == 0:
353                 data.append(JobCombinationMaterials(job_comb_id=jc.id,
                      material_uploaded_id=material_id))
354             elif job.job_type == 1:
355                 if material_id > 0:
356                     data.append(JobCombinationMaterialsSelected(
                          job_comb_id=jc.id, registered_material_id=
                          material_id))
357
358                 # Return indexed materials
359
```

```
360                 material = list(filter(lambda x: x["id"] == material_id
                        , materials_indexed))[0]
361
362                 combination_material_data = (
363                     jc.id,
364                     material["png_width"],
365                     material["png_height"],
366                     material["element"].name,
367                     material["png_file_path"],
368                     material["index"],
369                 )
370                 combination_materials.append(combination_material_data)
371
372         combinations.append((combination_materials, jc.id))
373
374     db.bulk_save_objects(data)
375     db.commit()
376
377     logging.info("---␣execution␣time␣to␣create␣combs:␣%s␣seconds␣---" %
            (time.time() - start_time))
378
379     return combinations if job.job_type == 1 else None
```

Код B.2: Combination үүсгэх бизнес логик код