

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

УДИРТГАЛ	1
1. БАЙГУУЛЛАГЫН ТАНИЛЦУУЛГА	3
1.1 Товч танилцуулга	3
1.2 Ямар үйлчилгээ үзүүлдэг вэ?	3
2. СИСТЕМИЙН ШААРДЛАГА	4
2.1 Танилцуулга	4
2.2 Функционал шаардлагууд	4
2.3 Функционал бус шаардлагууд	5
3. АШИГЛАХ ТЕХНОЛОГИУД	6
3.1 Back-end талын технологиуд	6
3.2 Front-end талын технологиуд	6
3.3 Бусад	7
4. ХЭРЭГЖҮҮЛЭЛТ	8
4.1 Текстээс зураг үүсгэх	8
4.2 Business logic-г хийх	11
5. НЭМЭЛТ СУДАЛГАА	15
5.1 JSON WEB TOKEN	15
6. ДҮГНЭЛТ	16
6.1 Үр дүнгийн тайлан	16
НОМ ЗҮЙ	17
ХАВСРАЛТ	18
А. ЖИШЭЭНҮҮД	18
В. КОДЫН ХЭРЭГЖҮҮЛЭЛТ	21
В.1 Python	21

ЗУРГИЙН ЖАГСААЛТ

4.1	Алгоритм	12
4.2	Алгоритм-2	14
A.1	Жишээ-1	18
A.2	Жишээ-2	19
A.3	Жишээ-3	20

ХҮСНЭГТИЙН ЖАГСААЛТ

1	Дадлагын төлөвлөгөө	2
---	---------------------------	---

Кодын жагсаалт

4.1	Table-рүү оруулсан өөрчлөлт	8
4.2	Зураг буцаан user-лүү илгээх endpoint	9
4.3	Endpoint дуудах function	10
4.4	Pydantic Validator	11
4.5	Сонголт хийх код	13
B.1	Text-ээс PNG зураг үүсгэдэг script	21
B.2	Combination үүсгэх бизнес логик код	28

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 групп 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 материалуудыг 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.

2.3. ФУНКЦИОНАЛ БУС ШААРДЛАГУУД БҮЛЭГ 2. СИСТЕМИЙН ШААРДЛАГА

- **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 (Scalability):** 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.
- **Maintainability:** 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 ASGI¹ framework ба үзүүлэлтийн хувьд nodejs эсвэл go зэргийн үзүүлдэг маш өндөр үзүүлэлтрүү дөхдөг билээ.

- Asynchronous байдлаар ажиллаж чаддаг байдал нь сру-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 ORM² ба давуу тал нь хөгжүүлэгч шууд database-тай харьцах биш python-г ашиглан харьцах боломжийг олгоно, ингэснээр database-н схемийг өөрчлөхөд 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 олгодог гадна бусад routing, local storage, гэх мэт хөгжүүлэгчдийн өөрсдөө тохируулдаг зүйлсийг цаанаас нь шийдэж өгсөн байдаг.

¹Asynchronous Server Gateway Interface

²Object 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 материалуудыг байршуулах боломжтой болно.

3.3.4 *Dockerizing*

Орчин үеийн нэгэн гайхалтай технологи бол контейнерчлах 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 хэмжээний зай, нөөц ашигладаг.

3.3.5 *CI/CD*

Мөн сүүлийн үед маш их өргөн түгж байгаа ойлголт бол Continuous Integration/Continuous Deployment. 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

```
1 alembic revision --autogenerate -m "your_commit_message"
```

гэсэн командын ашиглан өөрчлөлт оруулах мэдээллийг үүсгэн. Дараа нь

```
1 alembic upgrade head
```

комманд хийснээр Датабазын модел бүрэн өөрчлөгдөнө.

```
1 class User(Base):
2     # ...
3     # Other table information
4
5     text_title_string = Column(String(255), nullable=False, default="")
6     text_title_color = Column(String(255), nullable=False, default="#000000")
7     text_title_font_size = Column(Integer, nullable=False, default=0)
8     text_title_font_name = Column(String(255), nullable=False, default="")
9     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**

```
1 from fastapi import APIRouter, Depends, status
2 from sqlalchemy.orm import Session
3 from app import schemas
4 from app.feature.pillow import pillow_text_generator, preview_text
5 from app.models.user import User
6 from app.v1 import deps
7 router = APIRouter()
8
9 @router.post("/", status_code=status.HTTP_201_CREATED)
10 def pillow_text(
11     *,
12     obj_in: schemas.PillowBase,
13     db: Session = Depends(deps.get_db),
14     _: User = Depends(deps.get_current_user),
15 ):
16     return preview_text(
17         text=obj_in.text,
18         is_vertical=obj_in.is_vertical,
19         color=obj_in.color,
```

```

20     font_size=obj_in.font_size,
21     strokewidth=obj_in.stroke_width,
22     bordered=obj_in.bordered,
23     PPI=obj_in.PPI,
24 )

```

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.

```

1  async preview_text_to_image({ _ }, { data }) {
2      const result = await this.$axios.post(`text_image/`, data, {
3          responseType: 'arraybuffer',
4      })
5      if (result.status === 200) {
6          const b64 = btoa(String.fromCharCode(...new Uint8Array(result.
7              data)))
8          const imgData = 'data:' + result.headers['content-type'] + ';'
9              base64,' + b64
10         return imgData
11     } else {
12         console.log('error')
13     }
14 },

```


Код 4.3: Endpoint дуудах function

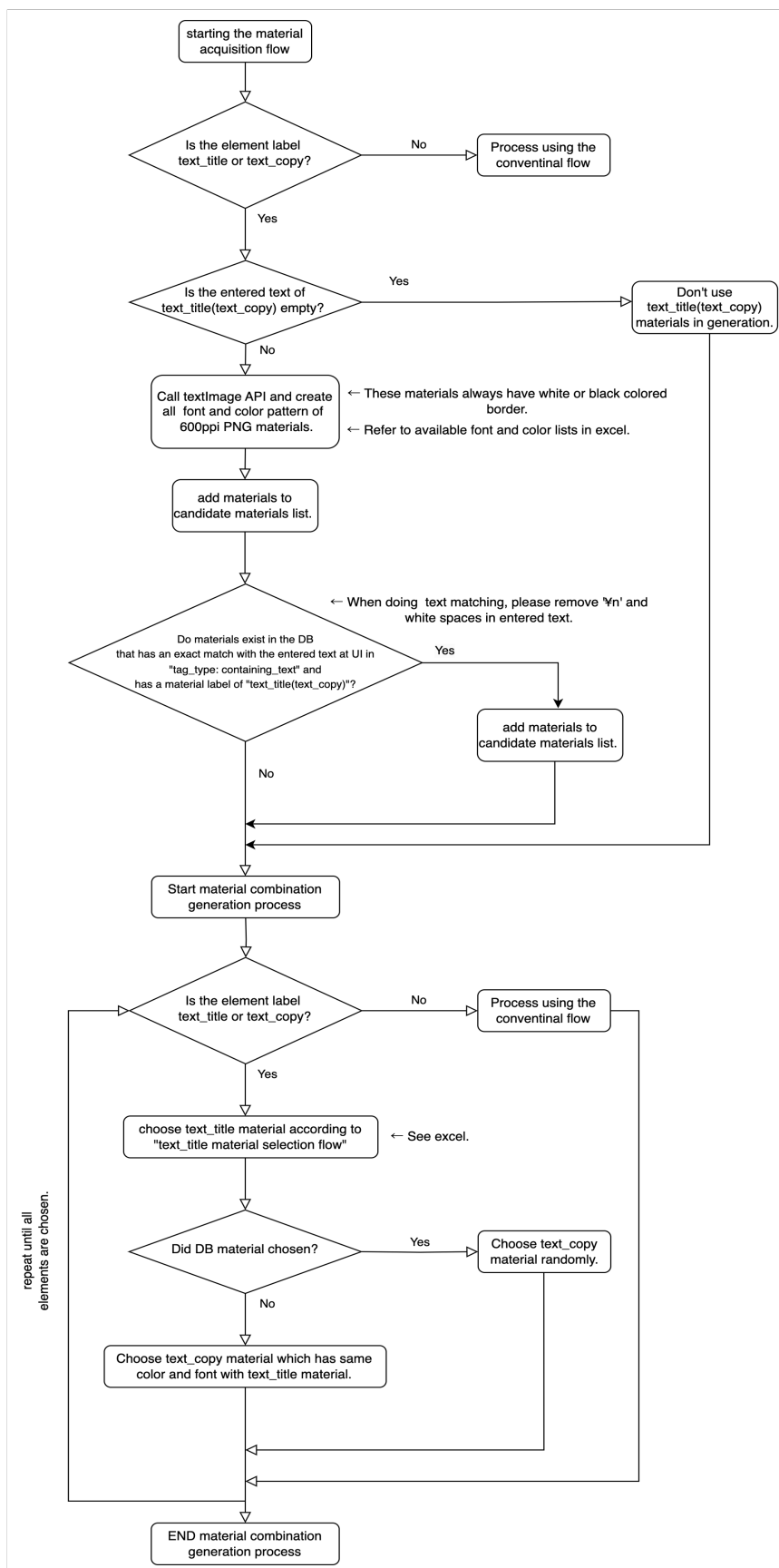
4.2 Business logic-г хийх

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.

```
1  from pydantic import BaseModel
2
3  class PillowBase(BaseModel):
4
5      text: str
6
7      font_path: str
8
9      is_vertical: bool = False
10
11     font_size: Optional[int] = 90
12
13     color: Optional[str] = "#000000"
14
15     bordered: Optional[bool] = True
16
17     stroke_color: Optional[str] = "#ffffff"
18
19     stroke_width: Optional[int] = 3
20
21     gaps: Optional[int] = 3
22
23     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.



4.2.1 Алгоритмын тайлбар

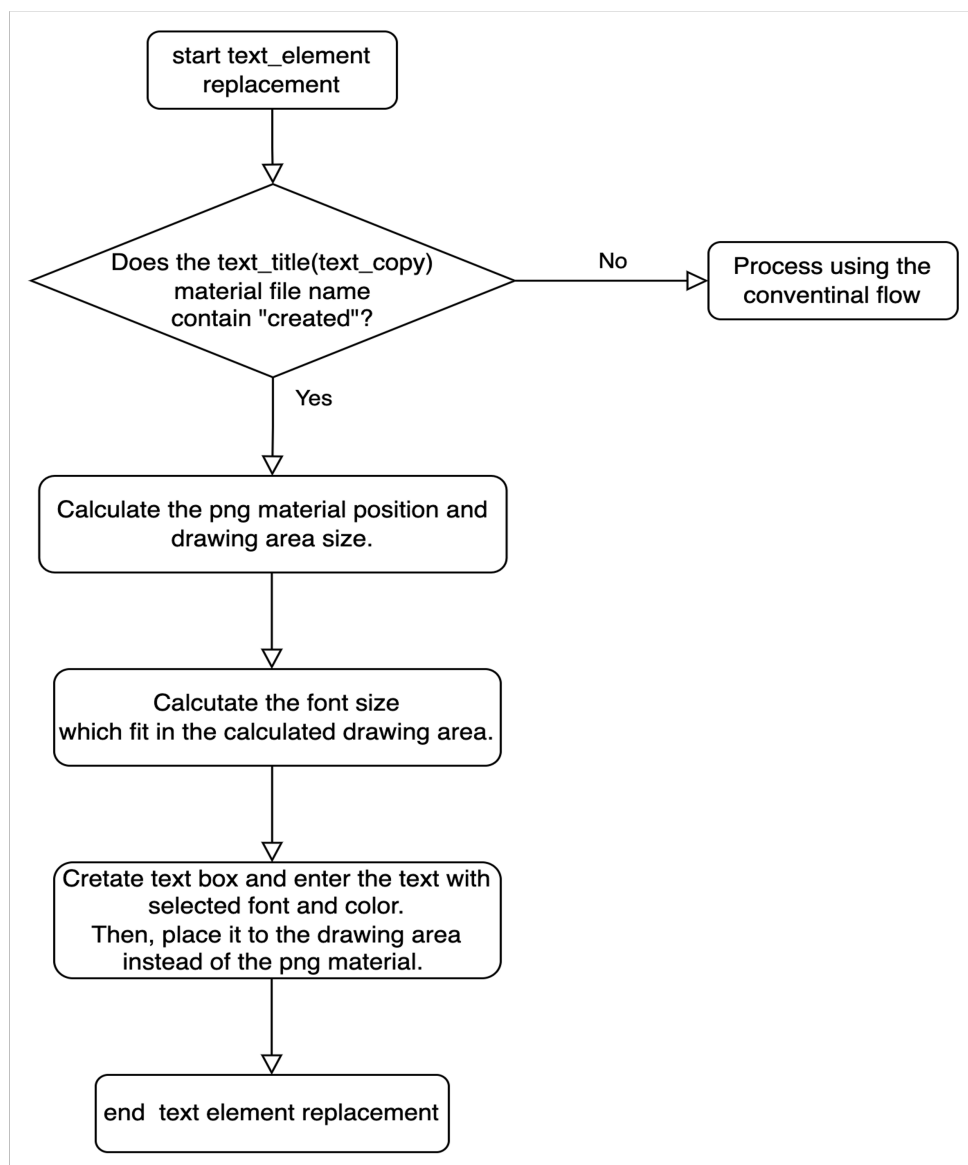
Дээрх зурганд харуулсан Алгоритмыг ерөнхийд нь тайлбарлавал, хэрэглэгчийн зураг үүсгэхийг хүссэн тексттэй адилхан агуулгатай өгөгдөл database дээр байгаа эсэхийг шалгах юм. Хэрэв тийм өгөгдөл олдвол тухайн өгөгдлийн цаашид үүсгэх шошгонууд дээрээ жин өгч магадлал тооцон ашиглах юм. Ингэхдээ текстээс зураг үүсгэдэг алгоритмыг ашиглан 8 өнгө 3 фонт дээр төрөл бүрийн зурагнууд үүсгэж түүнийгээ database дээр олдсон зурагнуудтай нийлүүлэн магадлал тооцно.

```
1  if registered_materials_from_db:
2      item_list.extend([black, other, db_item])
3      item_weight.extend([0.7 * 1 / 3, 0.3 * 1 / 3, 1 / 3])
4  else:
5      item_list.extend([black, other])
6      item_weight.extend([0.7, 0.3])
```

Код 4.5: Сонголт хийх код

4.2.2 Ажиллах дараалал

1. Хэрэглэгчийн оруулсан текстээс \n ялгаж хэдэн мөр байхыг гаргана.
2. Хэрэглэгчийн оруулсан фонт мөрийн тоо, нэг мөр дэх үсгийн тооноос хамаарч өөр өөр байх хоосон зургуудийг үүсгэнэ.
3. Хоосон зургуудруугаа текстийг нэг нэгээр нь нэмнэ.



Зураг 4.2: Алгоритм-2

5. НЭМЭЛТ СУДАЛГАА

5.1 JSON WEB TOKEN

5.1.1 Тайлбар

JSON Web Token (JWT) нь нээлттэй стандарт (RFC 7519) бөгөөд мэдээллийг JSON объект болгон талуудын хооронд найдвартай дамжуулах шалгарсан арга юм. Дамжуулагдах мэдээлэл нь server дээр гарын үсгээр (Signature) баталгаажсан байх тул итгэж болохуйц юм. JWT-г баталгаажуулахдаа **hash** хийх hmac алгоритм эсвэл нээлттэй хаалттай (public/private) түлхүүрийн хослолыг ашиглаж болдог (RSA эсвэл ECDSA).

5.1.2 Бүрэлдэхүүн

- **Толгой хэсэг:** Толгой хэсэг нь ихэвчлэн хоёр хэсгээс бүрдэнэ: токенын төрөл (JWT) ба HMAC SHA256 эсвэл RSA зэрэг ашиглаж буй гарын үсэг зурах алгоритм.
- **Их бие (payload):** Их бие нь токенын хэрэглэгчийн мэдээллийг агуулдаг. Хэрэглэгчийн мэдээллийг хадгалахын тулд хэрэглэгчийн ID, хэрэглэгчийн нэр, хэрэглэгчийн эрхийг хадгалдаг. Гэхдээ энэ хэсэгт нууц мэдээлэл хадгалж болохгүй, нэмэлт мэдээлэл л хадгална, яагаад гэвэл энэ нь хэн ч харж болох base64 string байдаг.
- **Гарын үсэг (signature):** Гарын үсгийн хэсгийг үүсгэхийн тулд их бие, толгой хоёрийг авч толгой хэсэгт заасан алгоритмыг ашиглан гарын үсэг зурна. Гарын үсэг зурна гэдэг нь ерөнхийдөө энэхүү мэдээллийг ХАЙШ хийх ба дараа нь үүссэн ХАЙШ нь үүнийг баталгаажуулахад ашиглагдах юм.

6. ДҮГНЭЛТ

6.1 Үр дүнгийн тайлан

Миний бие Dentsu Data Artist Mongol-д мэргэжлийн дадлагыг амжилттай гүйцэтгэж дуусгалаа. Уг хугацаанд хичээлийн хүрээнд үзсэн онолын ойлголтуудыг практик дээр туршиж, хэрэгжүүлсэн ба хөгжүүлэлт голчилсон технологийн компанийн ерөнхий үйл ажиллагаа, баг хооронд зохицон ажиллах чадвар, хөгжүүлэлтийн шинэ арга барилуудыг амжилттай эзэмшсэн гэж дүгнэж байна.

Back-end, Front-end-ийн хамгийн орчин үеийн технологиудтай танилцаж, машин сургалтын аргуудыг ашигладаг төсөл дээр ажиллаа. Мөн бие даан асуудлыг шийдэхийн тулд алхам дэс дараатай асуудлыг шийдвэрлэх мөн ашиглаж буй сан, технологийнхоо гарын авлага буюу documentation-тай илүү сайн танилцаж уг технологийнхоо цаана нь буй концептийг хялбараар ойлгох гэх мэт чадваруудыг эзэмшсэн.

Энэхүү дадлагын хугацаанд өөрийн ямар чиглэлд сонирхолтой, цаашид аль чиглэлрүү гүнзгийрэх талаараа үнэтэй туршлага цуглууллаа. Дадлагын хугацаанд хамгийн сонирхолтой байсан хэсэг нь бие даан, мэдээллийг нууцлах, мэдээллийг хадгалах технологиуд болон CI/CD чиглэлийн орчин үеийн аргыг судалж сервер дээр ажиллах байлаа.

Bibliography

- [1] introduction to JSON Web Token, <https://jwt.io/introduction>
- [2] Pillow Library Documentation, <https://pillow.readthedocs.io/en/stable/>
- [3] Inserting Images, Share LaTeX, https://www.sharelatex.com/learn/Inserting_Images
- [4] Paragraphs and new lines, Share LaTeX, https://www.sharelatex.com/learn/Paragraphs_and_new_lines
- [5] Bold, italics and underlining, Share LaTeX, https://www.sharelatex.com/learn/Bold,_italics_and_underlining
- [6] Lists, Share LaTeX, <https://www.sharelatex.com/learn/Lists>
- [7] Tables, Share LaTeX, <https://www.sharelatex.com/learn/Tables>

А. ЖИШЭЭНҮҮД



Зураг А.1: Жишээ-1



Зураг А.2: Жишээ-2



Зураг А.3: Жишээ-3

В. КОДЫН ХЭРЭГЖҮҮЛЭЛТ

В.1 Python

В.1.1 Font script

```
1 import io
2 import os
3 import random
4 import typing as t
5 import uuid
6
7 from fastapi.responses import StreamingResponse
8 from PIL import Image, ImageDraw, ImageFont
9
10 from app.core.config import settings
11 from app.schemas.materials import GeneratedMaterial
12 from app.utils.s3_handler import upload_file_object
13
14 color_dict = {
15     "#000000": "#FFFFFF", # black
16     "#FFFFFF": "#000000", # white
17     "#B20019": "#FFFFFF", # red
18     "#270B65": "#FFFFFF", # blue
19     "#158D34": "#FFFFFF", # green
20     "#D38A15": "#FFFFFF", # orange
21     "#F5F105": "#000000", # yellow
22     "#B7AC8B": "#FFFFFF", # gold
23 }
24
25
26 def get_items_from_child_dir(loc: str):
27     current_directory = os.getcwd()
28     current_directory = f"{current_directory}/{loc}"
29     items = os.listdir(current_directory)
30     return items
31
32
33 FONTS = get_items_from_child_dir("app/feature/fonts")
34
35
36 def vertical_dimensions(*, lines, font, font_size, gaps):
37     lines.reverse()
38     max_char_width = 0
39     for line in lines:
40         for char in line:
41             max_char_width = max(max_char_width, get_text_dimensions(
42                 char, font)[2])
43
44     # first max function gets height of every letter and gets the max,
```

```

44     # second max function gets the max of the length of each line
45     longest_text = max(len(line) for line in lines)
46     total_line_height = (
47         max(get_text_dimensions(line, font)[3] - get_text_dimensions(
48             line, font)[1] for line in lines) * longest_text
49     )
50     padding = 4 * gaps
51     width = (font_size + gaps) * len(lines) + padding
52     height = (font_size + gaps) * longest_text + padding
53     return width, height
54
55 def horizontal_dimensions(*, lines, font, font_size, gaps):
56     padding = 4 * gaps
57     width = max(get_text_dimensions(line, font)[2] for line in lines) +
58         padding
59     height = (font_size + gaps) * len(lines) + padding
60     return width, height
61
62 def get_text_dimensions(text, font):
63     draw = ImageDraw.Draw(Image.new("RGBA", (1, 1), color=(0, 0, 0, 0))
64     )
65     return draw.textbbox((0, 0), text, font)
66
67 def shrink_text(width, height, img, max_width, max_height):
68     aspect_ratio = width / height
69
70     if width > max_width or height > max_height:
71         if width > height:
72             new_width = max_width
73             new_height = int(new_width / aspect_ratio)
74         else:
75             new_height = max_height
76             new_width = int(new_height * aspect_ratio)
77
78     img = img.resize((new_width, new_height), Image.ANTIALIAS)
79     return img
80
81
82 def preview_text(
83     text: str,
84     is_vertical: bool,
85     color: str,
86     bordered: bool,
87     PPI: int,
88     font_size: int,
89     strokewidth: int,
90     gaps=3,
91 ):
92     height = 0

```

```

93     width = 0
94     if not bordered:
95         strobewidth = 0
96     current_directory = os.path.dirname(os.path.abspath(__file__)) + "/"
97         fonts"
98     font_path = os.path.join(current_directory, FONTS[1])
99     font = ImageFont.truetype(font_path, size=font_size)
100
101     font_list = get_items_from_child_dir("app/feature/fonts")
102
103     lines = text.split("\n")
104     if is_vertical:
105         gaps = gaps // 2
106         width, height = vertical_dimensions(
107             lines=lines,
108             font=font,
109             font_size=font_size,
110             gaps=gaps,
111         )
112     else:
113         width, height = horizontal_dimensions(lines=lines, font=font,
114             font_size=font_size, gaps=gaps)
115
116     img = Image.new("RGBA", (width, height), color=(0, 0, 0, 0))
117     draw = ImageDraw.Draw(img)
118
119     if is_vertical:
120         write_vertical(
121             lines=lines,
122             draw=draw,
123             font=font,
124             font_size=font_size,
125             gaps=gaps,
126             color=color,
127             outline_color="white",
128             strobewidth=strobewidth,
129         )
130     else:
131         write_horizontal(
132             lines=lines,
133             draw=draw,
134             font=font,
135             font_size=font_size,
136             gaps=gaps,
137             color=color,
138             outline_color="white",
139             strobewidth=strobewidth,
140         )
141
142     img_data = io.BytesIO()
143     dpi = PPI / 2.54

```

```

143     img.save(img_data, format="PNG")
144     img_data.seek(0)
145
146     return StreamingResponse(img_data, media_type="image/png")
147
148
149 def write_vertical(*, lines, draw, font, font_size, gaps, color,
outline_color, strokewidth):
150     padding = gaps * 2
151     gaps = gaps // 2
152     y = gaps
153     x = padding
154     # char_width = max(get_text_dimensions(char, font)[2] for line in
lines for char in line)
155     for idx, line in enumerate(lines):
156         box_w = max(get_text_dimensions(x, font)[2] -
get_text_dimensions(x, font)[0] for x in line)
157         for char_idx, char in enumerate(line):
158             if char == " ":
159                 char = " "
160                 alignment_val = (box_w - get_text_dimensions(char, font)
[2]) / 2
161                 draw.text(
162                     (x + alignment_val, y),
163                     char,
164                     font=font,
165                     fill=color,
166                     stroke_fill=outline_color,
167                     stroke_width=strokewidth,
168                 )
169                 y += font_size
170             x += font_size + gaps
171             y = padding
172
173
174 def write_horizontal(*, lines, draw, font, font_size, gaps, color,
outline_color, strokewidth):
175     padding = gaps * 2
176     y = gaps
177     for line in lines:
178         x = padding
179         draw.text(
180             (x, y),
181             line,
182             font=font,
183             fill=color,
184             spacing=gaps,
185             stroke_fill=outline_color,
186             stroke_width=strokewidth,
187         )
188         y += font_size + gaps
189

```

```

190
191 def pillow_text_generator(
192     text: str,
193     is_vertical: bool = True,
194     bordered: bool = True,
195     PPI: int = 300,
196     font_size: int = 90,
197     strokewidth: int = 3,
198     job_number: int = None,
199     text_type: str = "text_title",
200     gaps=3,
201     element=None,
202 ) -> t.List[str]:
203     if text_type == "text_title":
204         start_index = -23
205     else:
206         start_index = -63
207     result_list = []
208     for each_font in FONTS:
209         for color, outline_color in color_dict.items():
210             if not bordered:
211                 strokewidth = 0
212             current_directory = f"{os.path.dirname(os.path.abspath(
213                 __file__))}/fonts"
214             font_location = os.path.join(current_directory, each_font)
215             font = ImageFont.truetype(font_location, size=font_size)
216
217             lines = text.split("\n")
218             width = height = 0
219             if is_vertical:
220                 width, height = vertical_dimensions(
221                     lines=lines,
222                     font=font,
223                     font_size=font_size,
224                     gaps=gaps,
225                 )
226             else:
227                 width, height = horizontal_dimensions(lines=lines, font
228                 =font, font_size=font_size, gaps=gaps)
229
230             img = Image.new("RGBA", (width, height), color=(0, 0, 0, 0)
231             )
232             draw = ImageDraw.Draw(img)
233
234             if is_vertical:
235                 write_vertical(
236                     lines=lines,
237                     draw=draw,
238                     font=font,
239                     font_size=font_size,
240                     gaps=gaps,
241                     color=color,

```

```

239         outline_color=outline_color,
240         strokewidth=strokewidth,
241     )
242 else:
243     write_horizontal(
244         lines=lines,
245         draw=draw,
246         font=font,
247         font_size=font_size,
248         gaps=gaps,
249         color=color,
250         outline_color=outline_color,
251         strokewidth=strokewidth,
252     )
253
254 saving_location = f"/jobs/{job_number}/input/png"
255 text = text.replace("□", "_")
256 font_name = each_font.replace(".tff", "")
257 file_name = f"{text_type}_created_{color}_{font_name}.png"
258
259 if settings.IS_LOCAL:
260     directory = f"/static{saving_location}"
261     os.makedirs(directory, exist_ok=True)
262     saved_location = f"{directory}/{file_name}"
263     img.save(saved_location, "PNG", dpi=(PPI, PPI))
264     material_file = GeneratedMaterial(
265         id=start_index,
266         png_width=img.width,
267         png_height=img.height,
268         index=job_number,
269         png_file_path=saved_location,
270         job_id=job_number,
271         element_id=element.id,
272         element_name=element.name,
273         element=element,
274         font_name=font_name,
275         color=color,
276     )
277     result_list.append(material_file)
278 else:
279     img_data = io.BytesIO()
280     img.save(img_data, format="PNG")
281     img_data.seek(0)
282     saved_location = f"/static{saving_location}/{file_name}"
283
284     upload_file_object(img_data, saved_location[1:])
285     material_file = GeneratedMaterial(
286         id=start_index,
287         png_width=img.width,
288         png_height=img.height,
289         index=job_number,
290         png_file_path=saved_location,

```



```

290         job_id=job_number,
291         element_id=element.id,
292         element_name=element.name,
293         element=element,
294         font_name=font_name,
295         color=color,
296     )
297     result_list.append(material_file)
298     start_index -= 1
299
300     return result_list
301
302
303 def randomly_choose_text(generated_materials: t.List[GeneratedMaterial
304 ], registered_materials_from_db):
305     black = "black"
306     other = "other"
307     db_item = "db_item"
308     material_map = {
309         black: [],
310         other: [],
311     }
312
313     for item in generated_materials:
314         if "#000000".lower() in str(item.png_file_path).lower():
315             material_map[black].append(item)
316         else:
317             material_map[other].append(item)
318     if registered_materials_from_db:
319         material_map[db_item] = []
320         for each_item in registered_materials_from_db:
321             material_map[db_item].append(each_item)
322     item_list = []
323     item_weight = []
324     if registered_materials_from_db:
325         item_list.extend([black, other, db_item])
326         item_weight.extend([0.7 * 1 / 3, 0.3 * 1 / 3, 1 / 3])
327     else:
328         item_list.extend([black, other])
329         item_weight.extend([0.7, 0.3])
330
331     chosen_item: GeneratedMaterial = random.choices(item_list, weights=
332         item_weight)[0]
333     chosen_value: GeneratedMaterial = random.choice(material_map[
334         chosen_item])
335
336     return chosen_value
337
338
339 def get_font_and_color(ids, titles):
340     font, color = None, None
341     for id in ids:

```

```

339         for title in titles:
340             if not isinstance(title, GeneratedMaterial):
341                 continue
342             if id == title.id:
343                 font, color = title.font_name, title.color
344                 break
345             if font is not None:
346                 break
347         return font or "", color or ""

```

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

B.1.2 Business logic script

```

1  def create_combination_materials(db: Session, job_id: int,
2  label_generation_number: int, layout_pattern_number: int):
3      job = crud.jobs.get(db=db, id=job_id)
4      materials = []
5      info_materials = None
6
7      # text to image process
8      start_time = time.time()
9
10     title_chosen: List[GeneratedMaterial] = []
11     copy_chosen: List[GeneratedMaterial] = []
12
13     generated_titles: List[GeneratedMaterial] = []
14     generated_copies: List[GeneratedMaterial] = []
15     registered_titles_from_db: List[RegisteredMaterials] = []
16     registered_copies_from_db: List[RegisteredMaterials] = []
17
18     unique_pattern_num = label_generation_number //
19         layout_pattern_number
20
21     if job.text_title_string:
22         registered_titles_from_db = crud.registered_materials.
23             search_registered_material_by_text(
24                 db=db, text=job.text_title_string
25             )
26         title_elm = crud.labels.get_by_name(db=db, name="text_title")
27         generated_titles = pillow_text_generator(
28             text=job.text_title_string,
29             is_vertical=bool(job.text_title_is_vertical),
30             job_number=job_id,
31             text_type="text_title",
32             element=title_elm,
33         )
34
35     if job.text_copy_string:
36         registered_copies_from_db = crud.registered_materials.
37             search_registered_material_by_text(
38                 db=db, text=job.text_copy_string

```

```

35     )
36     copy_elm = crud.labels.get_by_name(db=db, name="text_copy")
37     generated_copies = pillow_text_generator(
38         text=job.text_copy_string,
39         is_vertical=bool(job.text_copy_is_vertical),
40         text_type="text_copy",
41         job_number=job_id,
42         element=copy_elm,
43     )
44
45     logging.info("---_execution_time_to_text_to_image:_%s_seconds_---"
46                 % (time.time() - start_time))
47
48     if job.job_type == 0:
49         materials = crud.materials.get_multi_by_job_id(db=db, job_id=
50             job_id)
51     elif job.job_type == 1:
52         # execution start time
53         start_time = time.time()
54
55         # Get materials from tag ids
56         job_tags: List[TagList] = []
57         tag_id_records = crud.input_tag_record.get_multi_job_id(db=db,
58             job_id=job.id)
59         for tag_id_record in tag_id_records:
60             tag = crud.tags.get(db=db, id=tag_id_record.tag_list_id)
61             job_tags.append(tag)
62         recommended_tags = crud.recommended_tag_record.get_multi_job_id
63             (db=db, job_id=job.id)
64         for tag_id_record in recommended_tags:
65             tag = crud.tags.get(db=db, id=tag_id_record.tag_list_id)
66             job_tags.append(tag)
67
68         logging.info({"Combination_tag_ids": job_tags})
69         tag_material_relations: List[TagMaterialRelation] = []
70         for tag_id in job_tags:
71             tag_material_relation = crud.tag_material_relation.
72                 get_multi_by_tag_id(db=db, tag_id=tag_id.id)
73             tag_material_relations = tag_material_relations +
74                 tag_material_relation
75         for tag_material_relation in tag_material_relations:
76             material = crud.registered_materials.get(db=db, id=
77                 tag_material_relation.registered_material_id)
78             materials.append(material)
79
80         # Exclude duplicates
81         materials = list(set(materials))
82
83         logging.info("---_execution_time_to_fetch_materials_by_tag:_%s_
84             seconds_---" % (time.time() - start_time))
85         for material in materials:
86             logging.info(

```

```

79         [
80             material.id,
81             material.element.name,
82             [[tag.tag_list.tag_types.name, tag.tag_list.name]
83              for tag in material.tag_material_relation],
84         ]
85     )
86
87     # Filter materials by tag relevancy
88     start_time = time.time()
89     materials = filter_materials_by_tag(materials, job_tags)
90     logging.info("---_execution_time_to_filter_materials_by_tags:_%
91                 s_seconds_---" % (time.time() - start_time))
92
93     # execution start time
94     start_time = time.time()
95
96     # Get info materials
97     info_materials = get_info_materials(db=db, job=job)
98     materials += info_materials
99
100     logging.info({"materials_before_pixta": len(materials)})
101
102     # Get images from pixta
103     keywords = extract_keywords(job.text_title_string)
104     pixta_back_images = get_images(db=db, job_id=job.id, keywords=
105     keywords, is_background=True)
106     pixta_main_images = get_images(db=db, job_id=job.id, keywords=
107     keywords)
108     materials += pixta_back_images
109     materials += pixta_main_images
110
111     logging.info({"last_material": materials[-1]})
112     if job.text_title_string:
113         materials = [material for material in materials if material
114                     .element_id != title_elm.id]
115
116     if job.text_copy_string:
117         materials = [material for material in materials if material
118                     .element_id != copy_elm.id]
119
120     logging.info({"materials_count": len(materials)})
121
122     logging.info("---_execution_time_to_fetch_material_infos:_%s_
123                 seconds_---" % (time.time() - start_time))
124
125     # execution start time
126     start_time = time.time()
127
128     logging.info([job_id, [[tag.tag_types.name, tag.name] for tag
129                             in job_tags]])
130     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]
130                  for tag in material.tag_material_relation],
131             ]
132         )
133
134     logging.info({"Combination_materials": materials})
135     if len(materials) < 1:
136         return
137     # Index materials in one label
138     materials_indexed = []
139     label_map = {}
140     for material in materials:
141         if material.element.name in label_map:
142             label_map[material.element.name].append(material.
143                 __dict__)
144         else:
145             label_map[material.element.name] = [material.__dict__]
146
147     for value in label_map.values():
148         for idx, item in enumerate(value):
149             value[idx] = {**item, "index": idx + 1}
150     materials_indexed += value
151
152     logging.info("--- execution time to index materials in one
153         label: %s seconds ---" % (time.time() - start_time))
154     logging.info({"materials_indexed": materials_indexed})
155     # execution start time
156     start_time = time.time()
157
158     # Copy material pngs to job input folder
159     png_folder = f"/static/jobs/{job_id}/input/png"
160     create_folder(png_folder)
161     for material in materials_indexed:
162         if material["id"] < -12:
163             continue
164         file_name = f"{material['element'].name}_{material['index
165             ']} .png"
166         file_path = os.path.join(png_folder, file_name)
167
168         if not settings.IS_LOCAL:
169             try:
170                 s3 = boto3.resource("s3")
171                 copy_source = {
172                     "Bucket": bucket_name,
173                     "Key": material["png_file_path"][1:],
174                 }

```

```

171         bucket = s3.Bucket(bucket_name)
172         s3_file_key = file_path[1:]
173         bucket.copy(copy_source, s3_file_key)
174     except Exception as e:
175         logging.info({"back_png_s3_copy_exception": e})
176     else:
177         shutil.copy2(material["png_file_path"], file_path)
178
179         material["png_file_path"] = file_path
180
181         logging.info("---_execution_time_to_copy_materials:_%s_seconds_
182         ---" % (time.time() - start_time))
183
184         if not settings.IS_LOCAL:
185             os.rmdir(png_folder)
186
187     # execution start time
188     start_time = time.time()
189
190     comb_map = {}
191     align_map = {}
192     for r in materials:
193         if r.png_height > r.png_width:
194             align_map[r.id] = False
195         else:
196             align_map[r.id] = True
197
198         if r.element_id in comb_map:
199             comb_map[r.element_id].append(r.id)
200         else:
201             comb_map[r.element_id] = [r.id]
202     logging.info(comb_map)
203
204     values = []
205     for i in sorted(comb_map):
206         values.append(comb_map[i][-10:])
207
208     logging.info(values)
209     material_combs = []
210     max_combo_num = label_generation_number // layout_pattern_number +
211     (
212         label_generation_number % layout_pattern_number > 0
213     )
214     for _ in range(max_combo_num):
215         random_title: GeneratedMaterial = None
216         random_copy: GeneratedMaterial = None
217         if job.text_title_string:
218             random_title = randomly_choose_text(
219                 generated_materials=generated_titles,
220                 registered_materials_from_db=
221                 registered_titles_from_db
222             )

```

```

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,
224             registered_materials_from_db=
225                 registered_copies_from_db
226         )
227         copy_chosen.append(random_copy)
228         materials_indexed += [random_copy.__dict__]
229
230     material_combs.append((random_product(*values, chosen_title=
231         random_title, chosen_copy=random_copy)))
232
233     logging.info({"comb_map": comb_map})
234     logging.info({"material_combs": material_combs})
235     logging.info({"len_material_combs": len(material_combs)})
236
237     # execution start time
238     start_time = time.time()
239
240     labels = []
241     for matid in material_combs[0]:
242         if job.job_type == 0:
243             mat = crud.materials.get(db=db, id=matid)
244             labels.append(mat.element.name)
245         elif job.job_type == 1:
246             mat = crud.registered_materials.get(db=db, id=matid)
247             if mat is None:
248                 continue
249             labels.append(mat.element.name)
250     if info_materials is not None:
251         labels += [x.element.name for x in info_materials]
252     logging.info({"labels": labels})
253
254     if ("info_barcode" in labels) and ("text_title" in labels):
255         tmp_mat_comb = []
256         barcode_idx = labels.index("info_barcode")
257         text_title_idx = labels.index("text_title")
258         for comb in material_combs:
259             text_long_width_flag = True
260             barcode_long_width_flag = True
261             # for matid in comb:
262             #     mat = crud.materials.get(db=db, id=matid)
263             #     if mat.label.name == "text_title":
264             #         if mat.height > mat.width:
265             #             text_long_width_flag = False
266             #     if mat.label.name == "info_barcode":
267             #         if mat.height > mat.width:
268             #             barcode_long_width_flag = False
269             barcode_matid = comb[barcode_idx]
270             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(
276         material_combs)})
277
278     tmp_aligned_comb = []
279     tmp_not_aligned_comb = []
280     labels_to_align = ["text_title", "text_copy", "text_explain"]
281     active_labels_to_align = [lab for lab in labels_to_align if lab in
282         labels]
283     active_labels_idx = [labels.index(lab) for lab in
284         active_labels_to_align]
285     if len(active_labels_to_align) > 1:
286         for comb in material_combs:
287             rule_align_flags = []
288             for actv_idx in active_labels_idx:
289                 matid = comb[actv_idx]
290                 rule_align_flags.append(align_map[matid])
291             # for matid in comb:
292             #     mat = crud.materials.get(db=db, id=matid)
293             #     if mat.label.name in active_labels_to_align:
294             #         if mat.height > mat.width:
295             #             rule_align_flags.append(False)
296             #         else:
297             #             rule_align_flags.append(True)
298             if (sum(rule_align_flags) == 0) or (sum(rule_align_flags)
299                 == len(active_labels_to_align)):
300                 tmp_aligned_comb.append(comb)
301             else:
302                 tmp_not_aligned_comb.append(comb)
303     else:
304         tmp_aligned_comb = material_combs
305         tmp_not_aligned_comb = []
306
307     # logging.info({"tmp_aligned_comb": tmp_aligned_comb})
308     # logging.info({"tmp_not_aligned_comb": tmp_not_aligned_comb})
309     logging.info({"len_tmp_aligned_comb": len(tmp_aligned_comb)})
310     logging.info({"len_tmp_not_aligned_comb": len(tmp_not_aligned_comb)
311         })
312
313     aligned_rate = 1
314     aligned_num = int(label_generation_number * aligned_rate)
315
316     if len(tmp_aligned_comb) > aligned_num:
317         tmp_aligned_comb = random.sample(tmp_aligned_comb, aligned_num)
318     tmp_not_aligned_comb = random.sample(tmp_not_aligned_comb, len(
319         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 аваад font-g ni job
325         # combosруу hadgalnaa
326         text_title_font, text_title_color = "", ""
327         copy_font, copy_color = "", ""
328
329         if job.text_title_string:
330             text_title_font, text_title_color = get_font_and_color(mc,
331                             title_chosen)
332
333         if job.text_copy_string:
334             copy_font, copy_color = get_font_and_color(mc, copy_chosen)
335
336         jc_in = schemas.JobCombinationsCreate(
337             job_id=job_id,
338             comb_name=comb,
339             text_title_font=text_title_font,
340             text_title_color=text_title_color,
341             text_copy_font=copy_font,
342             text_copy_color=copy_color,
343         )
344         crud.job_combinations.create(db=db, obj_in=jc_in)
345
346     job_combs = crud.job_combinations.get_multi_job_id(db=db, job_id=
347     job_id)
348
349     data = []
350     combinations = []
351
352     for jc, material_ids in zip(job_combs, material_combs):
353         combination_materials = []
354         for material_id in material_ids:
355             if job.job_type == 0:
356                 data.append(JobCombinationMaterials(job_comb_id=jc.id,
357                     material_uploaded_id=material_id))
358             elif job.job_type == 1:
359                 if material_id > 0:
360                     data.append(JobCombinationMaterialsSelected(
361                         job_comb_id=jc.id, registered_material_id=
362                         material_id))
363
364         # Return indexed materials

```

```
360         material = list(filter(lambda x: x["id"] == material_id
361                                , materials_indexed))[0]
362
363         combination_material_data = (
364             jc.id,
365             material["png_width"],
366             material["png_height"],
367             material["element"].name,
368             material["png_file_path"],
369             material["index"],
370         )
371         combination_materials.append(combination_material_data)
372
373         combinations.append((combination_materials, jc.id))
374
375         db.bulk_save_objects(data)
376         db.commit()
377
378         logging.info("---_execution_time_to_create_combs:_%s_seconds_---" %
379                     (time.time() - start_time))
380
381     return combinations if job.job_type == 1 else None
```

Код B.2: Combination үүсгэх бизнес логик код