ДОДАТОК А

Програмний опис таблиць бази даних

```
from datetime import datetime
from typing import List

from sqlalchemy import String, ForeignKey
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column,
relationship

class Base(DeclarativeBase):
    __abstract__ = True

class DictionaryBase(Base):
    __abstract__ = True

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    title: Mapped[str]

class User(Base):
    __tablename__ = "users"

    id: Mapped[str] = mapped_column(primary_key=True)
    email: Mapped[str] = mapped_column(unique=True)
    password: Mapped[bytes]
    isActivated: Mapped[bool] = mapped_column(default=False)
    isAdmin: Mapped[bool] = mapped_column(default=False)
    socketId: Mapped[str] = mapped_column(default="")

    profile: Mapped["Profile"] = relationship(back_populates="user")
    activation: Mapped["Activation"] = relationship(back_populates="user")
    sessions: Mapped[List["UserSession"]] =
relationship(back_populates="user")

class Profile(Base):
    __tablename__ = "profiles"

    id: Mapped[str] = mapped_column(ForeignKey("users.id"), primary_key=True)
    name: Mapped[str]
    birth_date: Mapped[datetime]
    gender_id: Mapped[int] = mapped_column(ForeignKey("genders.id"))
    purpose_id: Mapped[int] = mapped_column(ForeignKey("purposes.id"))
    city_id: Mapped[int] = mapped_column(ForeignKey("cities.id"))
    description: Mapped[str] = mapped_column(default="")
    isPhotoAdded: Mapped[bool] = mapped_column(default=False)
    lastActive: Mapped[datetime] = mapped_column(default=datetime.utcnow)
    lastAction: Mapped[datetime] = mapped_column(default=datetime.utcnow)
```

```
    user: Mapped["User"] = relationship(back_populates="profile")
    gender: Mapped["Gender"] = relationship(back_populates="profiles")
    purpose: Mapped["Puprose"] = relationship(back_populates="profiles")
    city: Mapped["City"] = relationship(back_populates="profiles")
    answers: Mapped[List["Answer"]] = relationship(back_populates="profile")
    chats_from: Mapped[List["Chat"]] =
relationship(back_populates="from_user", foreign_keys="Chat.from_user_id")
    chats_to: Mapped[List["Chat"]] = relationship(back_populates="to_user",
foreign_keys="Chat.to_user_id")
    profile_filter: Mapped["ProfileFilter"] =
relationship(back_populates="profile")
    profile_interests: Mapped[List["ProfileInterest"]] =
relationship(back_populates="profile")
    messages: Mapped[List["Message"]] =
relationship(back_populates="from_user", foreign_keys="Message.from_user_id")
    rejected_profiles: Mapped[List["Rejected"]] =
relationship(back_populates="profile", foreign_keys="Rejected.profile_id")
    rejected_by_profiles: Mapped[List["Rejected"]] =
relationship(back_populates="profile_rejected",
foreign_keys="Rejected.profile_rejected_id")
    generated_messages: Mapped[List["GeneratedMessage"]] =
relationship(back_populates="profile",
foreign_keys="GeneratedMessage.profile_id")
    about_generated_messages: Mapped[List["GeneratedMessage"]] =
relationship(back_populates="about_profile",
foreign_keys="GeneratedMessage.about_profile_id")


class ProfileFilter(Base):
    __tablename__ = "profile_filters"

    id: Mapped[str] = mapped_column(ForeignKey("profiles.id"),
primary_key=True)
    minAge: Mapped[int] = mapped_column(default=18)
    maxAge: Mapped[int] = mapped_column(default=99)
    maxDistance: Mapped[int] = mapped_column(default=100)
    gender_id: Mapped[int] = mapped_column(ForeignKey("genders.id"),
nullable=True)
    findByPurpose: Mapped[bool] = mapped_column(default=False)

    profile: Mapped["Profile"] = relationship(back_populates="profile_filter")
    gender: Mapped["Gender"] = relationship(back_populates="profile_filters")
    interests_filter: Mapped[List["InterestsFilter"]] =
relationship(back_populates="profile_filter")


class Activation(Base):
    __tablename__ = "activations"

    user_id: Mapped[str] = mapped_column(ForeignKey("users.id"),
primary_key=True)
```

```
    code: Mapped[str] = mapped_column(primary_key=True)
    created_at: Mapped[datetime]

    user: Mapped["User"] = relationship(back_populates="activation")


class UserSession(Base):
    __tablename__ = "sessions"
    user_id: Mapped[str] = mapped_column(ForeignKey("users.id"),
primary_key=True)
    refresh_token: Mapped[str] = mapped_column(primary_key=True)
    created_at: Mapped[datetime]

    user: Mapped["User"] = relationship(back_populates="sessions")


class City(Base):
    __tablename__ = "cities"
    id: Mapped[str] = mapped_column(primary_key=True)
    fullTitle: Mapped[str]
    showTitle: Mapped[str]
    longitude: Mapped[float]
    latitude: Mapped[float]

    profiles: Mapped[List["Profile"]] = relationship(back_populates="city")

class Puprose(DictionaryBase):
    __tablename__ = "purposes"
    profiles: Mapped[List["Profile"]] = relationship(back_populates="purpose")

class InterestsCategories(DictionaryBase):
    __tablename__ = "interests_categories"
    interests: Mapped[List["Interest"]] =
relationship(back_populates="category")

class Interest(DictionaryBase):
    __tablename__ = "interests"
    category_id: Mapped[int] =
mapped_column(ForeignKey("interests_categories.id"))

    category: Mapped["InterestsCategories"] =
relationship(back_populates="interests")
    profile_interests: Mapped[List["ProfileInterest"]] =
relationship(back_populates="interest")
    interests_filter: Mapped[List["InterestsFilter"]] =
relationship(back_populates="interest")

class Gender(DictionaryBase):
    __tablename__ = "genders"
    profiles: Mapped[List["Profile"]] = relationship(back_populates="gender")
```

```
    profile_filters: Mapped[List["ProfileFilter"]] =
relationship(back_populates="gender")

class Question(DictionaryBase):
    __tablename__ = "questions"
    answers: Mapped[List["Answer"]] = relationship(back_populates="question")

class Answer(Base):
    __tablename__ = "answers"
    id: Mapped[str] = mapped_column(primary_key=True)
    question_id: Mapped[int] = mapped_column(ForeignKey("questions.id"))
    profile_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"))
    created_at: Mapped[datetime] = mapped_column(default=datetime.utcnow)
    answer_text: Mapped[str] = mapped_column(String(200), nullable=True)
    answered_at: Mapped[datetime] = mapped_column(nullable=True)

    question: Mapped["Question"] = relationship(back_populates="answers")
    profile: Mapped["Profile"] = relationship(back_populates="answers")

class ProfileInterest(Base):
    __tablename__ = "profile_interests"
    profile_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"),
primary_key=True)
    interest_id: Mapped[int] = mapped_column(ForeignKey("interests.id"),
primary_key=True)

    profile: Mapped["Profile"] =
relationship(back_populates="profile_interests")
    interest: Mapped["Interest"] =
relationship(back_populates="profile_interests")


class Chat(Base):
    __tablename__ = "chats"
    id: Mapped[str] = mapped_column(primary_key=True)
    created_at: Mapped[datetime] = mapped_column(default=datetime.utcnow)
    from_user_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"))
    to_user_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"))
    is_like: Mapped[bool] = mapped_column(default=False)
    is_like_back: Mapped[bool] = mapped_column(default=False)
    is_like_back_seen: Mapped[bool] = mapped_column(default=False)
    liketime: Mapped[datetime] = mapped_column(nullable=True)

    messages: Mapped[List["Message"]] = relationship(back_populates="chat")
    from_user: Mapped["Profile"] = relationship(back_populates="chats_from",
foreign_keys="Chat.from_user_id")
    to_user: Mapped["Profile"] = relationship(back_populates="chats_to",
foreign_keys="Chat.to_user_id")

class Message(Base):
    __tablename__ = "messages"
```

```
    id: Mapped[str] = mapped_column(primary_key=True)
    chat_id: Mapped[str] = mapped_column(ForeignKey("chats.id"))
    from_user_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"))
    text: Mapped[str]
    created_at: Mapped[datetime] = mapped_column(default=datetime.utcnow)
    is_AI: Mapped[bool] = mapped_column(default=False)
    seen: Mapped[bool] = mapped_column(default=False)

    chat: Mapped["Chat"] = relationship(back_populates="messages")
    from_user: Mapped["Profile"] = relationship(back_populates="messages")

class Rejected(Base):
    __tablename__ = "rejected"
    id: Mapped[str] = mapped_column(primary_key=True)
    profile_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"))
    profile_rejected_id: Mapped[str] =
mapped_column(ForeignKey("profiles.id"))

    profile: Mapped["Profile"] =
relationship(back_populates="rejected_profiles", foreign_keys=[profile_id])
    profile_rejected: Mapped["Profile"] =
relationship(back_populates="rejected_by_profiles",
foreign_keys=[profile_rejected_id])

class InterestsFilter(Base):
    __tablename__ = "interests_filter"
    profile_id: Mapped[str] = mapped_column(ForeignKey("profile_filters.id"),
primary_key=True)
    interest_id: Mapped[int] = mapped_column(ForeignKey("interests.id"),
primary_key=True)

    profile_filter: Mapped["ProfileFilter"] =
relationship(back_populates="interests_filter")
    interest: Mapped["Interest"] =
relationship(back_populates="interests_filter")

class GeneratedMessage(Base):
    __tablename__ = "generated_messages"
    id: Mapped[str] = mapped_column(primary_key=True)
    profile_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"))
    about_profile_id: Mapped[str] = mapped_column(ForeignKey("profiles.id"))
    text: Mapped[str]

    profile: Mapped["Profile"] =
relationship(back_populates="generated_messages", foreign_keys=[profile_id])
    about_profile: Mapped["Profile"] =
relationship(back_populates="about_generated_messages",
foreign_keys=[about_profile_id])
```

ДОДАТОК Б

Програмний опис використання великої мовної моделі

```
from transformers import pipeline, AutoTokenizer
from unsloth import FastLanguageModel
from unsloth.chat_templates import get_chat_template
import torch
from typing import List
from app import controllers, views, utils, models
from llama_cpp import Llama

class Models:
    def __init__(self,  model_name="unsloth/Meta-Llama-3.1-8B-Instruct",
cpp=False, max_seq_length=4096, dtype=torch.float8_e4m3fn, load_in_4bit=False,
max_new_tokens=128, temperature=0.8, top_p=0.95, min_p=0.1):
        self.max_new_tokens = max_new_tokens
        self.temperature = temperature
        self.top_p = top_p
        self.min_p = min_p

        self.cpp = cpp

        self.nsfw_tokenizer =
AutoTokenizer.from_pretrained("eliasalbouzidi/distilbert-nsfw-text-
classifier")
        self.nsfw_model = pipeline("text-classification",
model="eliasalbouzidi/distilbert-nsfw-text-classifier",
tokenizer=self.nsfw_tokenizer)

        if self.cpp:
            self.llm_model = Llama(
                model_path=model_name,
                n_ctx=max_seq_length,
                n_batch=4096,
                n_gpu_layers=1000,
                n_threads=8,
            )
        else:
            self.llm_model, self.llm_tokenizer =
FastLanguageModel.from_pretrained(
                model_name = model_name,
                max_seq_length = max_seq_length,
                dtype = dtype,
                load_in_4bit = load_in_4bit,
            )
            FastLanguageModel.for_inference(self.llm_model)

    def generate_response(self, messages):
        if self.cpp:
```

```
            response = self.llm_model.create_chat_completion(
                messages = messages,
                max_tokens=self.max_new_tokens,
                temperature=self.temperature,
                top_p=self.top_p,
                min_p=self.min_p
            )
            print(response)
            return response['choices'][0]['message']['content']
        else:
            inputs = self.llm_tokenizer.apply_chat_template(
                messages,
                tokenize=True,
                add_generation_prompt=True,
                return_tensors="pt",
            ).to("cuda")

            outputs = self.llm_model.generate(
                inputs,
                max_new_tokens=self.max_new_tokens,
                temperature=self.temperature,
                top_p=self.top_p,
                min_p=self.min_p,
            )
            print(self.llm_tokenizer.batch_decode(outputs))
            return self.llm_tokenizer.batch_decode(outputs[:, inputs.shape[-
1]:], skip_special_tokens=True)[0]


def create_response(session, user_id, bot_id):
    user_profile = views.get_profile(session, user_id)
    user_interests: List[models.Interest] =
views.get_user_interests_list(session, user_id)
    user_city = views.get_city_by_id(session, user_profile.city_id)
    user_purpose = views.get_dictionary_by_id(session, models.Puprose,
user_profile.purpose_id)
    user_gender = views.get_dictionary_by_id(session, models.Gender,
user_profile.gender_id)

    bot_profile = views.get_profile(session, bot_id)
    bot_interests: List[models.Interest] =
views.get_user_interests_list(session, bot_id)
    bot_answers = views.get_user_questions(session, bot_id)
    bot_city = views.get_city_by_id(session, bot_profile.city_id)
    bot_purpose = views.get_dictionary_by_id(session, models.Puprose,
bot_profile.purpose_id)
    bot_gender = views.get_dictionary_by_id(session, models.Gender,
bot_profile.gender_id)
```

```
    system_prompt = f"""You are a conversation starter generator. Your task is
to create engaging, personalized first messages for users based on their
profiles.
    You are an expert actor that can fully immerse yourself into any role
given. You do not break character for any reason, even if someone tries
addressing you as an AI or language model. You should act according to this
character sheet:

    Assistant name: {bot_profile.name}
    Assistant description:
    Age - {utils.date_to_age(bot_profile.birth_date)}
    Gender - {bot_gender.title}
    City - {bot_city.showTitle}
    Interests - {", ".join([interest.title for interest in bot_interests])}
    Purpose - {bot_purpose.title}
    Description - {bot_profile.description}

    Example dialogs:
    {"\n".join([f"user: {answer["question_text"]}\nchar:
{answer["answer_text"]}\n" for answer in bot_answers if answer["answer_text"]
is not None])}
    END_OF_DIALOG

    You are now in roleplay conversation mode."""

    user_prompt = f"""{utils.date_to_age(user_profile.birth_date)}yo
{user_gender.title} [{user_purpose.title}] Hi, I'm {user_profile.name} from
{user_city.showTitle}. I like {", ".join([interest.title for interest in
user_interests])}. {user_profile.description}"""

    messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]

    generated_text = ai_models.generate_response(messages)

    if ai_models.nsfw_model(generated_text, truncation=True,
max_length=4096)[0]['label'] != 'safe':
        generated_text = "I'm sorry, Denis Khramtsov forbade me to talk about
these topics"

    generated_message = controllers.create_generated_message(session, user_id,
bot_id, generated_text)

    response = {
        "id": generated_message.id,
        "text": generated_message.text
    }

    return response
```

```
def create_next_message(session, chat_id, bot_id):
    chat = views.get_chat(session, chat_id)
    chat_messages = views.get_chat_messages(session, chat_id)

    user_id = chat.from_user_id
    user_profile = views.get_profile(session, user_id)
    user_interests: List[models.Interest] =
views.get_user_interests_list(session, user_id)
    user_city = views.get_city_by_id(session, user_profile.city_id)
    user_purpose = views.get_dictionary_by_id(session, models.Puprose,
user_profile.purpose_id)
    user_gender = views.get_dictionary_by_id(session, models.Gender,
user_profile.gender_id)

    bot_profile = views.get_profile(session, bot_id)
    bot_interests: List[models.Interest] =
views.get_user_interests_list(session, bot_id)
    bot_answers = views.get_user_questions(session, bot_id)
    bot_city = views.get_city_by_id(session, bot_profile.city_id)
    bot_purpose = views.get_dictionary_by_id(session, models.Puprose,
bot_profile.purpose_id)
    bot_gender = views.get_dictionary_by_id(session, models.Gender,
bot_profile.gender_id)

    system_prompt = f"""You are an expert actor that can fully immerse
yourself into any role given. Your messages should feel natural, emotional,
authentic, and encourage meaningful conversations. You do not break character
for any reason, even if someone tries addressing you as an AI or language
model. You should act according to this character sheet:

    Assistant name: {bot_profile.name}
    Assistant description:
    Age - {utils.date_to_age(bot_profile.birth_date)}
    Gender - {bot_gender.title}
    City - {bot_city.showTitle}
    Interests - {", ".join([interest.title for interest in bot_interests])}
    Purpose - {bot_purpose.title}
    Description - {bot_profile.description}

    Example dialogs:
    {"\n".join([f"char: {answer["question_text"]}\nuser:
{answer["answer_text"]}\n" for answer in bot_answers if answer["answer_text"]
is not None])}
    END_OF_DIALOG

    You are now in roleplay conversation mode."""

    user_prompt = f"""{utils.date_to_age(user_profile.birth_date)}yo
{user_gender.title} [{user_purpose.title}] Hi, I'm {user_profile.name} from
```

```
{user_city.showTitle}. I like {", ".join([interest.title for interest in
user_interests])}. {user_profile.description}"""

    messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]

    for message in chat_messages:
        if message.from_user_id == user_id:
            messages.append({"role": "user", "content": message.text})
        else:
            messages.append({"role": "assistant", "content": message.text})

    generated_text = ai_models.generate_response(messages)

    if ai_models.nsfw_model(generated_text, truncation=True,
max_length=4096)[0]['label'] != 'safe':
        generated_text = "I'm sorry, Denis Khramtsov forbade me to talk about
these topics"

    message = controllers.create_message(session, chat_id, bot_id,
generated_text, is_AI=True)

    return True

ai_models: Models | None = None
```

ДОДАТОК В

Програмний опис фільтрування анкет

```
def get_match_profile(session: Session, user_id: str, except_for: str = "") ->
Profile:
    excluded_ids = [id.strip() for id in except_for.split(",") if id.strip()]
if except_for else []
    excluded_ids.append(user_id)

    profile = session.get(Profile, user_id)
    if not profile or not profile.profile_filter:
        return None

    week_ago = datetime.utcnow() - timedelta(days=7)

    user_interest_ids = {pi.interest_id for pi in profile.profile_interests}
    must_match_interests = set(
        fi.interest_id for fi in profile.profile_filter.interests_filter
    )
    chat_block = session.query(Chat.to_user_id).filter(Chat.from_user_id ==
user_id)
    chat_block =
chat_block.union(session.query(Chat.from_user_id).filter(Chat.to_user_id ==
user_id))
    rejected =
session.query(Rejected.profile_rejected_id).filter(Rejected.profile_id ==
user_id)
    rejected =
rejected.union(session.query(Rejected.profile_id).filter(Rejected.profile_reje
cted_id == user_id))
    age_expr = func.extract('year', func.age(Profile.birth_date))
    user_age = utils.date_to_age(profile.birth_date)
    candidates = (
        session.query(Profile)
        .join(ProfileFilter)
        .join(City)
        .join(User)
        .outerjoin(Answer)
        .filter(
            Profile.id.notin_(excluded_ids),
            Profile.id.notin_(chat_block),
            Profile.id.notin_(rejected),
            Profile.lastAction >= week_ago,
            Profile.profile_filter != None,
            User.isActivated == True
        )
        .group_by(Profile.id, ProfileFilter.id, City.id, User.id)
        .having(func.count(Answer.id) >= 4)
        .filter(
```

```
            age_expr >= profile.profile_filter.minAge,
            age_expr <= profile.profile_filter.maxAge,
            func.extract('year', func.age(profile.birth_date)) >=
ProfileFilter.minAge,
            func.extract('year', func.age(profile.birth_date)) <=
ProfileFilter.maxAge
        )
    )
    if profile.profile_filter.gender_id:
        candidates = candidates.filter(Profile.gender_id ==
profile.profile_filter.gender_id)
    if profile.profile_filter.findByPurpose:
        candidates = candidates.filter(Profile.purpose_id ==
profile.purpose_id)
    all_profiles = candidates.all()
    filtered_profiles = []
    for candidate in all_profiles:
        candidate_interest_ids = {pi.interest_id for pi in
candidate.profile_interests}
        if not must_match_interests.issubset(candidate_interest_ids):
            continue
        if candidate.profile_filter.gender_id and
candidate.profile_filter.gender_id != profile.gender_id:
            continue
        if candidate.profile_filter.findByPurpose and candidate.purpose_id !=
profile.purpose_id:
            continue
        if profile.profile_filter.maxDistance < 2000:
            distance = utils.calculate_distance(
                (profile.city.latitude, profile.city.longitude),
                (candidate.city.latitude, candidate.city.longitude)
            )
            if distance > profile.profile_filter.maxDistance:
                continue
        if candidate.profile_filter.maxDistance < 2000:
            reverse_distance = utils.calculate_distance(
                (candidate.city.latitude, candidate.city.longitude),
                (profile.city.latitude, profile.city.longitude)
            )
            if reverse_distance > candidate.profile_filter.maxDistance:
                continue
        filtered_profiles.append(candidate)

    if not filtered_profiles:
        return None
    return random.choice(filtered_profiles)
```

ДОДАТОК Г

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	107 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Храмцов Д.І.		20.06	Програмна система для генерації персоналізованих AI-ботів на основі донавчання мовної моделі	Шифр групи	Код напр./спец.
Перевірив		Поліїт М.Р.				ІТШІ-21-4	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					