Documentation for `routes/auth.py`

The module `routes/auth.py` contains the endpoints for the authentication service. It provides routes for user registration, login, logout, and other related operations.

Endpoints

endpoint: `/users/{user_id}`

Delete a user by their user ID.

- **user_id**: int - The ID of the user to delete.
- **current_user**: User - The current authenticated user attempting the deletion.
- **db**: Session - The database session dependency.
- **return**: Returns a dictionary with a detail message on successful deletion.

Source code in `routes/auth.py`:

```
@router.delete("/users/{user_id}", tags=["users"], response_model=dict)
def delete_user(
    user_id: int, current_user=Depends(login_manager), db: Session = Depends(get_db)
):
    if not current_user:
        raise InvalidCredentialsException(detail="Invalid credentials")
    if current_user.id != 1:
        raise HTTPException(status_code=401, detail="Unauthorized")
    user = db.query(User).filter(User.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    db.delete(user)
    db.commit()
    return {"detail": "User deleted"}
```

endpoint: `/gui

Render a front-end GUI for testing signup/login functionality.

- **return**: Returns an HTMLResponse containing the content of the `index.html` page.

Source code in `routes/auth.py`:

```
@router.get("/gui", tags=["auth gui"], response_class=HTMLResponse)
```

```
def index():
    file_path = os.path.join("gui", "templates", "index.html")
    with open(file_path, "r") as f:
        return HTMLResponse(content=f.read())
```

endpoint: `/users`

List all users.

- **user**: User - The current authenticated user (unused in this function).
- **db**: Session - The database session dependency.
- **return**: Returns a list of dictionaries, each representing a user with their id and email.

Source code in `routes/auth.py` :

```
@router.get("/users", tags=["users"], response_model=list)
def list_users(user=Depends(login_manager), db: Session = Depends(get_db)):
    users = db.query(User).all()
    users = [{"id": user.id, "email": user.email} for user in users]
    return users
```

endpoint: `/token`

Authenticate a user and return an access token.

- **data**: OAuth2PasswordRequestForm - A form data model including username (email) and password.
- **return**: Returns a TokenData object containing the access token and token type.

Source code in `routes/auth.py` :

```
@router.post("/token", tags=["users"], response_model=TokenData)
def login(data: OAuth2PasswordRequestForm = Depends()):
    email = data.username
    password = data.password
    user = get_user(email)
    if not user or not bcrypt.checkpw(
        password.encode("utf-8"), user.hashed_password.encode("utf-8")
    ):
        raise InvalidCredentialsException
    access_token_expires = timedelta(minutes=DEFAULT_SETTINGS.access_token_expire_mi
    access_token = login_manager.create_access_token(
        data=dict(sub=email), expires=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

endpoint: `/private`

A private route that requires authentication.

- **user**: User - The current authenticated user.
- **return**: Returns a dictionary with a welcome message for the authenticated user.

Source code in `routes/auth.py` :

```
@router.get("/private", tags=["users"], summary="A private route that requires authe
def private_route(user=Depends(login_manager)):
    return {"detail": f"Welcome {user.email}, you are authenticated"}
```

endpoint: `/users/me`

Get the current authenticated user.

- **user**: User - The current authenticated user from the session.
- **return**: Returns the user object of the currently authenticated user.

Source code in `routes/auth.py` :

```
@router.get("/users/me", tags=["users"])
async def read_users_me(user: User = Depends(login_manager)):
    return user
```

endpoint: `/register`

Register a new user with the provided email and password.

- **user**: UserCreate - A user creation object containing the email and password.
- **db**: Session - The database session dependency.
- **return**: Returns a dictionary with a detail message on successful registration.

Source code in `routes/auth.py` :

```
@router.post("/register", tags=["users"], response_model=dict)
def register(user: UserCreate, db: Session = Depends(get_db)):
    db_user = get_user(user.email)
    if db_user:
        raise HTTPException(
            status_code=400, detail="A user with this email already exists"
        )
    hashed_password = hash_password(user.password)
    db_user = User(email=user.email, hashed_password=hashed_password)
    db.add(db_user)
    db.commit()
    return {"detail": "Successfully registered"}
```

For more details, visit the documentation.