

Documentation for `routes/auth.py`

This module contains the endpoints for the authentication service. It provides routes for user registration, login, and logout and other related operations.

```
delete_user(user_id,
current_user=Depends(login_manager),
db=Depends(get_db))
```

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`

```

87 @router.delete("/users/{user_id}", tags=["users"], response_model=dict)
88 def delete_user(
89     user_id: int, current_user=Depends(login_manager), db: Session =
90     Depends(get_db)
91 ):
92     """
93     Delete a user by their user ID.
94
95     - **user_id**: int - The ID of the user to delete.
96     - **current_user**: User - The current authenticated user attempting
97     the deletion.
98     - **db**: Session - The database session dependency.
99     - **return**: Returns a dictionary with a detail message on successful
100     deletion.
101     """
102     if not current_user:
103         raise InvalidCredentialsException(detail="Invalid credentials")
104     if current_user.id != 1:
105         raise HTTPException(status_code=401, detail="Unauthorized")
106     user = db.query(User).filter(User.id == user_id).first()
107     if not user:
108         raise HTTPException(status_code=404, detail="User not found")
109     db.delete(user)
110     db.commit()
111     return {"detail": "User deleted"}
```

index()

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

```

53 @router.get("/gui", tags=["auth gui"], response_class=HTMLResponse)
54 def index():
55     """
56     Render a front-end GUI for testing signup/login functionality.
57
58     - **return**: Returns an HTMLResponse containing the content of the
59     index.html page.
60     """
61     file_path = os.path.join("gui", "templates", "index.html")
62     with open(file_path, "r") as f:
63         return HTMLResponse(content=f.read())

```

list_users(user=Depends(login_manager), db=Depends(get_db))

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

```

122 @router.get("/users", tags=["users"], response_model=list)
123 def list_users(user=Depends(login_manager), db: Session =
124 Depends(get_db)):
125     """
126     List all users.
127
128     - **user**: User - The current authenticated user (unused in this
129     function).
130     - **db**: Session - The database session dependency.
131     - **return**: Returns a list of dictionaries, each representing a user
132     with their id and email.
133     """
134     users = db.query(User).all()
135     users = [{"id": user.id, "email": user.email} for user in users]
136     return users

```

login(data=Depends())

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

```

65 @router.post("/token", tags=["users"], response_model=TokenData)
66 def login(data: OAuth2PasswordRequestForm = Depends()):
67     """
68     Authenticate a user and return an access token.
69
70     - **data**: OAuth2PasswordRequestForm - A form data model including
71     username (email) and password.
72     - **return**: Returns a TokenData object containing the access token
73     and token type.
74     """
75     email = data.username
76     password = data.password
77     user = get_user(email)
78     if not user or not bcrypt.checkpw(
79         password.encode("utf-8"), user.hashed_password.encode("utf-8")
80     ):
81         raise InvalidCredentialsException
82     access_token_expires =
83     timedelta(minutes=DEFAULT_SETTINGS.access_token_expire_minutes)
84     access_token = login_manager.create_access_token(
85         data=dict(sub=email), expires=access_token_expires
86     )
87     return {"access_token": access_token, "token_type": "bearer"}

```

private_route(user=Depends(login_manager))

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`

```

111 @router.get("/private", tags=["users"], summary="A private route that
112 requires authentication.", response_model=dict)
113 def private_route(user=Depends(login_manager)):
114     """
115     A private route that requires authentication.
116
117     - **user**: User - The current authenticated user.
118     - **return**: Returns a dictionary with a welcome message for the
119     authenticated user.
120     """
121     return {"detail": f"Welcome {user.email}, you are authenticated"}

```

`read_users_me(user=Depends(login_manager))` `async`

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`

```

21 @router.get("/users/me", tags=["users"])
22 async def read_users_me(user: User = Depends(login_manager)):
23     """
24     Get the current authenticated user.
25
26     - **user**: User - The current authenticated user from the session.
27     - **return**: Returns the user object of the currently authenticated
28     user.
29     """
30     return user

```

`register(user, db=Depends(get_db))`

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

```
32 @router.post("/register", tags=["users"], response_model=dict)
33 def register(user: UserCreate, db: Session = Depends(get_db)):
34     """
35     Register a new user with the provided email and password.
36
37     - **user**: UserCreate - A user creation object containing the email
38     and password.
39     - **db**: Session - The database session dependency.
40     - **return**: Returns a dictionary with a detail message on successful
41     registration.
42     """
43     db_user = get_user(user.email)
44     if db_user:
45         raise HTTPException(
46             status_code=400, detail="A user with this email already exists"
47         )
48     hashed_password = hash_password(user.password)
49     db_user = User(email=user.email, hashed_password=hashed_password)
50     db.add(db_user)
    db.commit()
    return {"detail": "Successfully registered"}
```