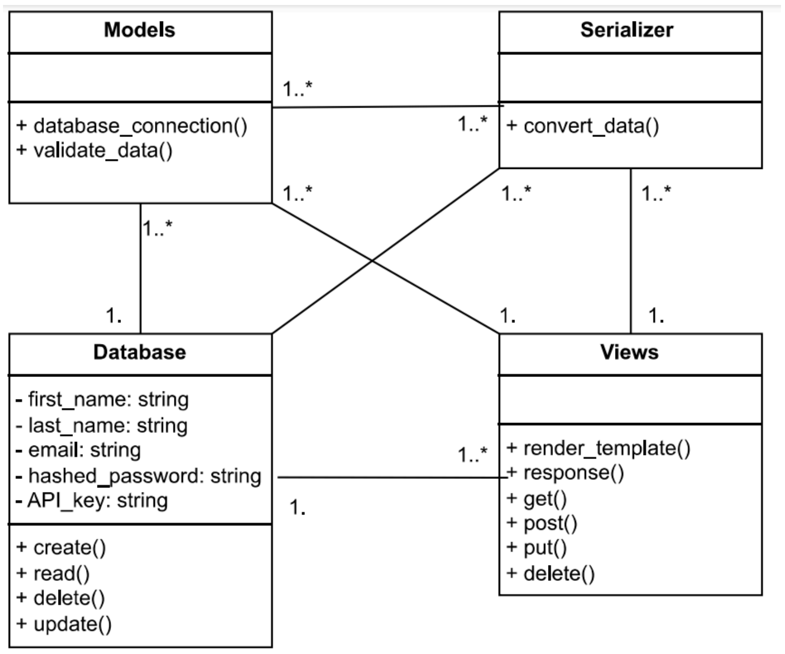
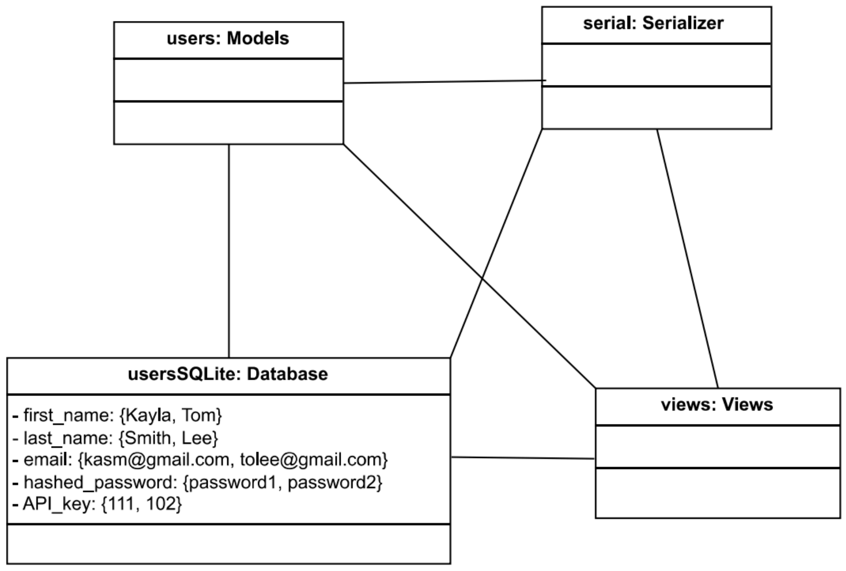
Class and Object Diagrams

ID: 4.1
Name: Models Class and Object Diagram

Class Diagram:



Object Diagram:



users: Models
serial: Serializer
usersSQLite: Database
- first_name: {Kayla, Tom}
- last_name: {Smith, Lee}
- email: {kasm@gmail.com, tolee@gmail.com}
- hashed_password: {password1, password2}
- API_key: {111, 102}
views: Views

ID: 4.2
Name: Overall Project Class and Object Diagram

Class Descriptions:
Models: The model class is responsible for the CRUD actions going to our database(s). Calls like creating a user, updating encrypted api information, reading data for user, and deleting user information will be through the Models class.

Movies: The Movies class is a backend class to be used for processing avatar content videos. The Movies class will utilize a Python library, movie.py, to record slideshows and merge avatar and slideshow videos into one streamlined downloadable video.

Views: The Views class is a generated class from Django. The Views class is responsible for handling API requests to/from our different resources, such as HeyGen, GoogleSlides, and ChatGPT OpenAI. The Vies class takes requests from the server (sent by the class Dashboard), handles the request appropriately, and sends back json information to the server.

Dashboard: The Dashboard class acts as the controller for this application (in the MVC design sense). The Dashboard is responsible for transferring requests to/from the frontend and backend. The Dashboard serves as the middle ground communication for both the client side and server side of this application.

Avatars: The Avatars class serves as the frontend representative for handling the avatars produced by the HeyGen API. Avatars are lifelike AI-generated speaking figures that are hyper realistic, and created directly by HeyGen (not by our application) via get/post requests to the Dashboard class.

Voices: The Voices class represents voices available for avatars located in HeyGen via get/post requests to the Dashboard class. The class is meant to be utilized for selecting a voice for an avatar.

Scripts: The Scripts class defines our AI-Generated scripts, and is responsible for sending and receiving requests to generate scripts based on a user input.

Video_Projects: The Video_Porjects class is responsible for hosting the requests for an avatar video to be made, and any edits requested by our user or system as well.

Interfaces: The interfaces class represents our interface throughout our program. The interface/GUI changes based on routing and logic between the Interface class, the client, and the Dashboard class.

Pre and Post Conditions for Methods:

Models:

Method: create(json{}): json{}
Pre-Condition: A valid email, firstname, and last name must exist in the json request.
Post-Condition: The database should create a new entry using the provided information in the json request.

Method: update(json{}): string
Pre-Condition: A user ID must exist and be present in the json request.
Post-Condition: The database should be updated with the correct requested information.

Method: read(json{}): json{}
Pre-Condition: A user ID must exist and be present in the json request.
Post-Condition: The resulting json should not be null.

Method: delete(json{}): string
Pre-Condition: A user ID must exist and be present in the json request.
Post-Condition: The database should not contain the data present in the request json.

Movies:

Method: record_slideshow(json{}): json{}
Pre-Condition: A slideshow must exist in the request json.
Post-Condition: The slideshow will be saved in a video file format.

Method: reframe_videos(json{}): json{}
Pre-Condition: Exactly two videos must be present in the request json{}
Post-Condition: A single video file exists in the return json containing both videos merged into one successfully.

Views:

Method: generate_script(json{}): json{}
Pre-Condition: The content generation prompt field must be filled out with text from the user.
Post-Condition: The output script should be in the language that the content prompt was in.

Method: parse_audio(json{}): json{}
Pre-Condition: An avatar video must be generated to grab the audio from that file.
Post-Condition: The audio should be correctly parsed.

Method: get_voices(json{}): json{}
Pre-Condition: A HeyGen API key must be present in a user's profile.
Post-Condition: The available voices should include Heygen's standard library of free voices to use in the returned json.

Method: get_avatars(json{}): json{}
Pre-Condition: A HeyGen API key must be present in a user's profile.
Post-Condition: The available avatars should include Heygen's standard library of free avatars to use in the returned json.

Method: get_video_link(json{}): json{}

Pre-Condition: A video must have been generated, and have a return code of 'success'.
Post-Condition: The link must be valid.

Method: generate_video(json{}): json{}
Pre-Condition: An avatar must be selected.
Pre-Condition: A voice must be selected
Pre-Condition: A script must be generated.
Post-Condition: A video lasting more than 30 seconds is generated.

Method: create_slides(json{}): json{}
Pre-Condition: A script must already be generated.
Post-Condition: A slideshow must be created and formatted into a json response correctly.

Method: save_slides(json{}): json{}
Pre-Condition: Slides must already be generated.
Post-Condition: The slides will be saved to a user's profile via their ID.

Method: validate_data(json{}): json{}
Pre-Condition: Data must exist in the json request.
Post-Condition: Data will be formatted and validated.

Dashboard:

Method: route_request(json{}): json{}
Pre-Condition: There must be a valid GET/POST request present.
Post-Condition: The requested route will be executed on the server side.

Method: recieve_request(json{}): json{}
Pre-Condition: There must be a valid GET/POST request present.
Post-Condition: The request is parsed correctly.

Method: determine_display(string): string
Pre-Condition: The user is on the ProfAI website.
Post-Condition: The user is routed to the correct display.

Avatars:

Method: avatars(json{}): json{}
Pre-Condition: A user has entered the 'My Projects' pane on the ProfAI website.
Post-Condition: The request will have routed through Dashboard, gathered the available avatars, and displayed them.

Voices:

Method: voices(json{}): json{}
Pre-Condition: A user has entered the 'My Voices' pane on the ProfAI website.
Post-Condition: The request will have been routed through Dashboard, gathered the available voices, and displayed them.

Scripts:

Method: script(json{}): json{}
Pre-Condition: A content prompt must exist in the content prompt filed on the ProfAI website, under the 'My Projects' pane.

Post-Condition: The request will have routed through Dashboard, generated the script, and displayed it.

Method: edit_script(json{}): json{}
Pre-Condition: A generated script has been displayed to a user.
Post-Condition: A user's edits to a script are displayed.

Video_Projects:

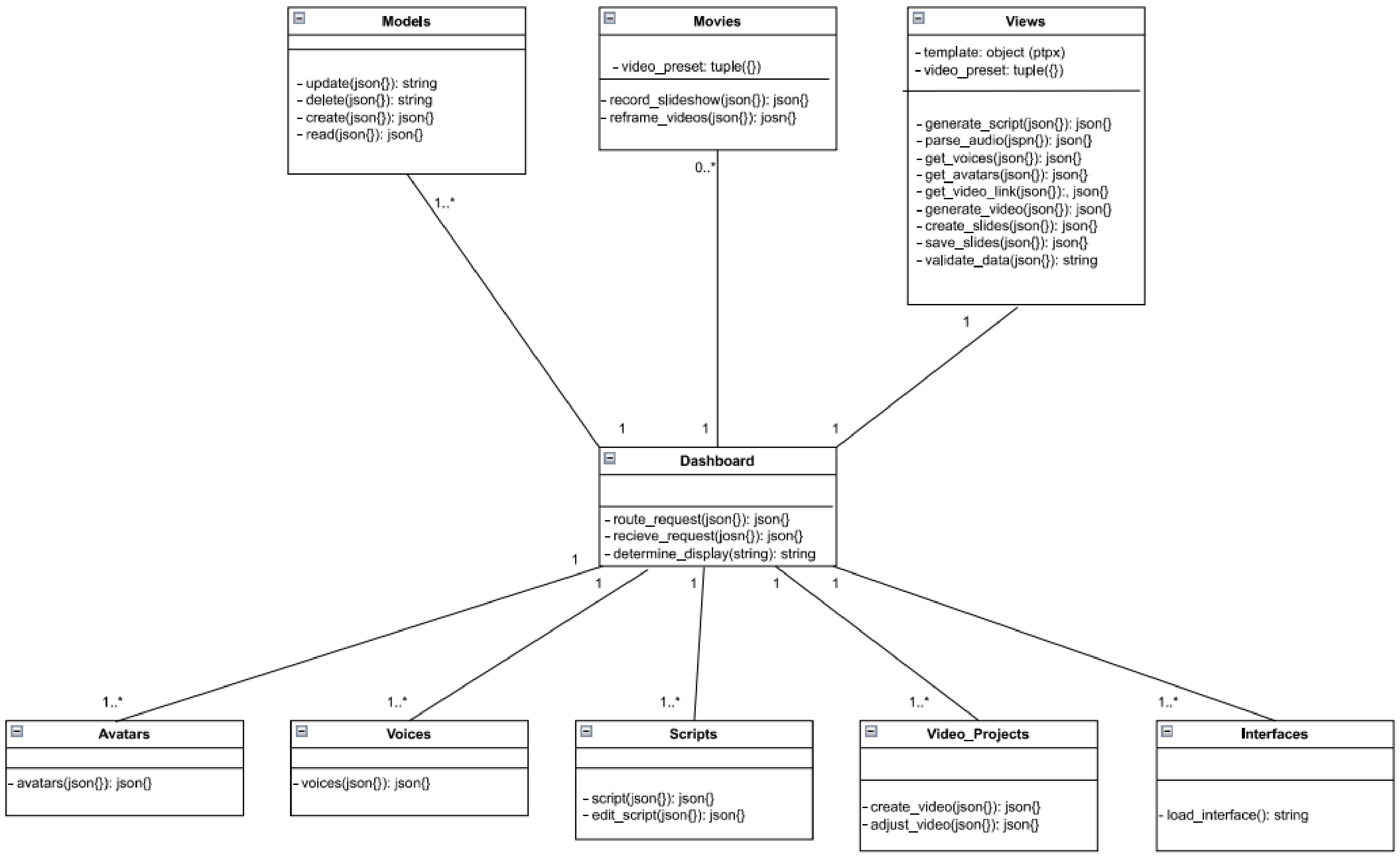
Method: create_video(json{}): json{}
Pre-Condition: An avatar, voice, and script have been created/generated on the 'My Projects' pane.
Post-Condition: A video has been successfully generated.

Method: adjust_video(json{}): json{}
Pre-Condition: A video has been generated.
Post-Condition: An updated video displaying the changes made through the Dashboard/Server Side are reflected in the video.

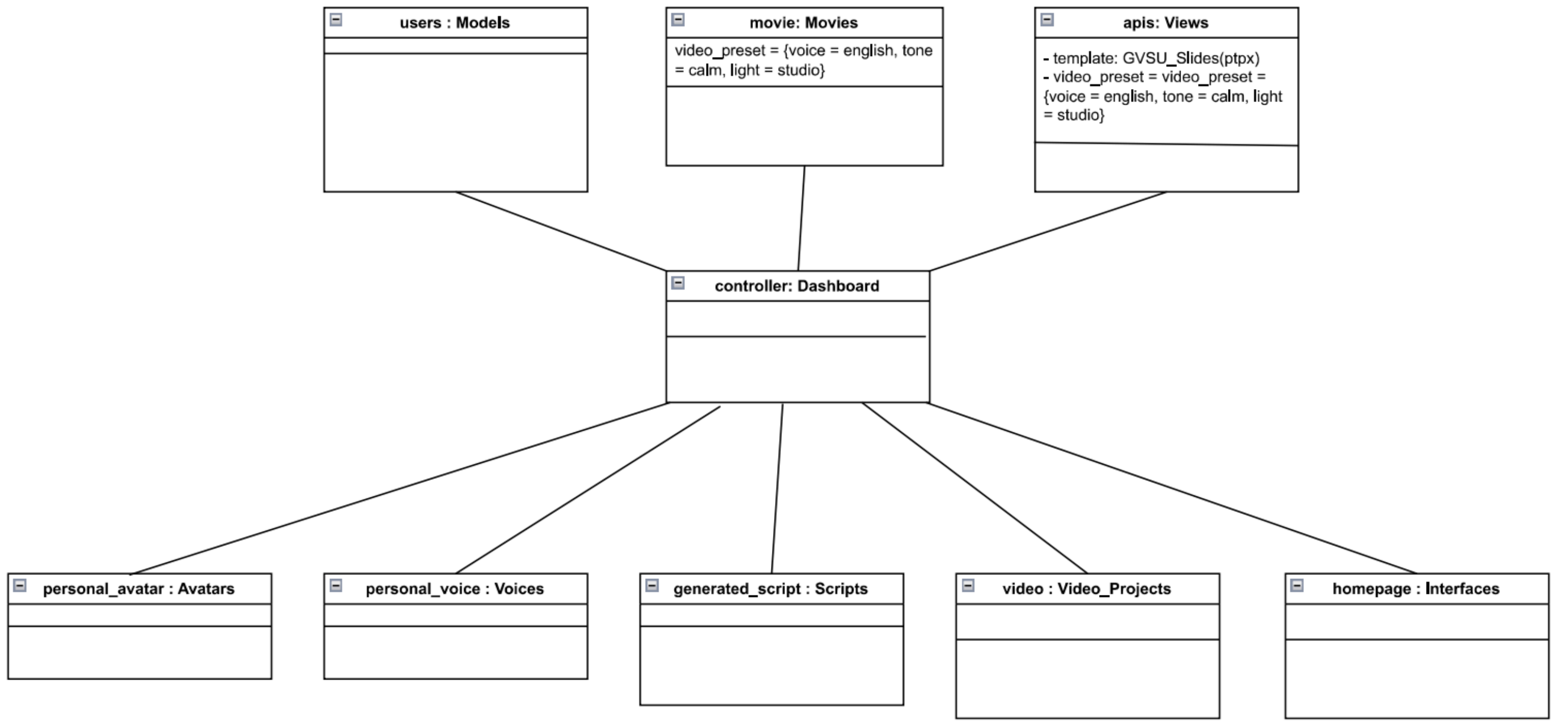
Interfaces:

Method: load_interface( ): string
Pre-Condition: A user is on the ProfAI website.
Post-Condition: A success message is displayed on the server side terminal when switching between interfaces.

Class Diagram:


Models
- update(json{}): string
- delete(json{}): string
- create(json{}): json{}
- read(json{}): json{}
Movies
- video_preset: tuple({})
- record_slideshow(json{}): json{}
- reframe_videos(json{}): josn{}
Views
- template: object (ptpx)
- video_preset: tuple({})
- generate_script(json{}): json{}
- parse_audio(jspn{}): json{}
- get_voices(json{}): json{}
- get_avatars(json{}): json{}
- get_video_link(json{}):, json{}
- generate_video(json{}): json{}
- create_slides(json{}): json{}
- save_slides(json{}): json{}
- validate_data(json{}): string
1..*
0..*
1
1
1
1
Dashboard
- route_request(json{}): json{}
- recieve_request(josn{}): json{}
- determine_display(string): string
1
1
1
1
1
1..*
1..*
1..*
1..*
1..*
Avatars
- avatars(json{}): json{}
Voices
- voices(json{}): json{}
Scripts
- script(json{}): json{}
- edit_script(json{}): json{}
Video_Projects
- create_video(json{}): json{}
- adjust_video(json{}): json{}
Interfaces
- load_interface(): string

Object Diagram:


users : Models
movie: Movies
video_preset = {voice = english, tone = calm, light = studio}
apis: Views
- template: GVSU_Slides(ptpx)
- video_preset = video_preset = {voice = english, tone = calm, light = studio}
controller: Dashboard
personal_avatar : Avatars
personal_voice : Voices
generated_script : Scripts
video : Video_Projects
homepage : Interfaces