ToneBase 1.0 Technical Requirements

This document specifies the scope and requirements of the “ToneBase 1.0” information system.

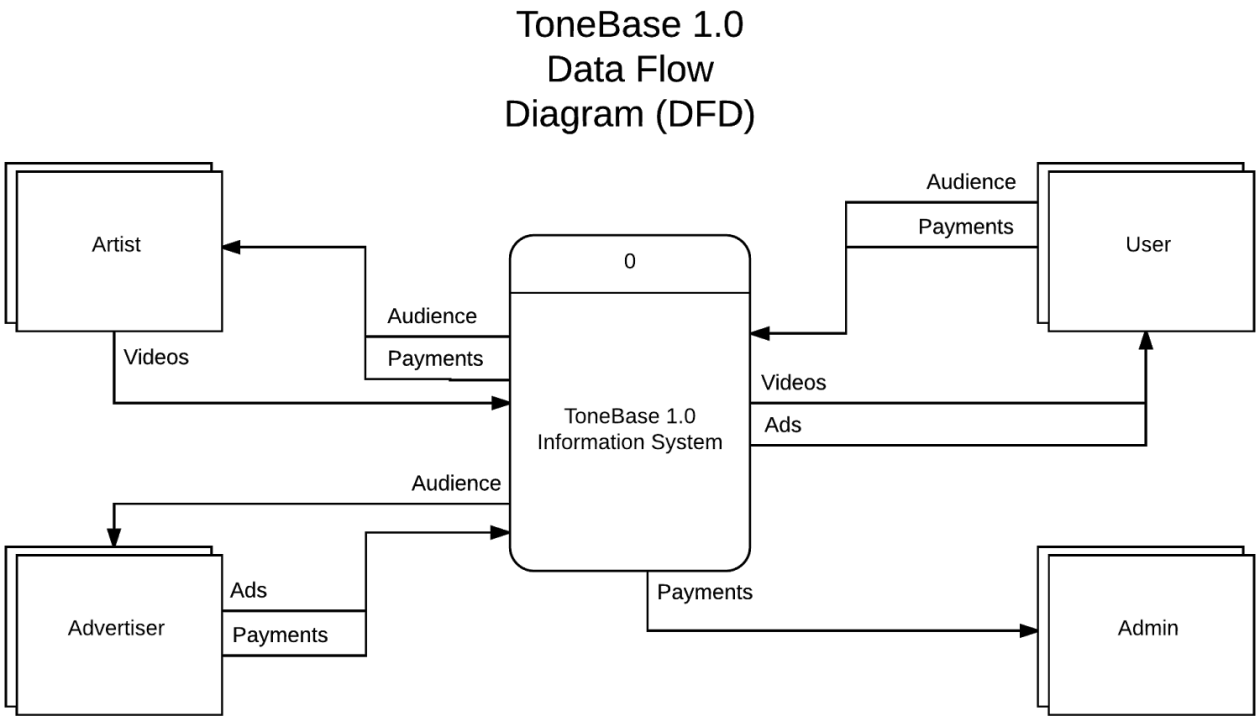
System Description

ToneBase 1.0 will be a web-based information system designed to facilitate online music instruction through the posting, curation, and viewing of online videos. See Information Requirements and System Functionality for more information.

From a technical perspective, ToneBase 1.0 will comprise a client-facing web application (“Client Application”) as well as a back-end web service (“API Server”). See System Functionality and System Architecture for more information.

Information Requirements

This diagram depicts system information inputs and outputs at a high level.



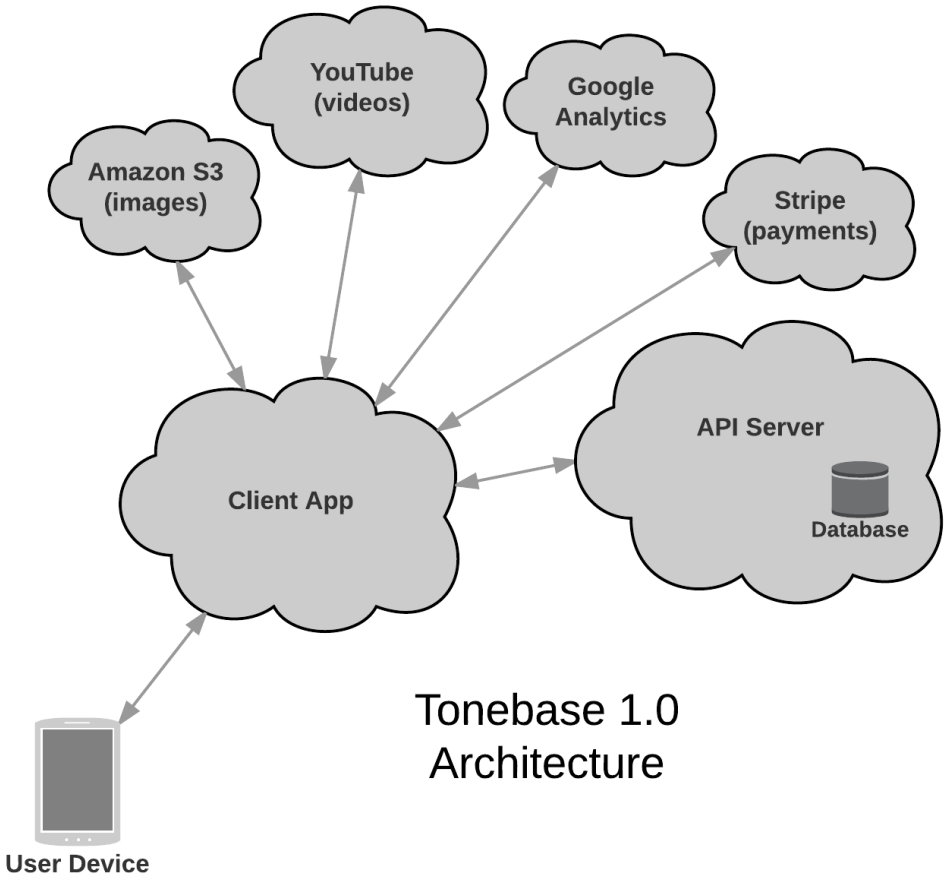
In general, stakeholders input and output information from the system in the following ways:

- Artists post instructional videos in exchange for payments and an audience.
- Users pay for access to videos, and are served advertisements in the process.
- Advertisers pay to place their ads in exchange for a targeted audience.
- Admins receive a percentage of payments.

System Architecture

Network Architecture

This diagram depicts an interconnected network of computers and services which comprise the system.



System users will leverage an Internet-connected device to access the system's Client Application.

The Client Application is responsible for providing all user-facing functionality. It is also responsible for interfacing with various back-end platforms and services:

- The Client Application is responsible for storing user-provided image files on an image service such as Amazon's Simple Storage Service (S3) ("S3"). The Client Application is also responsible for interfacing with the API Server to store and maintain URL references to those image files.

- The Client Application is responsible for uploading user-provided videos to a video service such as YouTube or Vimeo, and for interfacing with the video service to play those videos and

request viewership metrics about those videos. The Client Application is also responsible for interfacing with the API Server to store and maintain URL references to those videos.

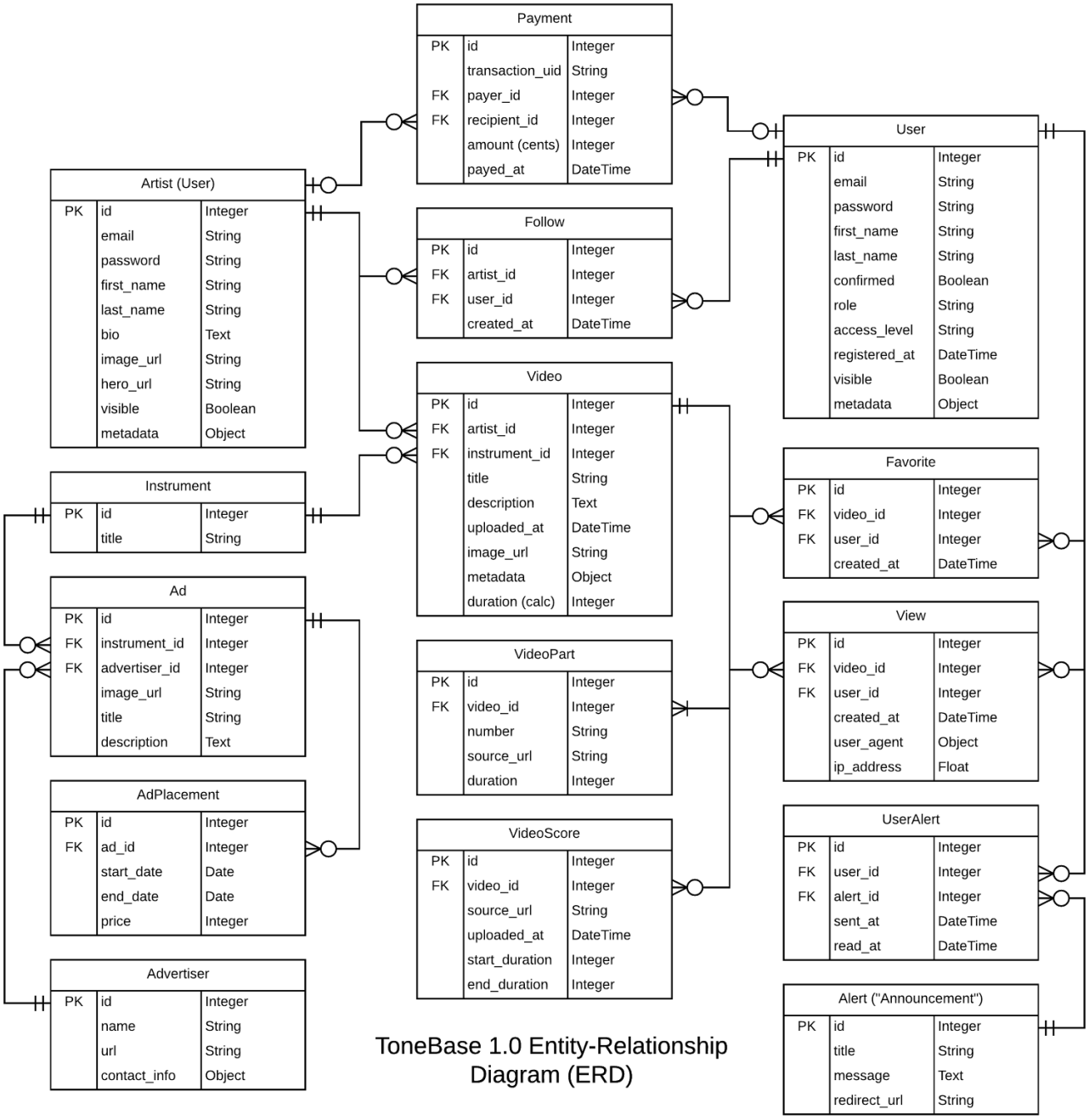
- The Client Application is responsible for interfacing with a payment service such as Stripe to process payments. The Client Application is also responsible for interfacing with the API Server to store and maintain transaction identifier references to those payments.

- Finally, the Client Application is responsible for interfacing with the API Server to fulfill other needs. The API Server is responsible for responding to all requests from the Client Application which are issued to documented API endpoints.

For more information about the API Server, see Database Architecture and API Endpoints.

Database Architecture

This diagram depicts the architecture of a relational database to store system data.



ToneBase 1.0 Entity-Relationship Diagram (ERD)

User Roles and Permissions

At any given time, a single user assumes exactly one user role. The four specific user roles are “Visitor”, “User”, “Artist”, and “Admin”, and they are described as follows:

- The Visitor role includes non-users, as well as any users who are not logged-in.
- The User role represents music students who consume system content.
- The Artist role includes professional musicians and music instructors who contribute videos and other content to the system.
- The Admin role includes the ToneBase team and any other individual responsible for maintaining and monitoring the system.

Visitor Permissions

The table below depicts a list of functions a visitor is allowed to perform on the site.

			Visitor Permissions					
Dev Id	**Dev Group**	**Resource(s)**	List	Show	Create	Update	Destroy	Notes
1.1	Core	*Instrument*	FALSE	FALSE	FALSE	FALSE	FALSE	
1.2	Core	*User, UserProfile*	FALSE	FALSE	TRUE	FALSE	FALSE	Visitor can register to become a user.
1.3	Core	*Artist, ArtistProfile*	FALSE	FALSE	FALSE	FALSE	FALSE	
2.1	Comms	*Alert*	FALSE	FALSE	FALSE	FALSE	FALSE	
2.2	Comms	*UserAlert*	FALSE	FALSE	FALSE	FALSE	FALSE	
2.3	Comms	*Advertiser*	FALSE	FALSE	FALSE	FALSE	FALSE	
2.4	Comms	*Ad, AdPlacement*	FALSE	FALSE	FALSE	FALSE	FALSE	
3.1	Interactivity	*ArtistFollow*	FALSE	FALSE	FALSE	FALSE	FALSE	
3.2	Interactivity	*Video, VideoPart, VideoScore*	FALSE	FALSE	FALSE	FALSE	FALSE	
3.3	Interactivity	*VideoView*	FALSE	FALSE	FALSE	FALSE	FALSE	
3.4	Interactivity	*VideoFavorite*	FALSE	FALSE	FALSE	FALSE	FALSE	
4.1	Payments	*UserPayment*	FALSE	FALSE	FALSE	FALSE	FALSE	
5.1	Metrics	*VideoMetric*	FALSE	FALSE	FALSE	FALSE	FALSE	
5.2	Metrics	*UsageMetric*	FALSE	FALSE	FALSE	FALSE	FALSE	

User Permissions

The table below depicts a list of functions a user is allowed to perform on the site.

			User Permissions					
API Dev Order	**Dev Group**	**Resource(s)**	**List**	**Show**	**Create**	**Update**	**Destroy**	**Notes**
1.1	Core	*Instrument*	TRUE	TRUE	FALSE	FALSE	FALSE	
1.2	Core	*User, UserProfile*	FALSE	TRUE	FALSE	TRUE	FALSE	User can view and edit only his/her own profile. User can mark account as "de-activated" but cannot delete their record from the database.
1.3	Core	*Artist, ArtistProfile*	TRUE	TRUE	FALSE	FALSE	FALSE	User can also view curated, filtered, sorted lists of artists.
2.1	Comms	*Alert*	TRUE	TRUE	FALSE	FALSE	FALSE	User can view list of generic alerts/announcements on home page.
2.2	Comms	*UserAlert*	TRUE	TRUE	FALSE	TRUE	FALSE	User can view and mark as read or unread an inbox of targeted alert messages.
2.3	Comms	*Advertiser*	FALSE	FALSE	FALSE	FALSE	FALSE	
2.4	Comms	*Ad, AdPlacement*	TRUE	TRUE	FALSE	FALSE	FALSE	User can view list of generic ads on home page.
3.1	Interactivity	*ArtistFollow*	FALSE	FALSE	TRUE	FALSE	TRUE	User can follow or unfollow an artist.
3.2	Interactivity	*Video, VideoPart, VideoScore*	TRUE	TRUE	FALSE	FALSE	FALSE	User can also view curated, filtered, sorted lists of artists.
3.3	Interactivity	*VideoView*	TRUE	TRUE	TRUE	FALSE	FALSE	User can view a video and see a list of their own video viewing history.
3.4	Interactivity	*VideoFavorite*	FALSE	FALSE	TRUE	FALSE	TRUE	User can favorite or unfavorite a video.
4.1	Payments	*UserPayment*	TRUE	TRUE	FALSE	FALSE	FALSE	User can see that user's own payment history.
5.1	Metrics	*VideoMetric*	FALSE	FALSE	FALSE	FALSE	FALSE	
5.2	Metrics	*UsageMetric*	FALSE	FALSE	FALSE	FALSE	FALSE	

Artist Permissions

The table below depicts a list of functions an artist is allowed to perform on the site.

			Artist Permissions					
Dev Id	**Dev Group**	**Resource(s)**	**List**	**Show**	**Create**	**Update**	**Destroy**	**Notes**
1.1	Core	*Instrument*	TRUE	TRUE	FALSE	FALSE	FALSE	
1.2	Core	*User, UserProfile*	TRUE	FALSE	FALSE	FALSE	FALSE	Artist can only see a list of users who have either followed them or favorited one of their videos.
1.3	Core	*Artist, ArtistProfile*	TRUE	TRUE	FALSE	TRUE	FALSE	Artist can view all other artists, but only update his/her own profile. Artist can mark account as "de-activated" or "hidden" but cannot delete their record from the database.
2.1	Comms	*Alert*	TRUE	TRUE	FALSE	FALSE	FALSE	Artist can view list of generic alerts/announcements on home page.
2.2	Comms	*UserAlert*	FALSE	FALSE	FALSE	FALSE	FALSE	
2.3	Comms	*Advertiser*	FALSE	FALSE	FALSE	FALSE	FALSE	
2.4	Comms	*Ad, AdPlacement*	TRUE	TRUE	FALSE	FALSE	FALSE	Artist can view list of ads on home page.
3.1	Interactivity	*ArtistFollow*	FALSE	FALSE	FALSE	FALSE	FALSE	
3.2	Interactivity	*Video, VideoPart, VideoScore*	TRUE	TRUE	TRUE	TRUE	TRUE	Artist can view all videos, but only create, edit, and delete their own videos.
3.3	Interactivity	*VideoView*	TRUE	TRUE	FALSE	FALSE	FALSE	Artist can see list of views for each of their own videos.
3.4	Interactivity	*VideoFavorite*	FALSE	FALSE	FALSE	FALSE	FALSE	
4.1	Payments	*UserPayment*	FALSE	FALSE	FALSE	FALSE	FALSE	
5.1	Metrics	*VideoMetric*	TRUE	TRUE	FALSE	FALSE	FALSE	Artist can see detailed viewership metrics for each of their own videos. These metrics are provided by the video service.

5.2	Metrics	*UsageMetric*	FALSE	FALSE	FALSE	FALSE	FALSE	

Admin Permissions

The table below depicts a list of functions an artist is allowed to perform on the site.

			Admin Permissions					
Dev Id	**Dev Group**	**Resource(s)**	**List**	**Show**	**Create**	**Update**	**Destroy**	**Notes**
1.1	Core	*Instrument*	TRUE	TRUE	TRUE	TRUE	TRUE	Admin can manage instruments.
1.2	Core	*User, UserProfile*	TRUE	TRUE	FALSE	TRUE	TRUE	Admin can view all users and change any user's role from "User" to either "Artist" or "Admin". Admins can "de-activate" or "hide" a user, and in rare cases remove the record from the database.
1.3	Core	*Artist, ArtistProfile*	TRUE	TRUE	FALSE	TRUE	TRUE	Admin can manage artist profiles on behalf of the artist. Admins can "de-activate" or "hide" an artist, and in rare cases remove the record from the database.
2.1	Comms	*Alert*	TRUE	TRUE	TRUE	TRUE	TRUE	Admin can manage alerts/announcements.
2.2	Comms	*UserAlert*	TRUE	TRUE	TRUE	TRUE	TRUE	Admin can assign alerts to be sent to a specific user or group of users.
2.3	Comms	*Advertiser*	TRUE	TRUE	TRUE	TRUE	TRUE	Admin can manage advertisers.
2.4	Comms	*Ad, AdPlacement*	TRUE	TRUE	TRUE	TRUE	TRUE	Admin can manage ads and placements.
3.1	Interactivity	*ArtistFollow*	FALSE	FALSE	FALSE	FALSE	FALSE	
3.2	Interactivity	*Video, VideoPart, VideoScore*	TRUE	TRUE	TRUE	TRUE	TRUE	Admin can view all videos and manage videos on behalf of an artist.
3.3	Interactivity	*VideoView*	TRUE	FALSE	FALSE	FALSE	FALSE	Admin can see a list of video-view events over time.
3.4	Interactivity	*VideoFavorite*	FALSE	FALSE	FALSE	FALSE	FALSE	
4.1	Payments	*UserPayment*	TRUE	TRUE	FALSE	FALSE	FALSE	Admin can see a list of payments for all users or a specific user.

5.1	Metrics	*VideoMetric*	TRUE	TRUE	FALSE	FALSE	FALSE	Admin can see detailed viewership metrics for each video, and aggregated metrics for all videos uploaded by a given artist. These metrics are provided by the video service.
5.2	Metrics	*UsageMetric*	TRUE	TRUE	FALSE	FALSE	FALSE	Admin can see a history of their visits, as well as a history of user and artist visits. This information comes from Google Analytics.

System Functionality

This section generally describes a non-comprehensive list of the most important functionality a user can expect from the system, as well as the general responsibilities of the Client Application and the API Server in providing that functionality. This section is superseded by the official Web Service Documentation.

It includes specific expectations about the structure of information passed from the client to the server, except fields called “metadata” which provide the client flexibility in choosing and even changing over time the structure of information contained therein.

This section comprises four logical components, each corresponding to one of the four user roles.

Visitor Functions

Registration

New User

User Experience

A visitor should be able to visit one or more publicized public-facing registration forms and/or interest surveys. The visitor should be able to input and submit profile information through these forms. After submitting, the visitor should be prompted to click a confirmation link in an email sent to them by the system. After clicking the confirmation link, the visitor becomes a user.

Additionally, there may be an opportunity to authenticate using social platforms (e.g. Facebook, Twitter, Google).

Client Responsibilities

The client should issue a POST request to **/api/users**. The request body should indicate the user’s access level (e.g. either “Full” or “Limited” access), and should resemble the following JSON object:

```
{
    first_name: String,
    last_name: String,
    email: String,
    password: Encrypted String,
    role: “User” (optional, implied)
    access_level: ”Full/Limited”,
    visible: Boolean,
    metadata: {
        image_url: String,
        bio: Text,
        social_urls: {facebook: String, twitter: String, etc.},
```

instruments_played: [String, String, etc.],
instruments_owned: Text,
preferred_musical_period: String,
favorite_artist: String,
favorite_composer: String,
birth_year: Integer,
professions: [String, String, etc.]
}
}

Server Responsibilities
The server will create a new User record. The user's "profile" will reflect the unstructured data contained in the metadata field.

New Artist

User Experience
After a visitor completes the New User process, an existing admin should be able to complete the Edit User Role process to change the user's role from "User" to "Artist", at which time the user becomes an artist. Then, either an admin or that artist should be able to perform the Edit Artist Profile process to manage the artist's profile information.

Client Responsibilities
When the client issues its request during the New User process, there is no need to specify any metadata for users that are intended to become artists.

New Admin

User Experience
After a visitor completes the New User process, an existing admin should be able to complete the Edit User Role process to change the user's role from "User" to "Admin", at which time the user becomes an admin.

Client Responsibilities
When the client issues its request during the New User process, there is no need to specify any metadata for users that are intended to become admins.

Artist Functions

Artist Profile

View Artist Profile

User Experience
After an artist submits an artist profile and marks it as being publicly-visible, all visitors, users, artists, and admins should be able to view that artist's profile information. The artist profile will also include a

list of all the artist's visible videos. If the artist is viewing his/her own profile, the artist will see links to edit each video.

Client Responsibilities

The client should issue a GET request to **/api/artists/:id**. The request's URL parameters may optionally include a videos parameter (e.g. **/api/artists/:id?videos=true**) to indicate a request to include video information in the response.

Server Responsibilities

The server will return an Artist object containing a nested Artist-Profile object, and optionally a nested array of Video objects (see View Video).

Edit Artist Profile

User Experience

An artist or admin should be able to edit that artist's profile information.

Client Responsibilities

The client should issue a PUT request to **/api/artists/:id**. The request body should resemble the following JSON object:

```
{
    first_name: String,
    last_name: String,
    email: String,
    password: Encrypted String,
    role: "Artist", (optional, implied)
    access_level: "Full" (optional, implied)
    visible: Boolean,
    metadata: {
        url: String,
        image_url: String,
        hero_url: String,
        social_urls: {facebook: String, twitter: String, youtube: String, etc.},
        bio: Text,
        education_info: Text,
        teaching_info: Text,
        work_info: Text,
        awards: Text,
        touring: {
            status: String,
            shows: [{date: Date, location: String, venue: String}, {}, {}, etc]
        }
    }
}
```

Server Responsibilities

The server will update the artist's User record. This includes updating the user's "profile" to reflect the unstructured metadata passed by the client.

Videos

New Video

User Experience

The artist should be able to upload a video. During this process, the artist should be able to specify the video's information, optionally ensure proper split times between video parts, and upload one or more scores. Each score must be assigned a unique, non-overlapping, video duration (associated with the global video object so it can overlap between video parts if necessary).

Client Responsibilities

When the user submits the final form, the client should issue a POST request to **/api/videos**. The request body should resemble the following JSON object:

```
{
    video: {
        instrument_id: Integer,
        title: String,
        image_url: String,
        duration: Integer (milliseconds or whatever the video service uses)
        description: Text,
        metadata: {
            tags: [String, String, etc.],
            composer: String,
            period: String,
            techniques_covered: [String, String, etc.]
        },
        parts: [
            {source_url: String, duration: Integer},
            {source_url: String, duration: Integer},
            {source_url: String, duration: Integer},
            etc.
        ],
        scores: [
            {image_url: String, start: Integer, end: Integer},
            {image_url: String, start: Integer, end: Integer},
            etc.
        ]

    }
```

```
}
```

Server Responsibilities
The server will create a new Video record, as well as one or more Video-Part records, as well as zero or more Score records.

Edit Video

User Experience
The artist should be able to edit certain information about their video, perhaps including the ability to revise and re-upload scores.

Client Responsibilities
The client should issue a PUT request to **/api/videos/:id**. The request body should contain the entire video object (see request body from New Video).

Remove Video

User Experience
An artist or admin should be able to delete one of that artist's own videos.

Client Responsibilities
The client should issue a DELETE request to **/api/videos/:id**.

User Functions

Ads

List Ads

User Experience
All visitors, users, artists, and admins should be able to view a list of visible ads currently being placed.

Client Responsibilities
The client should issue a GET request to **/api/ads**.

Server Responsibilities
The server will return an array of visible Ad objects, including a nested Advertiser object.

Alerts

List Alerts

User Experience
All visitors, users, artists, and admins should be able to view a list of visible alerts (a.k.a. "announcements").

Client Responsibilities
The client should issue a GET request to **/api/alerts**.

Server Responsibilities
The server will return an array of visible Alert objects.

Artists

List Artists

User Experience
All visitors, users, artists, and admins should be able to view a list of all visible artists.

Client Responsibilities
The client should issue a GET request to **/api/artists**.

Server Responsibilities
The server will return an array of visible Artist objects, each containing a nested Artist-Profile object.

Follow Artist

User Experience
A user should be able to follow an artist.

Client Responsibilities
The client should send a POST request to **/api/follows**. The request body should contain the following information:

```
{
    artist_id: Integer,
    user_id: Integer
}
```

Server Responsibilities
The server will create a corresponding Follow record.

Unfollow Artist

User Experience
A user should be able to unfollow an artist.

Client Responsibilities
The client should send a DELETE request to **/api/favorites**. The request body should contain the following information:

```
{
    artist_id: Integer,
```

```
        user_id: Integer
}
```

Server Responsibilities
The server will delete the corresponding Follow record.

User Alerts

View User Alerts

User Experience
A user should be able to view a personalized inbox of alert messages. The user should be able to mark each as being “read” or “unread”.

Client Responsibilities
The client will issue a GET request to **/api/users/:id/alerts**.

Server Responsibilities
The server will respond with an array of Alert objects.

User Payments

New Payment

User Experience
A user should be able to provide credit card information and select an appropriate billing plan. The payment service will automatically bill them. See User Profile.

Client Responsibilities
The client will interface with the Payment Service to facilitate payments.

List Payments

User Experience
A user should be able to view a history of payments made to the service.

Client Responsibilities
If the client is unable to interface with the payment service to facilitate the desired user experience, and if the server has been configured to interface with the payment service, the client should issue a GET request to **/api/users/:id/payments**.

Server Responsibilities
If the client is unable to interface with the payment service to facilitate the desired user experience, the server may become responsible for fetching relevant information from the payment service on a nightly basis and making this information available to the client. If this is the case, the server will return an array of Payment objects upon request.

User Profile

View User Profile

User Experience
A user or admin should be able to view that user's profile information.

Client Responsibilities
The client should issue a GET request to **/api/users/:id**.

Server Responsibilities
The server will return a User object containing a nested User-Profile object.

Edit User Profile

User Experience
A user or admin should be able to edit that user's profile information.

Client Responsibilities
The client should issue a PUT request to **/api/users/:id**. The request body should resemble the JSON object sent during the New User process.

Server Responsibilities
The server will update the User record. This includes updating the user's "profile" to reflect the unstructured metadata passed by the client.

Videos

List Videos

User Experience
All visitors, users, artists, and admins should be able to view a list of visible videos. The user should have the option to view one of the following filtered lists:

- Favorite Videos
- Most Recent Videos
- Popular Videos
- Videos Posted by Artists I Follow (default)
- Videos Recommended for Me
- Videos Recently Viewed by Me (3 by default)

Client Responsibilities
The client should issue a GET request to **/api/videos**. The request's URL parameters may optionally specify one of the available sorting mechanisms ("Favorites", "Recent", "Popular", "Following", and "Recommended", respectively), and may optionally specify limit and offset parameters to handle pagination considerations (e.g. **api/videos?filter=Recent&limit=25&offset=50**).

Server Responsibilities
The server will return an array of visible Video objects, each containing a nested Artist object, a nested array of Video-Part objects, and a nested Video-Score object.

View/Play Video

User Experience
Artists, admins, and certain users (depending on their access level) should be able to view a given video.

Client Responsibilities
The client may issue a GET request to **/api/videos/:id**. Regardless, the client should interface with the Video Service to render and allow playback of the video, and render parts appropriately. The client should also interface with the Image Service to render scores appropriately.

Server Responsibilities
The server will return a Video object containing a nested Artist object, a nested array of Video-Part objects, and a nested Video-Score object.

Favorite Video

User Experience
A user should be able to add a video to their list of favorites.

Client Responsibilities
The client should send a POST request to **/api/favorites**. The request body should contain the following information:

```
{
        video_id: Integer,
        user_id: Integer
}
```

Server Responsibilities
The server will create a corresponding Favorite record.

Unfavorite Video

User Experience
A user should be able to remove a video from their list of favorites.

Client Responsibilities
The client should send a DELETE request to **/api/favorites**. The request body should contain the following information:

```
{
```

```
        video_id: Integer,
        user_id: Integer
}
```

Server Responsibilities
The server will delete the corresponding Favorite record.

Admin Functions

Ads

Manage Ads and Advertisers

User Experience
An admin should be able to view, create, update, and delete information about ads, advertisers, and ad placements throughout the system.

Alerts

Manage Alerts

User Experience
An admin should be able to view, create, update, and delete information about alerts (a.k.a “announcements” placed throughout the system.

Artists

Deactivate Artist

User Experience
An admin or an artist should be able to prevent an artist’s information from appearing in user-facing parts of the system.

Client Responsibilities
The client should perform the Edit Artist Profile process, marking the artist’s visibility status as false.

Instruments

List Instruments

User Experience
All visitors, users, artists, and admins should be able to view a list of all visible instruments.

Client Responsibilities
The client should issue a GET request to **/api/instruments**.

Server Responsibilities
The server will return an array of visible Instrument objects.

View Instrument

User Experience
All visitors, users, artists, and admins should be able to view information about a specific instrument.

Client Responsibilities
The client should issue a GET request to **/api/instruments/:id**.

Server Responsibilities
The server will return an Instrument object.

New Instrument

User Experience
An admin should be able to create a new instrument.

Client Responsibilities
The client should issue a POST request to **/api/instruments**. The request body should resemble the following JSON object:

```
{
        name: String,
        description: Text,
        image_url: String
}
```

Edit Instrument

User Experience
An admin should be able to edit information about a specific instrument.

Client Responsibilities
The client should issue a PUT request to **/api/instruments/:id**. The request body should resemble the one sent during New Instrument.

Delete Instrument

User Experience
An admin should be able to delete an instrument.

Client Responsibilities
The client should issue a DELETE request to **/api/instruments/:id**.

Server Responsibilities

The server will delete the corresponding instrument record, as well as any related video records or ad records, from the database.

Users

List Users

User Experience
Admins should be able to view a list of all users.

Client Responsibilities
The client should issue a GET request to **/api/users**.

Server Responsibilities
The server will return an array of visible User objects, each containing a nested User-Profile object.

Edit User Role

User Experience
Admins should be able to change a user's role.

Client Responsibilities
The client should issue a PUT request to **/api/users/:id**. The request body should indicate the user's role ("User", "Artist", or "Admin"), and if the role is "User" then the client should also specify which access level the user should have ("Full" or "Limited").

Server Responsibilities
The server will change the User record's role and access level. The server will also wipe that user's metadata (profile).

Deactivate User

User Experience
An admin or a user should be able to prevent that user's information from appearing in user-facing parts of the system.

Client Responsibilities
The client should perform the Edit User Profile process, marking the user's visibility status as false.

Usage Metrics

View Usage Metrics

User Experience
An admin should be able to view web application usage metrics.

Client Responsibilities
The client is responsible for interfacing with the Google Analytics Service to request usage metrics and share the results. To facilitate the collection of usage data whenever a user visits any page, the client is responsible for sending a pageview event along with the user's identifier to Google Analytics.

For some simple usage metrics such as the number of current total users or the number of registrations over time, the client should make a GET request to **/api/metrics/...** and indicate the desired metric:

- **/api/metrics/users-total**
- **/api/metrics/users-over-time**

Server Responsibilities
For the "users-total" metric, the server will respond with a single integer number.

For the "users-over-time" metric, the server will respond with an array of small User-Registration objects resembling the following:

```
[
        {user_id: Integer, registered_at: DateTime},
        {user_id: Integer, registered_at: DateTime},
        {user_id: Integer, registered_at: DateTime},
        etc.
]
```

Video Metrics

View Video Metrics

User Experience
An artist or an admin should be able to view viewership metrics about each of the artist's videos, as well as for all videos published by that artist. In addition, admins should be able to view metrics about all videos across the site.

Client Responsibilities
The client is responsible for interfacing with the Video Service to request video metrics and share the results.

For some simple video metrics such as the number of current total videos or the number of uploads over time, the client should make a GET request to **/api/metrics/...** and indicate the desired metric:

- **/api/metrics/videos-total**
- **/api/metrics/video-uploads-over-time**
- **/api/metrics/video-views-over-time**

Server Responsibilities
For the "videos-total" metric, the server will respond with a single integer number.

For the “video-uploads-over-time” metric, the server will respond with an array of small Video-Upload objects resembling the following:

```
[
    {video_id: Integer, uploaded_at: DateTime},
    {video_id: Integer, uploaded_at: DateTime},
    {video_id: Integer, uploaded_at: DateTime},
    etc.
]
```

For the “video-views-over-time” metric, the server will respond with an array of small Video-View objects resembling the following:

```
[
    {video_id: Integer, viewed_at: DateTime},
    {video_id: Integer, viewed_at: DateTime},
    {video_id: Integer, viewed_at: DateTime},
    etc.
]
```

Payment Metrics

View Payment Metrics

User Experience

An admin should be able to view viewership metrics about all payments across the site.

Client Responsibilities

The client is responsible for interfacing with the Payment Service to request payment metrics and share the results. If the client is unable to interface with the payment service to facilitate the desired user experience, and if the server has been configured to interface with the payment service, the client may make requests to the server for payment metrics. In this case, the client should make a GET request to **/api/metrics/...** and indicate the desired metric:

- **/api/metrics/payments-total**
- **/api/metrics/payments-over-time**

Server Responsibilities

If the client is unable to interface with the payment service to facilitate the desired user experience, and if the server has been configured to interface with the payment service, the server would be responsible for the following activities:

For the “payments-total” metric, the server will respond with an object resembling the following JSON object:

{count: Integer, sum: Integer (cents)}

For the “payments-over-time” metric, the server will respond with an array of small Payment objects resembling the following:

```
[
    {transaction_id: Integer, payment_at: DateTime, amount: Integer (cents)},
    {transaction_id: Integer, payment_at: DateTime, amount: Integer (cents)},
    {transaction_id: Integer, payment_at: DateTime, amount: Integer (cents)},
    etc.
]
```

Web Service Documentation

API Endpoints

The table below contains a tentative list of API Server endpoints and provides high-level instructions on how the Client Application should expect to interface with each. Refer to the Official API Endpoint Documentation for an up-to-date list.

Dev Group	Primary Resource	Action	Request Method	Endpoint URL
1-Core	Artist (User)	1-List	GET	/api/artists
1-Core	Artist (User)	3-Show	GET	/api/artists/:id
1-Core	Artist (User)	4-Update	PUT	/api/artists/:id
1-Core	Artist (User)	5-Destroy	DELETE	/api/artists/:id
1-Core	Instrument	1-List	GET	/api/instruments
1-Core	Instrument	2-Create	POST	/api/instruments
1-Core	Instrument	3-Show	GET	/api/instruments/:id
1-Core	Instrument	4-Update	PUT	/api/instruments/:id
1-Core	Instrument	5-Destroy	DELETE	/api/instruments/:id
1-Core	User	1-List	GET	/api/users
1-Core	User	2-Create	POST	/api/users
1-Core	User	3-Show	GET	/api/users/:id
1-Core	User	4-Update	PUT	/api/users/:id
1-Core	User	5-Destroy	DELETE	/api/users/:id
2-Comms	Ad	1-List	GET	/api/ads
2-Comms	Ad	2-Create	POST	/api/ads
2-Comms	Ad	3-Show	GET	/api/ads/:id
2-Comms	Ad	4-Update	PUT	/api/ads/:id
2-Comms	Ad	5-Destroy	DELETE	/api/ads/:id
2-Comms	AdPlacement	1-List	GET	/api/ads/:id/placements
2-Comms	AdPlacement	2-Create	POST	/api/ads/:id/placements
2-Comms	AdPlacement	3-Show	GET	/api/ads/:id/placements/:id
2-Comms	AdPlacement	4-Update	PUT	/api/ads/:id/placements/:id
2-Comms	AdPlacement	5-Destroy	DELETE	/api/ads/:id/placements/:id
2-Comms	Advertiser	1-List	GET	/api/advertisers
2-Comms	Advertiser	2-Create	POST	/api/advertisers
2-Comms	Advertiser	3-Show	GET	/api/advertisers/:id

2-Comms	Advertiser	4-Update	PUT	/api/advertisers/:id
2-Comms	Advertiser	5-Destroy	DELETE	/api/advertisers/:id
2-Comms	Alert	1-List	GET	/api/alerts
2-Comms	Alert	2-Create	POST	/api/alerts
2-Comms	Alert	3-Show	GET	/api/alerts/:id
2-Comms	Alert	4-Update	PUT	/api/alerts/:id
2-Comms	Alert	5-Destroy	DELETE	/api/alerts/:id
2-Comms	UserAlert	1-List	GET	/api/users/:id/alerts
2-Comms	UserAlert	2-Create	POST	/api/users/:id/alerts
2-Comms	UserAlert	3-Show	GET	/api/users/:id/alerts/:id
2-Comms	UserAlert	4-Update	PUT	/api/users/:id/alerts/:id
2-Comms	UserAlert	5-Destroy	DELETE	/api/users/:id/alerts/:id
3-Interactivity	Favorite	2-Create	POST	/api/favorites
3-Interactivity	Favorite	5-Destroy	DELETE	/api/favorites
3-Interactivity	Follow	2-Create	POST	/api/follow
3-Interactivity	Follow	5-Destroy	DELETE	/api/follow
3-Interactivity	Video	1-List	GET	/api/videos
3-Interactivity	Video	2-Create	POST	/api/videos
3-Interactivity	Video	3-Show	GET	/api/videos/:id
3-Interactivity	Video	4-Update	PUT	/api/videos/:id
3-Interactivity	Video	5-Destroy	DELETE	/api/videos/:id
3-Interactivity	View	1-List	GET	/api/views/
3-Interactivity	View	2-Create	POST	/api/views/
3-Interactivity	View	3-Show	GET	/api/views/:id
4-Payments	Payment	1-List	GET	/api/payments
4-Payments	Payment	3-Show	GET	/api/payments/:id