Design Document

CS 307
Iguana Team

Alec Gorge, Manik Kalra, Scott Opell, Andrew Shildmyer, Kirby Kohlmorgen

Purpose

Our project will upgrade existing applications (iOS and Web) to add features and consolidate existing elements. We will add alternate interfaces such as a chrome extension and possibly a Chromecast application. The Iguana Suite is a pre-existing suite of software developed by Alec Gorge that provides programmatic access to hundreds of artists and hundreds of thousands of songs.

Background + Terminology

The Iguana Suite is a project that Alec Gorge started last September. It is a term that encompases the API server, database and all clients. *iguana* is the project name and the abbreviation “IG” or “ig” is used significantly throughout the code where traditional namespacing isn’t available. Despite the prevalence of iguana terminology throughout the codebase, the end-user almost never sees *iguana* anywhere.

The GUIs (iguana-ios, iguana-chrome and iguana-web) that make use of the core iguana project and API are branded as “relisten.net”. A domain name that Alec owns and operates. This distinction is made because the GUIs previously were branded specifically to each artist (Marco All Day, STS9OD, etc) but one of the core tenants of this project is making the music from all of the artists available in one unified user experience per platform.

When Alec started this project, he hoped to eventually make all of the music indexed by iguana available to as many platforms as possible, starting with iOS. This means that the various components of iguana are very loosely coupled and interact exclusively via RESTful, HTTP, JSON-based APIs.

iguana

iguana refers to the API server written in CoffeeScript on node.js. It utilizes the Archive.org Live Music Archive API as the upstream source of the data. The data is analyzed, cleansed, indexed and organized and stored in a MySQL database. *iguana* exposes this data through a clean, RESTful API. This API currently has no authentication, rate-limiting or state management. This API will need to support some of these technical features to add many of the features outlined in this Design Document.

All business logic occurs on the server and all clients are “dumb” viewers for the data encapsulated by *iguana*. This does not mean the clients are simple. The clients are quite complex, but it means the clients can focus on making the information beautiful and simple to harness.

All playlists, favorites and user accounts will be stored in the database associated with *iguana.*

iguana-ios

iguana-ios is one of the thin clients to *iguana*. The Xcode project currently has a separate target per artist, generating one app per artist. This has quickly grown unmanageable. These targets need to be reduced down to a single, easy to use application that allows access to all of the artists indexed by *iguana* at once.

iguana-ios has most of the functionality expected out of a basic music player. This project will expand the playback capabilities of the app, add in playlist, favorites and user accounts and maintain the high standard of user-experience *iguana-ios* users have come to expect.

iguana-web

iguana-web and *iguana-ios* should have nearly complete feature parity but many aspects of *iguana-ios* will need to be rethought in the context of a desktop web application to ensure the user experience is just as good.

iguana-chrome

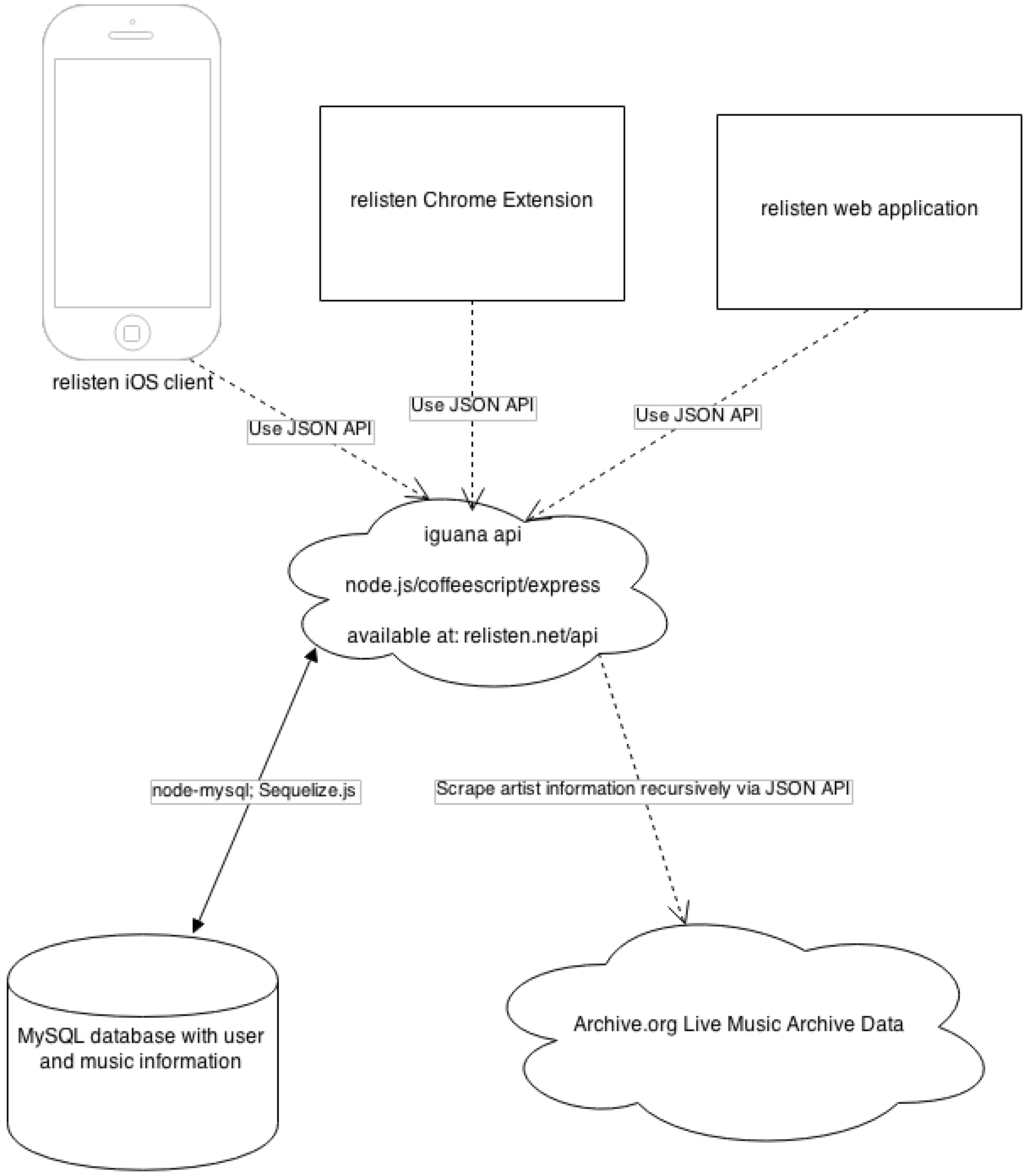
This is another client to *iguana* in yet another context: a Chrome extension. *iguana-web* and *iguana-ios* strive to immerse you in the music and help you find the perfect concert to listen to, *iguana-chrome* is all about quickly identifying some music and then staying out of your way as you continue to work in other tabs. This results in different expectations from users and yet another completely different interface and user-experience.

Integration

Once this project is completed, there will be 3 different clients. It is very likely that users will make use of two or three of these access points which means that terminology and branding should be consistent across all three.

Even if all four major components of this projects are very loosely coupled, they should appear seamless to the end user.

Design Outline


relisten Chrome Extension
relisten web application
relisten iOS client
Use JSON API
Use JSON API
Use JSON API
iguana api
node.js/coffeescript/express
available at: relisten.net/api
node-mysql; Sequelize.js
Scrape artist information recursively via JSON API
MySQL database with user and music information
Archive.org Live Music Archive Data

This diagram outlines the highest level of interactions between the various components in the Iguana Suite

iguana

iguana is built in the MVC (model-view-controller) paradigm, but because *iguana* does not have any UI of its own, *iguana* primarily focuses on the model-controller aspect of MVC.

iguana makes use of an ORM called Sequelize.js for all database interactions. Sequelize.js allows for abstraction of the tables, columns and rows. *iguana* has model objects for each different kind of data (Track, Show, Artist, etc). More models will be added to support the new features this project adds: Playlist, Favorite, User.

iguana uses a framework called *Express* to map URL routes to controllers. These controllers then fetch the relevant data from the database via models and the ORM. The data is then serialized and sent back.

Secure authentication will be a complex addition to *iguana*. Fortunately, there are libraries such as *bcrypt* and *passport.js* that can unify the authentication process that varies per platform (HTTP basic over SSL for *iguana-ios*, sessions and session cookies for *iguana-web* and *iguana-chrome*).

iguana-ios

The native iOS frameworks are very heavily MVC. This means makes integration with *iguana* much easier because *iguana-ios* has an Objective-C equivalent of the model classes. *iguana-ios* has custom views to increase information density without sacrificing readability. All these custom views are managed by custom view controllers. Both receive their data from a class that manages all of the HTTP API requests to *iguana* and hydrates the *iguana* JSON responses into native objects.

The API class is currently state-less. Major modifications will be needed to securely store the username and password and then subsequently transmit them to the server in a safe manner.

iguana-web

No HTML is ever generated server side. All rendering is done client side based on the JSON data fetched from *iguana* via AJAX. *iguana-web* will also be based on MVC, just like *iguana-ios*. Since no HTML is generated by the server (*iguana*), *iguana-web* can be hosted separately or even on a CDN such as Amazon S3. This improves response time and makes the whole project more maintainable.

Design Issues:

Functional Issues

Issue #1: How do we store playlists in the database?

Option A: Mark each track as belonging to a playlist inside the track table.

Option B: Create a table for playlists that has a foreign key reference to both tracks and a user or users.

Decision: Option B is the best option because it will allow for more efficient querying of the database. In option A, it would be incredibly hard to query a list kept on a field on the song. Option B allows all of the querying to stay inside SQL.

Issue #2: How do we store the user’s password securely in the database?

Option A: Store the password in the database right next to the username in plain text.
Option B: Hash the password with BCrypt or similar and store this hash.
Decision: Option B is the best option because its is vastly more secure. Even if the database gets compromised, there is no way to log in with the information in the database.

Issue #3: How should the chrome extension connect with the web player?
Option A: Embed the web player inside of the extension.
Option B: Allow the chrome extension to connect to the player running inside its own tab.
Decision: Option A will allow for the more compatibility and flexibility for the website. Changes in the webplayer might break the chrome extension in Option B.

Issue #4: How should signing in effect the functionality of the iOS app?
Option A: Allow users to listen without signing in but require signing in to view/share personal playlists and favorites.
Option B: Not allow users to do anything without signing up.
Decision: Allow users to listen to all artists and public playlists without signing up, but require signing in to create and share personal playlists and favorite artists and shows.

Issue #5: How do we store relisten.net account passwords?
Option A: Use iOS keychain.
Option B. Store the passwords in plain text in a database.
Decision: Use iOS keychain because it is easier to implement and more secure.

Issue #6: What kind of audio formats should we support on the iOS app.
Option A: MP3 and FLAC.
Option B: Just MP3
Decision: We should support both MP3 and FLAC as not all files are in MP3 format.

Issue #7: How do we validate emails?
Option A: Send each user an email to the email they registered with with a unique key that will validate the user.
Option B: Have the user enter their email twice to prevent typos.
Decision: Both are valid options, but Option A is more bulletproof. Even if the user has to enter in the information twice, they could make the same typo twice or copy and paste with a typo in it.

Non-Functional Issues
Issue #1: How can we develop with performance in mind?
Option A: Research well known performance solutions and utilize them.
Option B: While developing, actively analyze performance and optimize slow segments.
Option C: After developing, analyze performance and optimize slow segments.
Decision: Option C is the best option because in option A and B, we are optimizing prematurely. If the initial solution we come up with has enough performance, then there is no reason to optimize and if we did it early on, then that would simply be wasted time.

Issue #2: How do we make the iOS app user-friendly and attractive?
Option A: Model the app design after existing, successful music streaming iOS apps.
Option B: Create a new UI from scratch.
Decision: Model the UI after existing iOS apps as creating a new UI is time-consuming and tedious. Also, it is better to use existing user-approved UI models for a better experience.

Issue #3: How do make sure that the different iguana interfaces are functioning properly?
Option A: Have surveys inside the interfaces that ask people how well the app works.
Option B: Use a variety of different user analytic tools like New Relic and Crashlytics.
Decision: Option B provides us with the best insight into how well the interfaces are functioning without user interaction.

Design Details

iguana

Models:

Track
Encapsulates a single playable track (FLAC or MP3)
- This is the only item that is playable in the system.
- Attached to shows through a belongs_to relationship
- Each track will have a counter of "favorite"s (basically likes) associated with it

Artist
Describes a single artist that has recordings in the data.
- has_many shows
- has metadata about the artist attached to it

Venue
A venue is a geographical location where concerts would be played. This has no date attached to it
- has_many shows

Show
A show is a specific concert performed at a venue on a given date.
- has_many tracks
- belongs_to venue, artist

Favorite
An entry in the favorite table will be created each time a user likes a track.
- Allows us to limit users to like tracks only once.

- To avoid recalculating the number of favorites, we can store a cache on each track and only modify when we see a change in the Favorites table.

Year

Encapsulates and caches statistics about a year of shows for an artist

User

An entry in the user table will represent an actual user of this service. They will register with an email, username, and password.

- Validations on data:
  - checking for duplicate accounts
  - checking validity of the email
  - enforcing password requirements
- This will also have various auxiliary functions attached to it such as:
  - resetting password
  - logging in/out
  - changing account information
- Links to other tables:
  - has_many playlists
  - has_many favorites

Playlist

A playlist will basically be a set of tracks that are all attached to a user. Each user will be able to create their own playlists and manage them in various ways, such as rearranging them, etc.

- has_many tracks
- belongs_to user
- ordered
- potentially shareable/collaborative

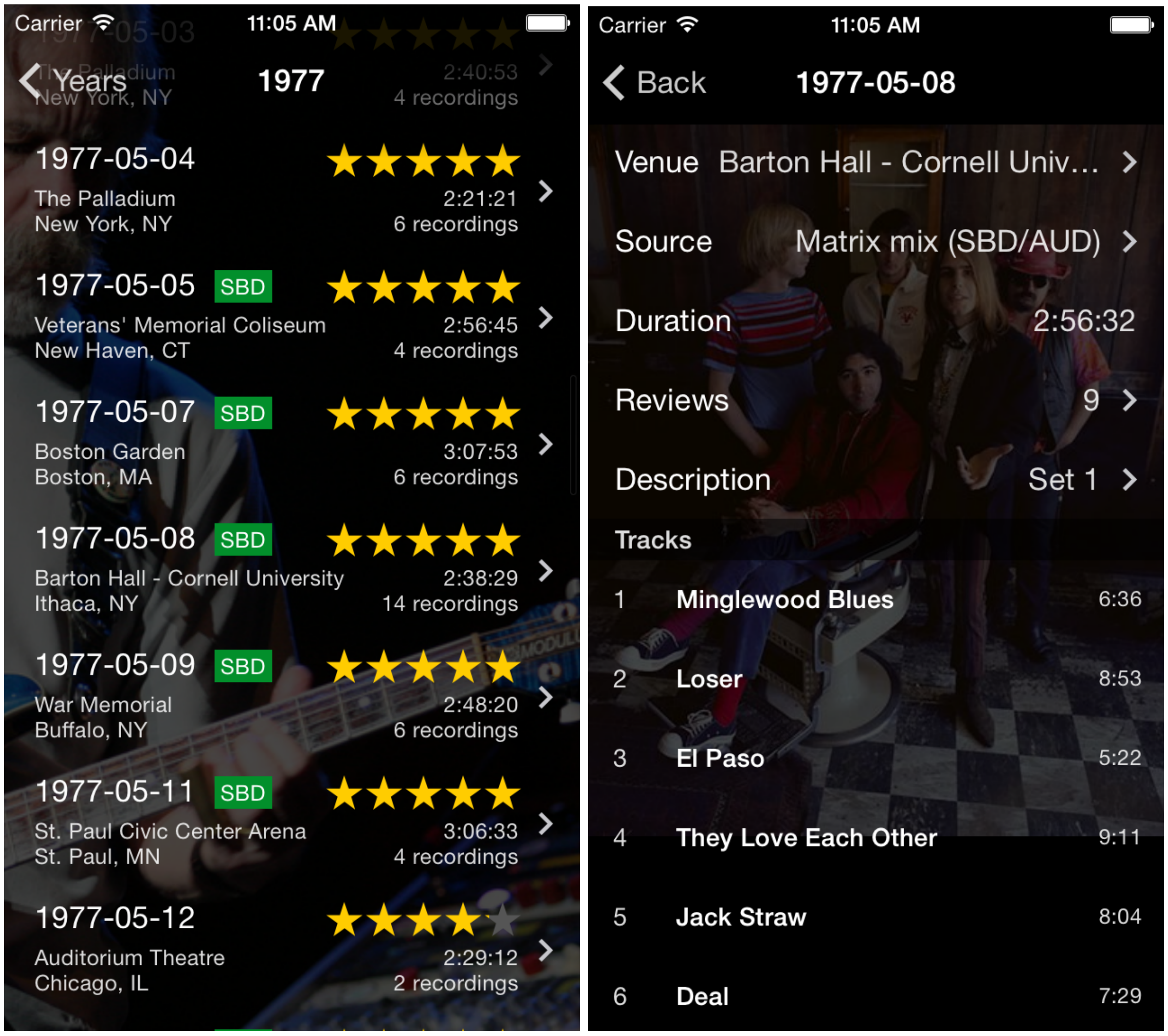
iguana-ios

There will be a corresponding model in iOS for every model on the server side: IGTrack, IGShow, IGVenue, IGYear, IGUser, IGFavorite and IGPlaylist.

The iOS app will have support for every artist, fetched via the iguana API from archive.org, instead of just a few. It will also allow users to create new relisten.net accounts, or log into existing accounts.

Logging into the iOS app through their relisten.net acount will open up multiple features for the users. These features include custom playlists; users will be able to create their own custom playlists, save/share them with other users, and view other user's playlists. It will also allow users to favorite playlists, tracks, artists and shows. This information will be sent to the relisten.net databases via POST.

All this user specific information will be stored on the relisten.net cloud and will downloaded to the iOS app in JSON, and parsed accordingly.


Carrier 11:05 AM

‹ Years **1977**

1977-05-04 The Palladium New York, NY	★★★★★ 2:21:21 6 recordings	›
1977-05-05 SBD Veterans' Memorial Coliseum New Haven, CT	★★★★★ 2:56:45 4 recordings	›
1977-05-07 SBD Boston Garden Boston, MA	★★★★★ 3:07:53 6 recordings	›
1977-05-08 SBD Barton Hall - Cornell University Ithaca, NY	★★★★★ 2:38:29 14 recordings	›
1977-05-09 SBD War Memorial Buffalo, NY	★★★★★ 2:48:20 6 recordings	›
1977-05-11 SBD St. Paul Civic Center Arena St. Paul, MN	★★★★★ 3:06:33 4 recordings	›
1977-05-12 Auditorium Theatre Chicago, IL	★★★★☆ 2:29:12 2 recordings	›

Carrier 11:05 AM

‹ Back **1977-05-08**

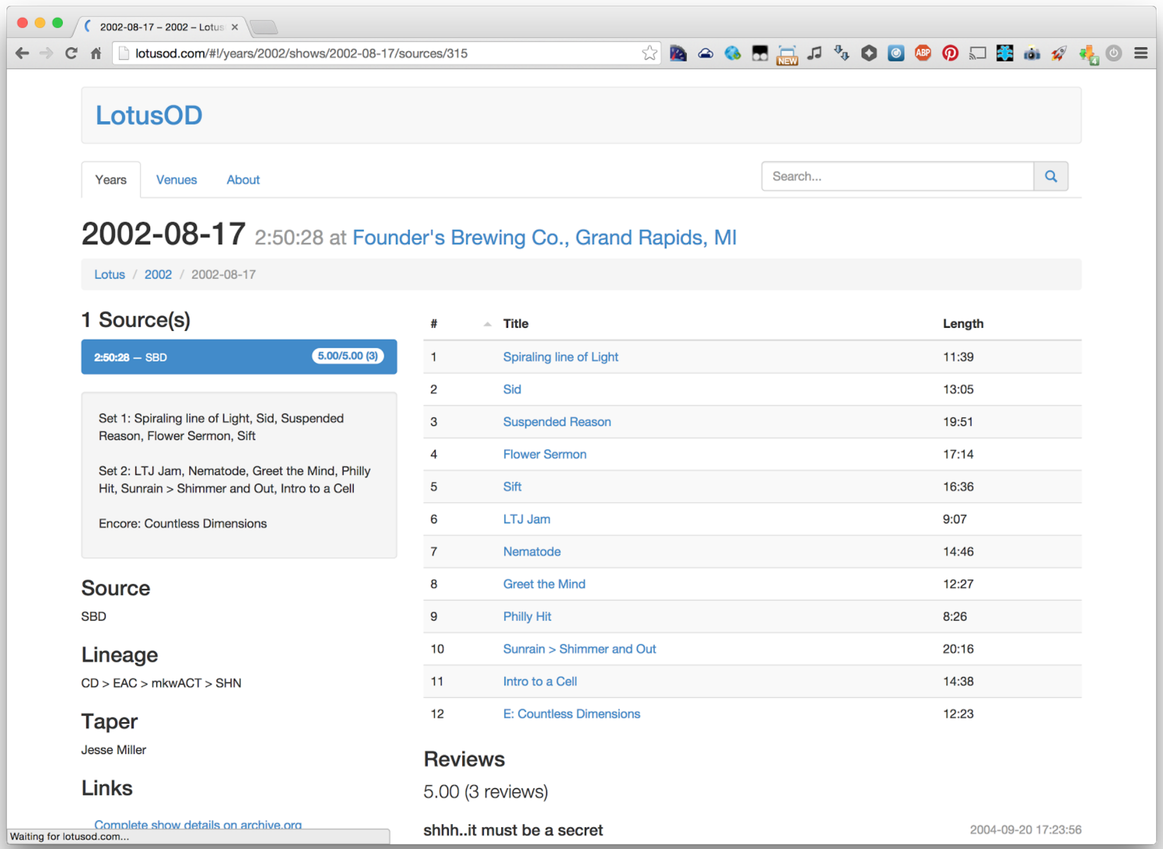
Venue	Barton Hall - Cornell Univ… ›
Source	Matrix mix (SBD/AUD) ›
Duration	2:56:32
Reviews	9 ›
Description	Set 1 ›

Tracks

1	**Minglewood Blues**	6:36
2	**Loser**	8:53
3	**El Paso**	5:22
4	**They Love Each Other**	9:11
5	**Jack Straw**	8:04
6	**Deal**	7:29

The current iOS app

iguana-web



2002-08-17 – 2002 – Lotus

lotusod.com/#!/years/2002/shows/2002-08-17/sources/315

LotusOD

Years | Venues | About

Search...

2002-08-17 2:50:28 at Founder's Brewing Co., Grand Rapids, MI

Lotus / 2002 / 2002-08-17

1 Source(s)

2:50:28 – SBD 5.00/5.00 (3)

Set 1: Spiraling line of Light, Sid, Suspended Reason, Flower Sermon, Sift

Set 2: LTJ Jam, Nematode, Greet the Mind, Philly Hit, Sunrain > Shimmer and Out, Intro to a Cell

Encore: Countless Dimensions

Source

SBD

Lineage

CD > EAC > mkwACT > SHN

Taper

Jesse Miller

Links

Complete show details on archive.org

#	Title	Length
1	Spiraling line of Light	11:39
2	Sid	13:05
3	Suspended Reason	19:51
4	Flower Sermon	17:14
5	Sift	16:36
6	LTJ Jam	9:07
7	Nematode	14:46
8	Greet the Mind	12:27
9	Philly Hit	8:26
10	Sunrain > Shimmer and Out	20:16
11	Intro to a Cell	14:38
12	E: Countless Dimensions	12:23

Reviews

5.00 (3 reviews)

shhh..it must be a secret 2004-09-20 17:23:56

Waiting for lotusod.com...

The current web interface

The new iguana-web interface will have corresponding view elements for each iguana model on the server side: IGTrack, IGShow, IGVenue, IGYear, IGUser, IGFavorite and IGPlaylist.

When users go to the root directory of the web interface the server will respond with a prompt for the user to login (or create an account) for full app access (Favoriting and Playlist support) or to continue without logging in. Logging in will be achieved by sending the user's account credentials over SSL to the iguana api. On the server the credentials will be hashed and checked against the credentials in the database. If valid, the server will respond with an authentication token that will be stored as a cookie, preventing the user from having to login in every time they want to use the application. New users will have a similar process, except their credentials won't be checked against existing credentials in the database, but instead will create a new User in the database all together. Users who choose not to log in will have a similar interface to the current interface seen above.

All of the information (Tracks, Shows, Venues, etc) displayed in the application will be provided by GET requests to the iguana api and the actual data will be bound to the DOM via angularJS. Times that the user needs to change data server side (favoriting a song, creating a playlist, changing user data) the app will POST the changes via the iguana api.

PushState will be used for audio playback, so when a user switches between views the music is uninterrupted.

iguana-chrome

<u>Audio Player</u>
An audio player will be embedded into the Chrome extension so that music playback can be separate from the web player. This will allow for music to be played directly off the extension without the web page being open and makes pausing/playing music easier to handle.

<u>Connection to Database</u>
Connect the Chrome extension to the database using a RESTful API.

- Each time new data is needed, a call will be made to the RESTful API.

<u>Internal State</u>
There will be a state kept inside the Chrome extension in terms of what song is currently playing, etc. This will reduce the amount of overhead on the server and make the API effectively state-less for certain clients. This allows the server to be more light-weight and therefore faster for other clients.

<u>Controls</u>
Basic controls for music playback will include a play/pause button, a skip button, and a previous button.

- The play/pause button will only modify the internal state.
- The skip button will send a request to the database to pull information.
- The previous song ID will be stored so that a request can be made to the database to pull the previous song.