**Requirements (90 points + 4 bonus points)**

1. Authentication method: {**total 12 points + 4 bonus points**}
   1. Local authentication mechanism which uses email as the username and require a password. {**4 points**} 💗
   2. One external authentication mechanism (e.g. Google, Apple, Facebook etc) that authenticates via a third-party such as Google, Facebook etc. {**4 bonus points**}
   3. Input validation for email (proper format). {**2 points**} 💗
   4. Local passwords stored using the most secure method available (e.g. Argon2). {**1 point**} 💗
   5. Verification of of email (see References). {**3 points**}
   6. If the account is marked as deactivated, show a message asking to contact the site administrator and not allow logging in. {**2 points**}
2. **Limited functionality for unauthenticated users. {total 24 points}**
   1. Start page showing application name, a short “about” blurb that says what the site offers, and login button. {**2 points**}💗
   2. List of songs (up to 10) ordered by popularity (e.g. number of users who reviewed that song or average rating). Any sensible criteria of popularity is acceptable. {**4 points**}💗
   3. Ability to search songs based on keywords. {**6 points**}💗
   4. Keywords are matched with all attributes of the item (see ID3v1 below). {**2 points**} 💗
   5. Keywords are soft-matched (e.g ignore differences in case, white-space, minor spelling variations). {**2 points**}
   6. Ability to view all information on a song by clicking or expanding (shows all attributes, most recent review, the number of reviews and the average rating. {**4 points**}💗
   7. Ability to view all reviews for a song: {**2 points**}
   8. Each review shows the rating, the review and reviewer’s username: {**2 points**}
3. **Additional functionality for authenticated users: {total 18 points}**
   1. Add a review to a song. {**4 points**} 💗
   2. Add a rating (1-5, star etc) to reviews created by the user.  {**2 points**} 💗
   3. Add a new song to the site. {**4 points**}💗
   4. Support storing all ID3v1 attributes for each new song. {**4 points**}💗
   5. Enforce required attributes “title” and “artist” when adding a new song. {**2 points**}
   6. Add a review while adding a new song if necessary. {**2 points**}
4. **Site manager functionality related to site maintenance: {total 10 points}**
   1. Special user with site manager access. {**4 points**}💗
   2. Ability to grant site manager privilege to one or more existing users: {**2 points**}
   3. Ability to mark a song as hidden and clear the “hidden” flag if set:  {**2 points**}
   4. Ability to mark a user as “deactivated” and mark as “active” if deactivated: {**2 points**}
5. **Web service API: {total 14 points}**
   1. Provide an API that has at least four nouns (URLs) and at least 8 separate noun+ HTTP verb combinations. Must have at least two nouns that support two or more verbs. {**8 points**} 💗
   2. Build your application using this API. {**6 points**} 💗
6. **Site manager functionality related to copyright enforcement: {total 12 points}**
   1. Create a security and privacy policy that is publicly accessible. {**2 points**} 💗
   2. Create a DMCA notice & takedown policy that is publicly accessible. {**2 points**} 💗
   3. Provide a DMCA takedown procedure and tools for the site administrator: {**total 8 points**}
      1. Document to describe the workflow and usage of tools. {**2 points**} 💗
      2. Tools to log requests, notices, and disputes. E.g. Set-up properties for storing “date request received”, “date notice sent”, “date dispute received” with each song and provide an interface to set these properties. {**2 points**}
      3. Tools to disable display of songs with alleged copyright violations. {**2 points**}
      4. Tools to restore any contested songs. {**2 points**}

Below IS NOT REQUIREMENTS

1. **Usability and code quality: {up to -30 points}**
   1. Resistant to HTML and JavaScript injection attacks, {**up to -2  point**}
   2. Able to handle user input in any language. {**up to -2  point**}
   3. Usability of the application on multiple browsers and form factors. {**up to -2 point**}
   4. Modular code that is easily extensible and maintainable. {**up to -2 point**}
      1. E.g. Changes to operational parameters such as server names, port numbers etc should not cause changes in code.
   5. Avoid code duplication. {**up to -2 point**}
      1. E.g. Full URLs that are duplicated in calls to ReST api
   6. Avoid hard-coded literals in code. {**up to -4 points**}
      1. E.g. Hard-coded port numbers, URLS
   7. Frequent git commits with meaningful commit messages. {**up to -10 points**}
   8. Sufficient and meaningful comments in code. {**up to -2 points**}
   9. Proper precautions in saving user information. {**up to -4 points**}

**API SKETCH**

1. Sketch a preliminary API that provides the basic functionality. Following is an example design.
   1. Designate separate API prefixes for non-authenticated, regular user and admin categories. E.g. All paths in “/api/secure/” require authentication as a regular user. Paths in “/api/admin/” require admin privileges. Paths in “/api/open/” does not require authentication.
   2. PUT /api/secure/song/ - save the JSON array for a song in the database and return the ID.
   3. POST /api/secure/song/:id - update the record of the given song ID with JSON array of properties sent in the body.
   4. GET /api/open/song - return a list of 10 songs ordered by average rating. Optionally, you may pass a query parameter to indicate the number of results to return.
   5. GET /api/open/search - return a list of songs matching the search criteria provided as query parameters.
   6. GET /api/open/reviews/:id - return all reviews for a given song ID.
   7. PUT /api/secure/add-review/:id - Create a new review for the song with the given ID based on JSON array provided in the body.
   8. POST /api/admin/copyright/:id - Set or update copyright violation attributes for a given song ID. JSON array with new values is provided in the body.
   9. GET /api/admin/copyright - Return all songs which are marked as copyright violations.
   10. POST /api/admin/deactivate/:id - Set or clear “account deactivated” flag for a given user.