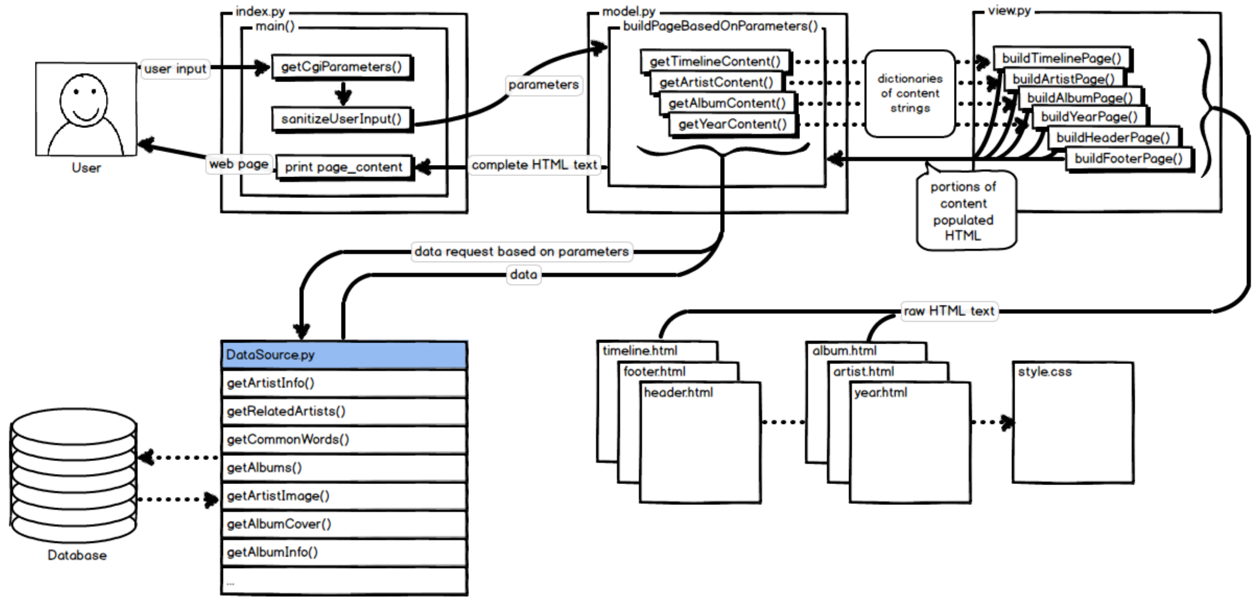
System Architecture



Overview:

The system architecture for our web application is divided into three distinct modules for the model, view, and controller components of the system, as well as a database interface and some auxiliary data (HTML template files and a CSS stylesheet). When the user provides input in the form of CGI parameters, the controller module (index.py) receives and sanitizes the dictionary of parameters, and passes the sanitized input to the model.

The method buildPageBasedOnParameters() of the module model.py takes these parameters as an argument, and determines what content to create based on the parameters. The buildPageBasedOnParameters() method then passes individual parameters to helper methods such as getArtistContent(), which make database queries in order to create a dictionary of content strings containing relevant information for the view module. When all necessary content has been generated by helper methods in the model module, buildPageBasedOnParameters() then passes the content dictionaries as arguments to the appropriate methods in the view module. Each of the methods in the view module constructs a part of the HTML page to be delivered to the user, placing content generated by the model where it belongs in the template HTML text.

The buildPageBasedOnParameters() method then constructs a complete HTML page by combining the strings of HTML text returned by methods of the view module. Once the web page to be displayed is completed, buildPageBasedOnParameters() returns the complete string of HTML text to index.py, which prints the page to the user's screen.

Deliverables

Index.py *[Module] - Controller*
Description: This module contains methods for the controller portion of the system, and is the file which is run from the browser in order to use the web application. This module's functionality is limited to receiving input from the user, and printing what is generated by the web app.
Test Deliverables: We plan to develop a suite of unit tests for the two methods of the module. These will test if the sanitize inputs function is able to correctly remove the correct characters, for example by passing the string N'as. If the input isn't returned as Nas, then we know the test failed.
Functions:

`GetCgiParameters()`
Output: A dictionary of parameters from the webapp (Dictionary<String>)
Description: This method returns the CGI parameters generated from user input.

`sanitizeUserInputs(parameters)`
Argument(s): CGI Parameters (String)
Output: Sanitized user input (String)
Description: This method removes characters from the user input string which might be used to perform SQL injection.

Model.py *[Module] - Model*
Description: This module contains methods for the model portion of the system. The core method of this module is buildPageBasedOnParameters(), which interprets the dictionary of CGI parameters to determine which methods of the view module should be called to construct the web page. The module also has a number of helper methods for the model portion of the system, which take forms of user input and use the database interface in order to generate content for the view module.
Test Deliverables: This module will require a large suite of unit tests for the different methods, since this is the model portion of the system. Testing the buildPageBasedOnParameters() method will require hardcoding print statements to see if the correct helper methods are called based on a set of parameters, since the method returns a large string of HTML text. For the different getContent methods, we will pass in some different arguments expecting the method to return string dictionary that matches our own construction of the dictionary. If they aren't equal, then the test fails.
Functions:

`buildPageBasedOnParameters(parameters)`
Argument(s): The sanitized input dictionary from the controller (Dictionary<String>)

Output: Complete HTML text for a web page (String)
Description: This method takes a dictionary of strings as an argument, and returns a string containing the complete HTML text for a web page. The method does this by determining which view methods should be called to generate the web page, based on which parameters exist. This method also makes calls to helper methods in the module, passing individual parameters as arguments in order to get the string content which is used in the HTML text. The string content is then used as arguments for the view methods which generate the web page.

`getAlbumContent(albumID)`
Argument(s): Album ID (Integer)
Output: Dictionary of content title keys and content string values (Dictionary<String>)
Exceptions: IO exception if anything but an int is passed
Description: This method takes an album identifier as an argument, and creates an instance of the album class of the database interface. The method then uses methods of the album object to build a dictionary of strings containing content to be used in creating the album page.

`getArtistContent(artistID)`
Argument(s): Artist ID (Integer)
Output: Dictionary of content title keys and content string values (Dictionary<String>)
Exceptions: IO exception is anything but an int is passed
Description: This method takes an artist identifier as an argument, and creates an instance of the artist class of the database interface. The method then uses methods of the album object to build a dictionary of strings containing content to be used in creating the album page.

`getTimelineContent()`
Output: Dictionary of content title keys and content string values (Dictionary<String>)
Description: This method generates a content dictionary for the entire timeline by making successive calls of the method getYearContent(), getting content for each of the years.

`getYearContent(year)`
Argument(s): Year (Integer)
Output: Dictionary of content title keys and content string values (Dictionary<String>)
Exceptions: IO expection if the year isn't passed an integer
Description: This method takes an integer year as an argument, and creates an instance of the timeline class of the database interface. The method then uses methods of the timeline

object to build a dictionary of strings containing content to be used in creating the year page.

View.py *[Module] - View*
Description: This module contains methods for the view portion of the system. Each method is used to build a specific portion of the web page for our web application. These methods take dictionaries of content strings as arguments, and place the content into HTML template text in order to build portions of a content populated web page.
Test Deliverables: The test deliverables for this class will pass a content dictionary for test artists and albums, and then see if the content is returned correctly, like the artist name, and the album description.
Methods:

`buildAlbumPage(contentDictionary)`
Argument(s): Dictionary of content title keys and content string values (Dictionary<String>)
Output: Raw HTML text (String)
Description: This method takes a dictionary of content strings as an argument, and uses it to populate a template HTML file. The strings for album name, description, etc. are then replaced in the template file album.html, and the HTML text is returned.

`buildArtistPage(contentDictionary)`
Argument(s): Dictionary of content title keys and content string values (Dictionary<String>)
Output: Raw HTML text (String)
Description: This method takes a dictionary of content strings as an argument, and uses it to populate a template HTML file. The strings for artist name, description, etc. are then replaced in the template file artist.html, and the HTML text is returned.

`buildHeaderPage()`
Output: Raw HTML text (String)
Description: Returns a string of HTML text from the template file header.html.

`buildFooterPage()`
Output: Raw HTML text (String)
Description: Returns a string of HTML text from the template file footer.html.

`buildTimelinePage(contentDictionary)`
Argument(s): Dictionary of content title keys and content string values (Dictionary<String>)

Output: Raw HTML text (String)
Description: This method takes a dictionary of content strings as an argument, and uses it to populate a template HTML file. The strings for album name and artist for each year are then replaced in the template file timeline.html, and the HTML text is returned.

`buildYearPage(contentDictionary)`
Argument(s): Dictionary of content title keys and content string values (Dictionary<String>)
Output: Raw HTML text (String)
Exceptions: IO exception if the dictionary value for year isn't passed as an integer
Description: This method takes a dictionary of content strings as an argument, and uses it to populate a template HTML file. The strings for album name and artist for the year are then replaced in the template file year.html, and the HTML text is returned.

DataSource.py *[Module] - Model*
Description: This module will be used to perform database queries. Each class within the interface represents the type of data that we are seeking to store in the content dictionary that is eventually passed to the view module.
Methods:

`getYearsOnTimeline()`
Output: List of integer values for years which have information that can be displayed on the timeline. (List<Integer>)
Description: This method creates a list of year integers based on the year values for each album in our database.

Classes:
Album *[Class] - Model*
Constructor: `__init__(albumID)`
Argument(s): Album ID (Integer)
Exceptions: IO exception if the albumID isn't an integer
Description: This class represents an album object, with instance variables for the name, artist, description and release year based on database entries. Methods of the class are then used to get the different instance variables.
Test Deliverables: For this class, we will construct an object of the album class with the albumID for Illmatic. Then we will try to get the album name, year, artist and description, and confirm that this information corresponds to the correct information for Illmatic. If the information doesn't match what we expect, we will throw an exception, and rewrite the code for the class.
Methods:

`getAlbumName()`
Output: Album title (String)

Description: This method returns the title for an album object.

`getAlbumYear()`
Output: Album release year (Integer)
Description: This method returns the album release year for an album object as an integer.

`getAlbumArtist()`
Output: Album artist (String)
Description: This method returns the album artist for an album object.

`getAlbumDescription()`
Output: Album description (String)
Description: This method returns the album description for an album object.

Artist *[Class] - Model*
Constructor: `__init__(artistID)`
Argument(s): Artist identifier (Integer)
Exceptions: IO exception if the artistID isn't passed as an int
Description: This class represents an artist object, with instance variables for the name, major albums, description, vocabulary and common words based on database entries. Methods of the class are then used to get the different instance variables.
Test Deliverables: We will construct an object of the artist class with the artistID for Nas. The content of the getMajorAlbums and getArtistDescription should match the customized variables we will build. If they do not, we will throw an exception and rewrite the code for the class.
Methods:

`getMajorAlbums()`
Output: A list of album identifiers (List<Integer>)
Description: This method returns the artist's release year for an artist object as a list of album identifiers.

`getArtistDescription()`
Output: Artist description (String)
Description: This method returns the description for an artist object as a string.

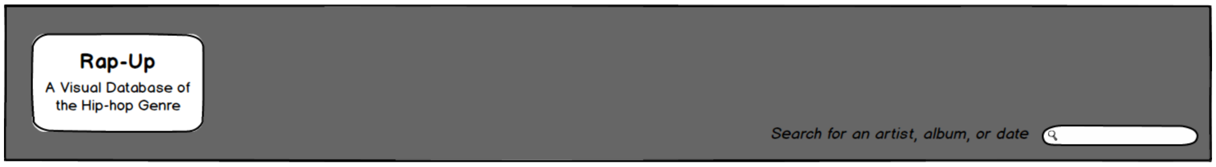
`getArtistVocabularyNum()`
Output: Number of unique words used by artist (Integer)
Description: Returns the artist vocabulary, an integer value representing the number of unique words used by the artist.

`getCommonWords()`
Output: List of most commonly used words (List<String>)
Description: Returns a list of the top 20 common words of the artist.

Timeline *[Class] - Model*
Constructor: `__init__(year)`
Argument(s): The year as an integer
Output: None
Exception: IO exception if the year isn't passed an int
Description: This class represents a timeline object, with an instance variable that is a list of the major albums released that year, based on database entries. The class has one method to access its instance variables.
Test Deliverables: For this class, we will construct a list of albums we expect to see for the year 1994. If the getAlbumsForYear doesn't match this content, we will throw an exception, and will have to change the construction of the class.
Methods:

`getAlbumsForYear()`
Output: List of albums released for this year (List<Integer>)
Description: Returns a list of album identifiers for that year.

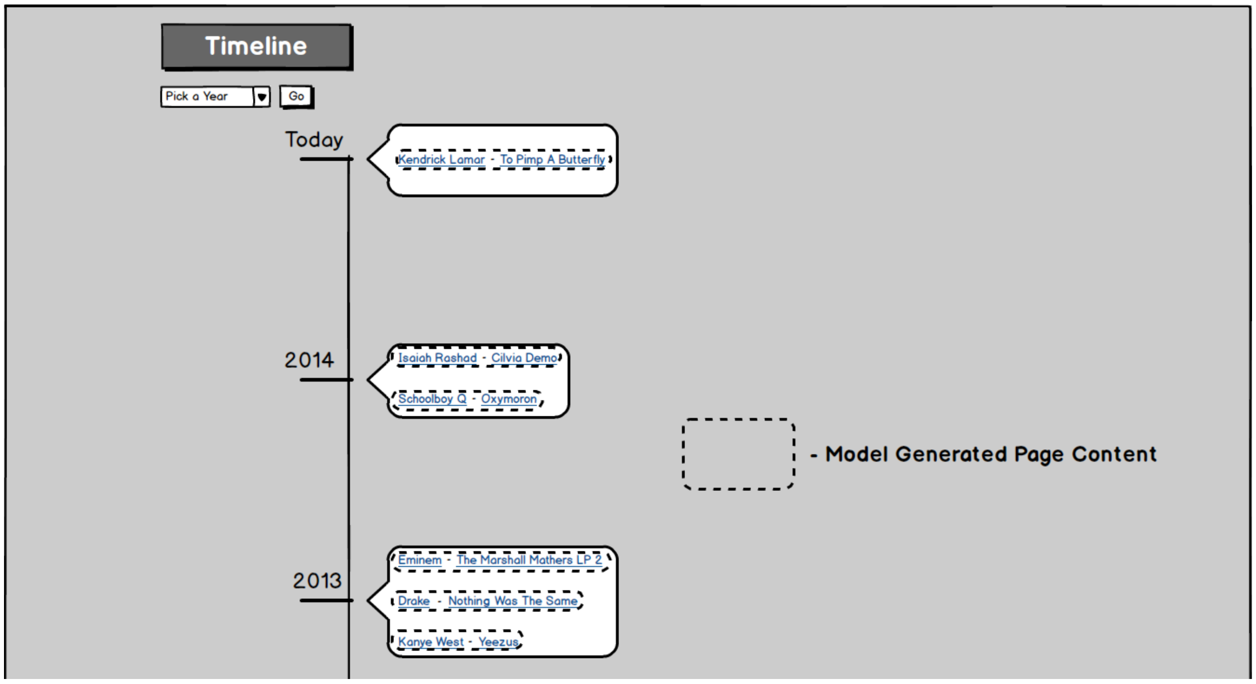
Auxiliary Data:
HTML Templates
header.html - HTML template for the header portion of web application. The header template is a static document which is independent of user input and is included on all pages displayed by the web app.
Preview:


Rap-Up
A Visual Database of the Hip-hop Genre
Search for an artist, album, or date

footer.html - HTML template for the footer portion of web application. The footer template is a static document which is independent of user input and is included on all pages displayed by the web app.

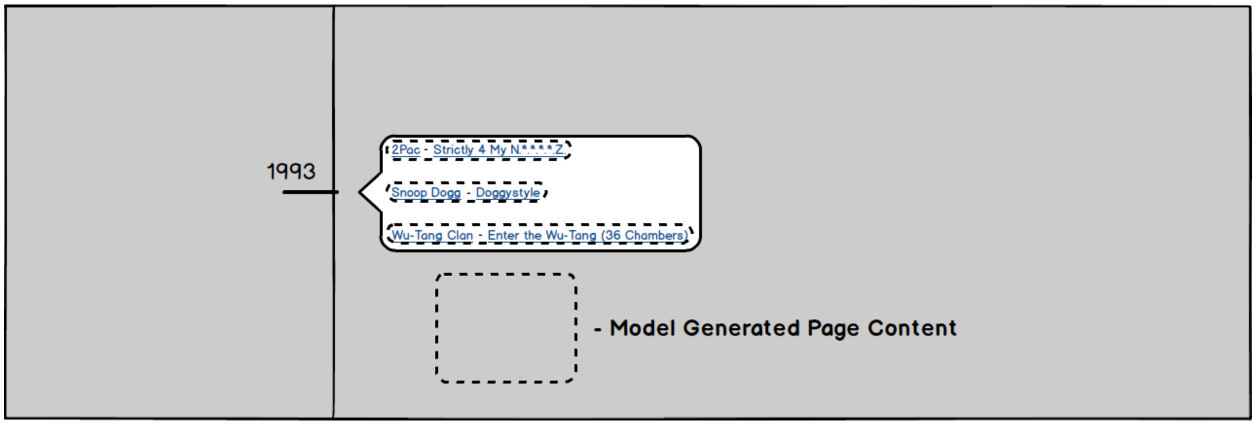
timeline.html - HTML template for the timeline portion of web application. We are planning to include the timeline on the primary home page for the web application, and will include content for each year.

Preview:


Timeline
Pick a Year
Go
Today
Kendrick Lamar - To Pimp A Butterfly
2014
Isaiah Rashad - Cilvia Demo
Schoolboy Q - Oxymoron
- Model Generated Page Content
2013
Eminem - The Marshall Mathers LP 2
Drake - Nothing Was The Same
Kanye West - Yeezus

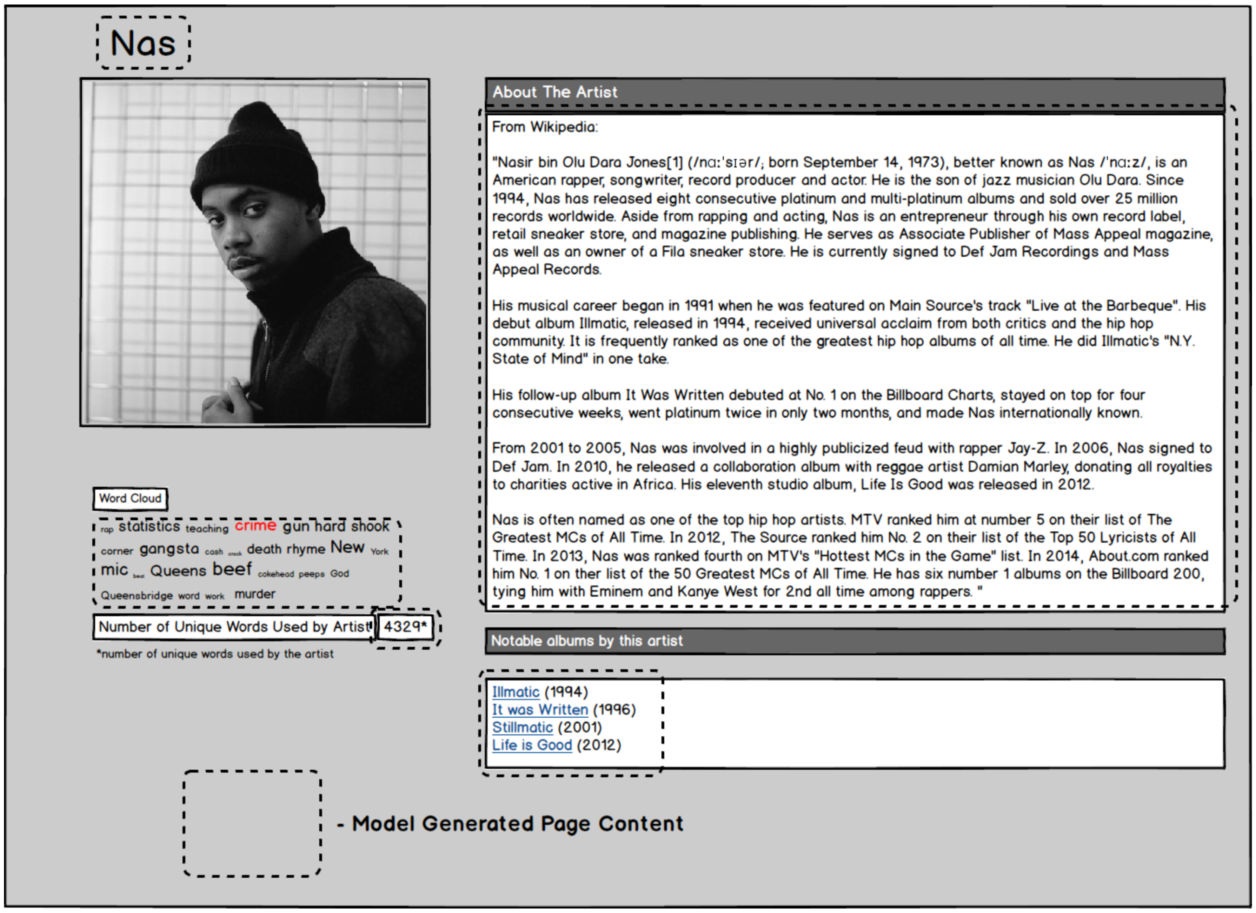
year.html - HTML template for the individual years on the timeline portion of the web application. When a user selects a year to view on the timeline, year.html is populated with content based on their input.

Preview:


1993
2Pac - Strictly 4 My N.*.*.*.Z.
Snoop Dogg - Doggystyle
Wu-Tang Clan - Enter the Wu-Tang (36 Chambers)
- Model Generated Page Content

artist.html - HTML template for the artist page of web application. When a user selects an artist to view from the timeline, artist.html is populated with content based on their input.

Preview:


Nas
Word Cloud
rap statistics teaching crime gun hard shook corner gangsta cash crack death rhyme New York mic beat Queens beef cokehead peeps God Queensbridge word work murder
Number of Unique Words Used by Artist
4329*
*number of unique words used by the artist
About The Artist
Notable albums by this artist
- Model Generated Page Content

From Wikipedia:

"Nasir bin Olu Dara Jones[1] (/nɑːˈsɪər/; born September 14, 1973), better known as Nas /ˈnɑːz/, is an American rapper, songwriter, record producer and actor. He is the son of jazz musician Olu Dara. Since 1994, Nas has released eight consecutive platinum and multi-platinum albums and sold over 25 million records worldwide. Aside from rapping and acting, Nas is an entrepreneur through his own record label, retail sneaker store, and magazine publishing. He serves as Associate Publisher of Mass Appeal magazine, as well as an owner of a Fila sneaker store. He is currently signed to Def Jam Recordings and Mass Appeal Records.

His musical career began in 1991 when he was featured on Main Source's track "Live at the Barbeque". His debut album Illmatic, released in 1994, received universal acclaim from both critics and the hip hop community. It is frequently ranked as one of the greatest hip hop albums of all time. He did Illmatic's "N.Y. State of Mind" in one take.

His follow-up album It Was Written debuted at No. 1 on the Billboard Charts, stayed on top for four consecutive weeks, went platinum twice in only two months, and made Nas internationally known.

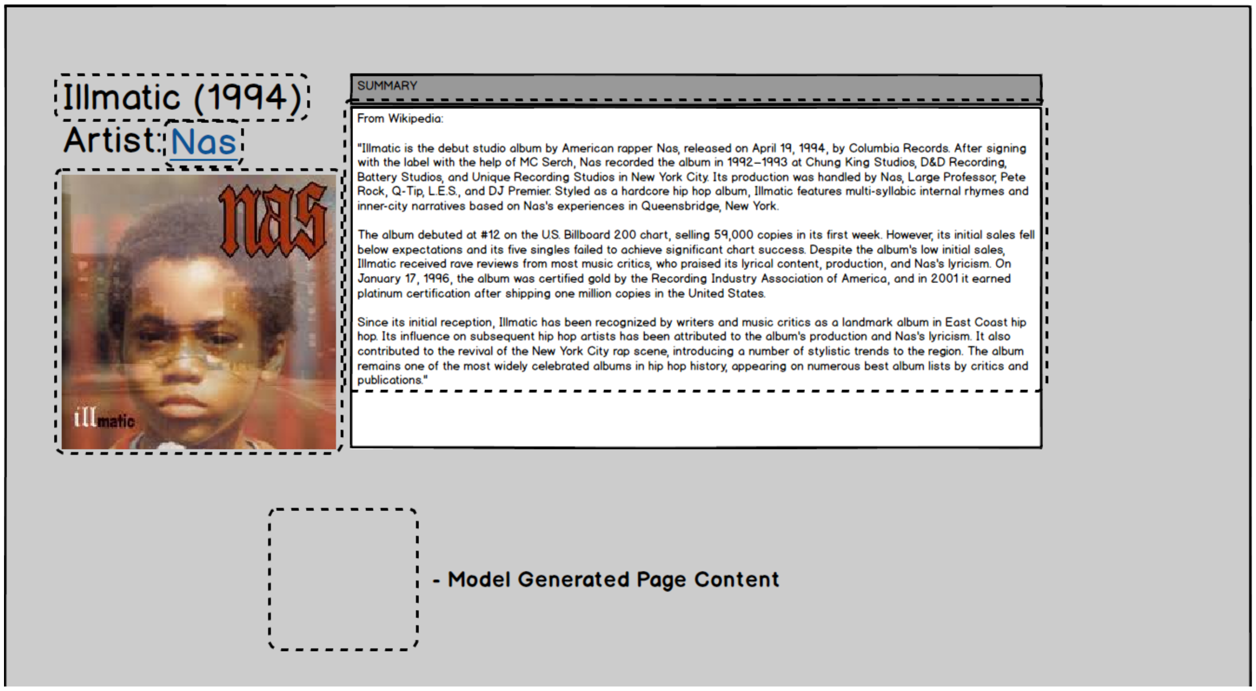
From 2001 to 2005, Nas was involved in a highly publicized feud with rapper Jay-Z. In 2006, Nas signed to Def Jam. In 2010, he released a collaboration album with reggae artist Damian Marley, donating all royalties to charities active in Africa. His eleventh studio album, Life Is Good was released in 2012.

Nas is often named as one of the top hip hop artists. MTV ranked him at number 5 on their list of The Greatest MCs of All Time. In 2012, The Source ranked him No. 2 on their list of the Top 50 Lyricists of All Time. In 2013, Nas was ranked fourth on MTV's "Hottest MCs in the Game" list. In 2014, About.com ranked him No. 1 on their list of the 50 Greatest MCs of All Time. He has six number 1 albums on the Billboard 200, tying him with Eminem and Kanye West for 2nd all time among rappers. "

Illmatic (1994)
It was Written (1996)
Stillmatic (2001)
Life is Good (2012)

album.html - HTML template for the album page of web application. When a user selects an album to view from the timeline, album.html is populated with content based on their input.

Preview:


Illmatic (1994)

Artist: Nas

SUMMARY

From Wikipedia:

"Illmatic is the debut studio album by American rapper Nas, released on April 19, 1994, by Columbia Records. After signing with the label with the help of MC Serch, Nas recorded the album in 1992–1993 at Chung King Studios, D&D Recording, Battery Studios, and Unique Recording Studios in New York City. Its production was handled by Nas, Large Professor, Pete Rock, Q-Tip, L.E.S., and DJ Premier. Styled as a hardcore hip hop album, Illmatic features multi-syllabic internal rhymes and inner-city narratives based on Nas's experiences in Queensbridge, New York.

The album debuted at #12 on the U.S. Billboard 200 chart, selling 59,000 copies in its first week. However, its initial sales fell below expectations and its five singles failed to achieve significant chart success. Despite the album's low initial sales, Illmatic received rave reviews from most music critics, who praised its lyrical content, production, and Nas's lyricism. On January 17, 1996, the album was certified gold by the Recording Industry Association of America, and in 2001 it earned platinum certification after shipping one million copies in the United States.

Since its initial reception, Illmatic has been recognized by writers and music critics as a landmark album in East Coast hip hop. Its influence on subsequent hip hop artists has been attributed to the album's production and Nas's lyricism. It also contributed to the revival of the New York City rap scene, introducing a number of stylistic trends to the region. The album remains one of the most widely celebrated albums in hip hop history, appearing on numerous best album lists by critics and publications."

CSS Sheet(s)

+ style.css - Cascading stylesheet which will be referenced by HTML template files to improve aesthetic quality of web application output.

Development Schedule

May 5 (Tuesday - Week 6)

- The test deliverables for view.py, model.py, datasource.py and index.py are complete.
- Methods within the module index.py pass unit tests, and make calls to methods of other modules.

May 7 (Thursday - Week 6)

- Database is populated with artist and album data for the timeline, and a subset of information from corresponding Wikipedia pages.

May 8 (Friday - Week 6)

- DataSource.py is implemented.
- Core methods of DataSource.py pass unit tests.

May 9 (Saturday - Week 6)

- The view.py module is complete, and can produce some HTML content based on the methods of DataSource.py.
- Some methods of view.py will pass unit tests.

May 10 (Sunday - Week 6)

- The module model.py are mostly complete, but not all methods will pass their unit tests. Calls to DataSource.py and view.py are be implemented.

May 11 (Monday - Week 7)

- The module model.py is complete, and passes the test deliverable.
- Version 1 of web application is complete, implementing basic features from proposal with a reasonable front-end. All modules should pass their corresponding test deliverables.

May 13 (Wednesday - Week 7)

- Improvements made to the style.css sheet, improving visual aesthetic.

May 18 (Monday - Week 8)

- Final version of web application complete.