Datalytics
by Byte Peeps

System Design document

1st July 2022

Collective work by Mohamed Tayeh (Moh), Bassel Ashi, Connie Lin, Karandeep Lubana (Giani), Jiaming Yang (Gloria), and Juan Camilo Corral

Table of Contents

CRC Cards

Database Model Classes

User		
Responsibilities • Database model for application users		**Fields** • username: string • password: string
Parent Class Model	**Sub Class** None	**Collaborations** • FacebookApi • InstagramApi • YouTubeChannel

FacebookApi		
Responsibilities • Database model for Facebook APIs		**Fields** • token: string • isActive: boolean • userId: number
Parent Class Model	**Sub Class** None	**Collaborations** • FacebookPost

FacebookPost		
Responsibilities • Database model for Facebook posts		**Fields** • message: string • date: Date • likes: number • loves: number • cares: number • hahas: number • wows: number • sads: number • angrys: number • dataId: string
Parent Class Model	**Sub Class** None	**Collaborations** • FacebookApi • FacebookMedia

FacebookComment		
Responsibilities ● Database model for Facebook comments		**Fields** ● dataId: string ● username: string ● message: string ● date: Date ● likes: number ● sentimentAnalysis: string ● topicClassification: string ● subjectivityAnalysis: string ● postId: number
Parent Class Model	**Sub Class** None	**Collaborations** ● FacebookPost

InstagramApi		
Responsibilities ● Database model for Instagram API		**Fields** ● facebookApiId: number ● userId: number ● nodeId: string
Parent Class Model	**Sub Class** None	**Collaborations** ● FacebookApi ● InstagramMedia

InstagramMedia		
Responsibilities ● Database model for Instagram media		**Fields** ● dataId: string ● caption: string ● date: Date ● likes: number ● numComments: number ● apiId: number
Parent Class Model	**Sub Class** None	**Collaborations** ● InstagramApi ● InstagramComment

InstagramComment		
Responsibilities ● Database model for Instagram comments		**Fields** ● dataId: string ● userName: string ● message: string ● date: Date ● likes: number ● sentimentAnalysis: string ● topicClassification: string ● subjectivityAnalysis: string ● mediaId: number
Parent Class Model	**Sub Class** None	**Collaborations** ● InstagramMedia

InstagramTag		
Responsibilities ● Database model for Instagram tags, which are basically posts that the user is tagged in		**Fields** ● dataId: string ● username: string ● caption: string ● date: Date ● likes: number ● numComments: number ● sentimentAnalysis: string ● topicClassification: string ● subjectivityAnalysis: string ● apiId: number
Parent Class Model	**Sub Class** None	**Collaborations** ● InstagramApi

YouTubeChannel		
Responsibilities ● Database model for YouTube channels		**Fields** ● resourceId: string ● name: string ● userId: string
Parent Class Model	**Sub Class** None	**Collaborations** ● YouTubeVideo

YouTubeVideo		
Responsibilities ● Database model for YouTube videos		**Fields** ● resourceId: string ● date: number ● views: number ● likes: number ● channelId: number
Parent Class Model	**Sub Class** None	**Collaborations** ● YouTubeChannel ● YouTubeComment

YouTubeComment		
Responsibilities ● Database model for YouTube videos		**Fields** ● resourceId: string ● userName: stringr ● message: number ● likes: number ● sentimentAnalysis: string ● topicClassification: string ● subjectivityAnalysis: string ● videoId: number
Parent Class Model	**Sub Class** None	**Collaborations** ● YoutubeVideo

Backend CRC Classes

Controller/Instagram/Comment		
Responsibilities ● Has the API endpoint implementations for instagram comments		**Endpoints** ● getCommentById() ● getCommentsByMediaId() ● getCommentsSentimentAnalysis() ● getCommentsSubjectivityAnalysis()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/Instagram/Comment ● DatumBox ● InstagramComment

Controller/Instagram/media		
Responsibilities ● Has the API endpoints for posts made by the user company		**Fields** ● getMediaById() ● getMediaByUserId()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/Instagram/Comment ● InstagramMedia

Controller/Instagram/tag		
Responsibilities ● Has the API endpoint implementations for instagram tags		**Fields** ● getTagById() ● getTagsByMediaId() ● getTagsSentimentAnalysis() ● getTagsSubjectivityAnalysis()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/Instagram/Tag ● DatumBox ● InstagramTag

Controller/Youtube/Comment		
Responsibilities ● Has the API endpoint implementations for youtube comments		**Endpoints** ● getCommentById() ● getAllComments()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/YouTube/Comment ● YoutubeComment

Controller/Youtube/channel		
Responsibilities ● Returns the youtube channels		**Fields** ● getChannelById() ● getAllChannels() ● createChannel() ● deleteChannel()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/YouTube/Comment ● YoutubeChannel

Controller/Youtube/video		
Responsibilities ● Returns the		**Fields** ● getVideoById() ● getAllVideos() ● deleteVideo() ● createVideo()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/Youtube/Video ● YoutubeVideo

Controller/Setup/facebook		
Responsibilities ● Has the API endpoint implementations for setting up Facebook API		**Fields** ● getPages() ● connectPage() ● getCurrentPage()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/Setup/facebook ● User ● FacebookApi ● FacebookPost ● FacebookComment

Controller/Setup/instagram		
Responsibilities ● Has the API endpoint implementations for setting up Instagram API		**Fields** ● getSettings() ● connectPage()
Parent Class Model	**Sub Class** None	**Collaborations** ● Routes/Setup/instagram ● User ● FacebookApi ● InstagramApi ● InstagramMedia ● InstagramComment

Routes/Instagram/Comment		
Responsibilities ● Maps API endpoints to their routes		**Endpoints** ● getCommentById() ● getCommentsByMediaId() ● getCommentsSentimentAnalysis() ● getCommentsSubjectivityAnalysis()
Parent Class Model	**Sub Class** None	**Collaborations** ● Controller/Instagram/Comment

Routes/Instagram/media		
Responsibilities ● Maps API endpoints to their routes		**Fields** ● getMediaById() ● getMediaByUserId()
Parent Class Model	**Sub Class** None	**Collaborations** ● Controller/Instagram/media

Routes/Instagram/tag		
Responsibilities ● Maps API endpoints to their routes		**Fields** ● getTagById() ● getTagsByMediaId() ● getTagsSentimentAnalysis() ● getTagsSubjectivityAnalysis()
Parent Class Model	**Sub Class** None	**Collaborations** ● Controller/Instagram/tag

Routes/Youtube/Comment		
Responsibilities ● Maps API endpoints to their routes		**Endpoints** ● getCommentById() ● getAllComments()
Parent Class Model	**Sub Class** None	**Collaborations** ● Controller/Youtube/Comment

Routes/Youtube/channel		
Responsibilities • Maps API endpoints to their routes		**Fields** • getChannelById() • getAllChannels() • createChannel() • deleteChannel()
Parent Class Model	**Sub Class** None	**Collaborations** • Controller/Youtube/channel

Routes/Youtube/video		
Responsibilities • Maps API endpoints to their routes		**Fields** • getVideoById() • getAllVideos() • createVideo() • deleteVideo()
Parent Class Model	**Sub Class** None	**Collaborations** • Controller/Youtube/video

Routes/Setup/facebook		
Responsibilities • Maps API endpoints to their routes		**Fields** • getPages() • connectPage() • getCurrentPage()
Parent Class Model	**Sub Class** None	**Collaborations** • Controller/Setup/facebook

Routes/Setup/instagram		
Responsibilities • Maps API endpoints to their routes		**Fields** • getSettings() • connectPage()
Parent Class Model	**Sub Class** None	**Collaborations** • Controller/Setup/instagram

DatumBox		
Responsibilities ● Creates a wrapper for calling the datumbox external machine learning api		**Endpoints** ● DatumBoxAPICall
Parent Class Model	**Sub Class** None	**Collaborations** None

Keyword Extraction		
Responsibilities ● Creates a wrapper to extract the latest keywords from a string text		**Endpoints** ● KeywordExtraction
Parent Class Model	**Sub Class** None	**Collaborations** None

Frontend Auth Classes

<table>
<tr><th colspan="3">login_screen</th></tr>
<tr><td colspan="2">Responsibilities
Allow users to login using a username and a password
Users can choose to register an account if they don’t have one, and they will be redirected to a sign up page
Users will be redirected to the home page after successful login</td><td>Fields
N/A</td></tr>
<tr><td>Parent Class
None</td><td>Sub Class
None</td><td>Collaborations
signup_screen
home_screen</td></tr>
</table>

<table>
<tr><th colspan="3">signup_screen</th></tr>
<tr><td colspan="2">Responsibilities
Allow users to sign up an account
After sign up, the user can choose to go back to the login page</td><td>Fields
N/A</td></tr>
<tr><td>Parent Class
None</td><td>Sub Class
None</td><td>Collaborations
login_screen</td></tr>
</table>

Navigation Classes

App		
Responsibilities ● Run the frontend app and initialize users to the login page		**Fields** ● N/A
Parent Class None	**Sub Class** None	**Collaborations** ● Login_screen ● Header

Header		
Responsibilities ● Build the navigation menu that shows the four main pages on the website		**Fields** ● N/A
Parent Class None	**Sub Class** None	**Collaborations** ● Dashboard ● Surveys ● ReviewApps ● SocialMedia

Dashboard		
Responsibilities ● Provide a summary of reported statistics		**Fields** ● N/A
Parent Class None	**Sub Class** None	**Collaborations** None

Surveys		
Responsibilities ● Lets users create surveys to collect data from participants		**Fields** ● N/A
Parent Class None	**Sub Class** None	**Collaborations** None

ReviewApps		
Responsibilities ● Provides statistics gathered from review apps		**Fields** ● N/A
Parent Class None	**Sub Class** None	**Collaborations** None

SocialMedia		
Responsibilities ● Provides statistics gathered from social media		**Fields** ● N/A
Parent Class None	**Sub Class** None	**Collaborations** None

Chart Classes

TimeSeries Chart		
Responsibilities ● Show sentiment analysis of posts over a time range		**Fields** ● N/A
Parent Class Dashboard	**Sub Class** None	**Collaborations** ● Routes/Instagram/media

<table>
<tr><th colspan="3">Pie Chart</th></tr>
<tr><td colspan="2">Responsibilities
Show sentiment analysis of a post</td><td>Fields
N/A</td></tr>
<tr><td>Parent Class
Dashboard</td><td>Sub Class
None</td><td>Collaborations
Routes/Instagram/comments
Routes/Facebook/comments</td></tr>
</table>

System Interaction

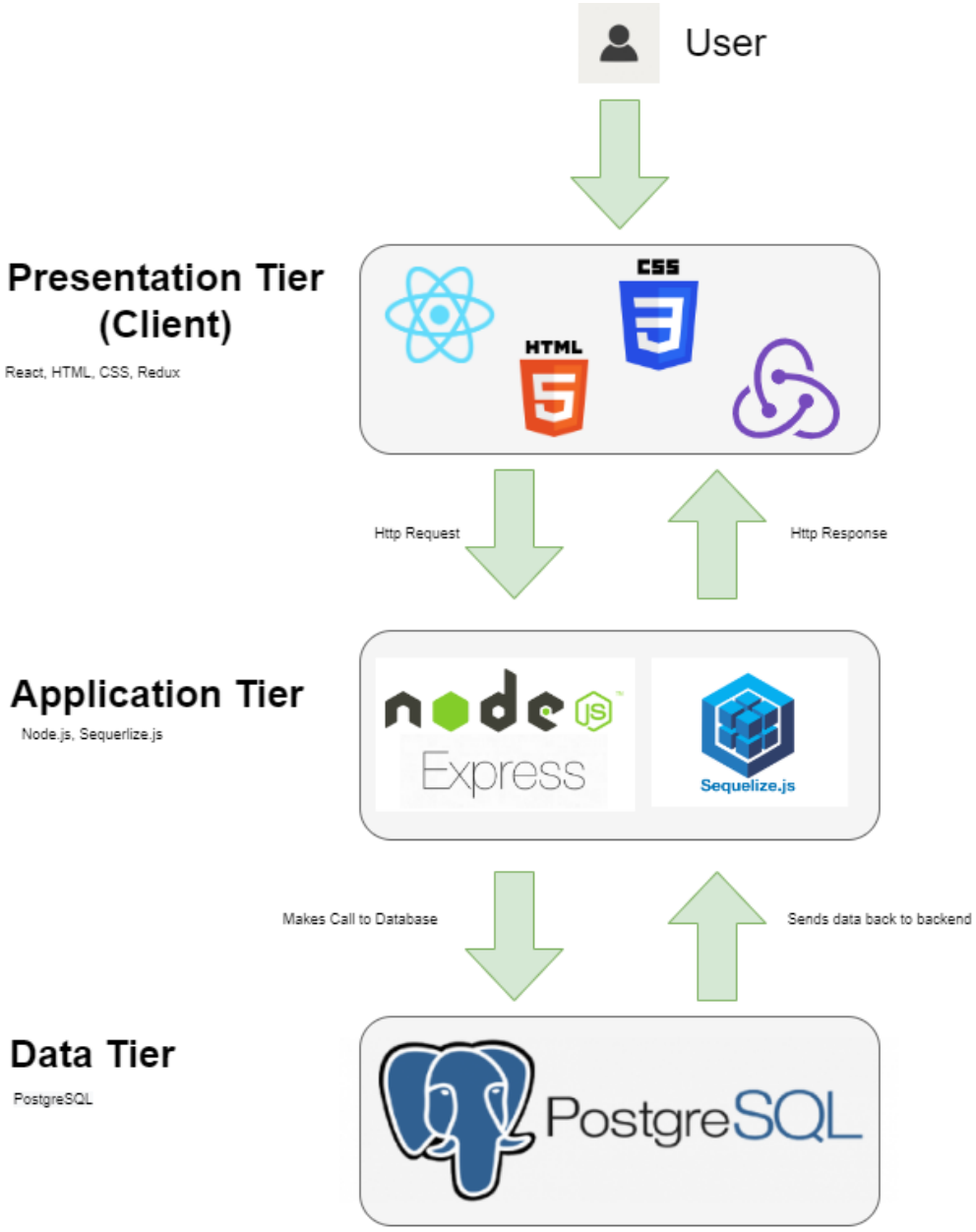
Client:
The frontend is designed to run on Chrome, a chromium engine based website, and Firefox.

Local Development:
There are no assumptions based on the operating system, the README.md instructions are provided for Windows and Mac/Linux. There are no dependencies on the DB's and network configuration.

Deployed System:
We are using docker containers to deploy each component of our application, hence inherently there are no dependencies on the environment of the virtual machine. The network will be configured in the future with CORS so that it is only allowed to communicate with the frontend that is hosted on our domain.

System Architecture

The application utilizes the **Three-Tier Architecture**, consisting of three layers, the presentation layer, application layer and data layer. Our presentation tier uses the React framework for building graphic user interfaces. The presentation tier also consists of tools such as Redux that allow us to write clean code. The presentation tier uses the http transfer protocol to communicate to the application tier which consists of a Node/Express server. The Express server utilizes the sequilize.js library to connect to the data tier, which consists of a postgreSQL database.



Reference for this article: https://www.ibm.com/cloud/learn/three-tier-architecture

System Decomposition

Client Decomposition:

The client is using a web browser to access the website. This means that the client's browser makes a GET request to the server and fetches the frontend. The client interacts with the frontend to make API requests with the backend servers using predefined URLs. These requests are routed in EXPRESS (running on NodeJS) to the correct API endpoint and if the user request is legal, it is allowed. These endpoints use sequelize to communicate with our Postgres database.

Local Development System Decomposition:

Assuming the developer has followed the README.md and completed the installation correctly, they should be able to simply start a frontend server that tracks their continuous changes in the code. The developer should also have a simultaneously running backend server that also tracks their continuous code changes while developing. The local frontend communicates directly with the backend through the local URL. Furthermore, the database is also running the local machine using the built-in Postgres server program. The backend communicates with the database server locally using a URL as well. In case of any errors, these will be

Deployed System Decomposition:

The deployed system works based on docker containers running on the same virtual machine rented from digital ocean. Each of the frontend, backend and database have their own URL and associated docker container. The communication between them is done using TLS to ensure the security and integrity of communication. All servers/docker containers are running at the same time. In addition, there is another container running NGINX to act as a reverse proxy to the requests that are coming into the server and handle the TLS certificate.