3.1.1. Authentication (Angelos). A secure authentication system shall be provided for users to access GameEye. The following requirements shall be met:

3.1.1.1. Account Creation. The system shall allow new users to create an account in two ways: using an email and password, or using their Google or Microsoft account. The following functional requirements shall be met:

1. Using an email and password. The following are required:

a. Valid email address

b. Password with at least 8 characters

2. Using an existing third-party account. The following external providers shall be supported:

a. Google

b. Microsoft

3.1.1.2. Login. The system shall allow existing users to login. Login shall be accomplished in one of two ways:

1. Using an email and password.

2. Using an existing third-party account. The following external providers shall be supported:

a. Google

b. Microsoft

3.1.1.3. Email Verification. Email addresses shall be tested to ensure their validity. The following functional requirements shall be met:

1. Send verification email after user registration.

2. Provide the ability to resend a verification email.

3. Provide the ability to provide another valid email address.

3.1.1.4. Password Reset. The system shall provide a way for users to reset their password.

1. Users shall be sent a password reset email.

3.1.1.5. Persistent Sessions. The system shall ensure that users do not have to sign in again indefinitely. The following functional requirements shall be met:

1. Users shall have to reauthenticate if they explicitly sign out.

2. Users shall have to reauthenticate if they changed their email or password.

3.1.2. Account Management (Angelos). Standard account management features shall be provided by GameEye so that users have control over their account information. The following requirements shall be met:

3.1.2.1. Change Name. The system shall provide a way for users to change their names or provide one since GameEye does not require that users provide a name on account creation.

1. Users shall be able to set a name, even if they have not previously specified one.

3.1.2.2. Change Email. The system shall provide a way for users to change their email. The following requirement shall be met:

1. The system shall ensure that users must verify any new email address they use.

2. Users shall not be able to use GameEye until they verify their new email address.

3. Users shall be able to change their email address while they have an unverified email address.

3.1.2.3. Change Password. The system shall allow users to change their password once they are signed in.

1. Users shall be sent a password reset email.

3.1.2.4. Reauthentication. The system shall require users to reauthenticate themselves before changing important account information such as their email or password.

3.1.3. Watchlist Creation. The system shall allow users to add games to their watchlist by providing partial or complete game titles. Autocompletion suggestions shall be generated based on search strings entered by users. The following requirements shall be met:

3.1.3.1. Autocompletion Suggestions. Searching shall support the autocompletion of partial game titles provided by users. Game title suggestions shall be generated using ElasticSearch. The following requirements shall be met:

1. An edge n-gram index shall be created in ElasticSearch to allow for partial search matches.

2. ElasticSearch queries shall use maximum fuzziness.

3. The system shall provide relevant autocompletion suggestions even if a user makes spelling errors or misses some characters.

4. The system shall provide autocompletion suggestions in real-time, as the user is typing in the search box.

3.1.4. Article Impact Scoring (Angelos, everyone for labeling). The system shall compute an impact score for every news article title using machine learning. The following requirements shall be met:

3.1.4.1. Training Dataset. The training dataset shall consist of 11,000 article titles. The following requirements shall be met:

1. Every title shall be labeled with an impact score from 1 to 3. (req. 3.1)

2. Titles shall be game-specific.

3. Shall not contain any titles present in the training dataset.

3.1.4.2. Test Dataset. The evaluation dataset shall consist of 1,000 article titles. The following requirements shall be met:

1. Every title shall be labeled with an impact score from 1 to 3. (req. 3.1)

2. Titles shall be game-specific.

3. Shall not contain any titles present in the training dataset.

3.1.4.3. Machine Learning Framework. The impact scoring component shall be implemented using Python with TensorFlow and Keras.

3.1.4.4. Model Serving. The machine learning model shall be served using TensorFlow Serving. The following requirements shall be met:

1. The machine learning system shall provide a REST API to call the machine learning model.

2. The machine learning system, upon receiving an article title, shall return an impact score of 1, 2, or 3.

3.1.5. Push-Notifications (Angelos, Chris, Brenden). The system shall prepare and send notifications to users when new articles have been found for a game in their watchlist. The following requirements shall be met:

3.1.5.1. Notification Backend. Notifications shall be sent by the primary GameEye backend via Firebase Cloud Messaging.

3.1.5.2. Process. Notifications shall be sent immediately after new articles have been found. The following requirements shall be met:

1. Scraped article *title* fields shall be scanned to find which specific game they refer to. This shall be done by finding the longest possible game *title* contained in an article *title* from all game *title* fields stored in the *games* collection.

2. Articles that do not refer to a specific game shall be deleted.

3. The *title* fields of the articles that remain shall be sent to the machine learning backend for impact score calculation.

4. For all new articles, a list of users that have the games they refer to in their watchlists shall be compiled.

5. Notifications shall be sent to only the users that watch games for which new articles have been found.

3.1.5.3. Game-Specific Notifications. Game-specific notifications shall be sent when new articles have been found for a specific game. Article data is derived from the nested *articles* structure in the *games* collection as specified in section 3.1.8.3. The following requirements shall be met:

1. A notification's content shall include:

a. Game *title*

b. Categories of scraped resources, such as "news article"

3.1.5.4. Notification Grouping. Notifications for several different games shall be grouped and sent as a single notification to avoid flooding the user's inbox if multiple games had new articles during a scraping cycle. The following notification content is required:

1. Grouped notifications shall state that new game information has been found for several games.

3.1.5.5. Cross-Platform Notifications. Users shall receive notifications on each of the following platforms:

1. Desktop

2. Mobile devices

3.1.6. Primary Backend Endpoints (Angelos, Ben, Jacob). The primary backend serves requests from the frontend, coordinates web scraping, and interacts with the database. Endpoint requests shall be structured in JSON. The following requirements shall be met:

3.1.6.1. User Profile Endpoints. A set of endpoints that coordinate user profile operations shall be created. User profile database fields are defined in section 3.1.8.2. The following endpoints shall be implemented:

3.1.6.1.1. User Profile Creation. An endpoint that stores a user profile in the GameEye database shall be created. The following requirements shall be met:

1. Retrieve the user *id* from the ID token contained in the request.

2. Create a user document if the user *_id* does not match any within the *users* collection.

3.1.6.1.2. User Profile Deletion. An endpoint for deleting a user profile shall be created. The following requirements shall be met:

1. Retrieve the user *id* from the ID token contained in the request.

2. Delete the user profile matching the received user *id* from the *users* collection.

3. Delete the user account matching the received user *id* from Firebase.

3.1.6.1.3. User Profile Settings. An endpoint that stores user profile settings shall be created. The following requirements shall be met:

1. Retrieve the user *id* from the ID token contained in the request.

2. Receive new user profile settings from the request.

3. Update any settings in the database that differ to match the new user profile settings.

3.1.6.2. Watchlist Endpoints. A set of endpoints for coordinating operations on user watchlists shall be created. The following endpoints shall be implemented:

3.1.6.2.1. Retrieve Watchlist Games. An endpoint for retrieving data from all games within a user's watchlist shall be created. The following requirements shall be met:

1. Retrieve the user ID from the ID token contained in the request.

2. Return data for all games in the user's watchlist. The following data is required:

a. *title*

b. Game thumbnail or logo

c. *notificationCount*

3.1.6.2.2. Add Watchlist Game. An endpoint that adds a new game to a user's watchlist shall be created. The following requirements shall be met:

1. Retrieve the user *id* from the ID token contained in the request.

2. Retrieve a game *id* from the request

3. Add the game referenced by the received game *id* to the user's watchlist

3.1.6.2.3. Delete Watchlist Game. An endpoint for deleting a game from a user's watchlist shall be created. Users' *watchlist* structures are contained within the *users* collection as specified in section 3.1.8.2. The following requirements shall be met:

1. Retrieve the user *id* from the ID token contained in the request.

2. Retrieve a game index from the request

3. Delete the game under the retrieved index from the user's watchlist.

3.1.6.3. Searching Endpoints. A set of endpoints for searching games using autocompletion shall be created. The following endpoints shall be implemented:

3.1.6.3.1. Game Title Autocompletion. An endpoint that returns a list of autocompleted game *title* fields given a partial game *title* shall be created. The following requirements shall be met:

1. Retrieve a game *title* fragment from frontend.

2. Query the ElasticSearch database to retrieve a list of autocompleted game *title* fields with their *id* fields.

4. Return the list of autocompleted game *title* fields with their *id* fields.

3.1.6.4. IGDB Endpoints. A set of administrative endpoints for interfacing with IGDB game data shall be created. The following endpoints shall be implemented:

3.1.6.4.1. IGDB Replication Endpoint. An endpoint shall be created that accepts minimum and maximum IGDB IDs. The endpoint shall accept a limit parameter. The endpoint shall populate the GameEye database with IGDB game data for games within the inclusive minimum-maximum range. Requests shall contain up to a number of games equal to the limit parameter. The endpoint shall update existing game data. The following requirements shall be met:

1. Retrieve the minimum and maximum IGDB game IDs contained in the request.

2. Retrieve the limit parameter contained in the request.

3. Clone game data from IGDB to the GameEye database. Game data from IGDB shall be copied into the *games* collection as specified in section 3.1.8.3. The following GameEye database fields shall be populated from the IGDB database:

a. *igdbId*

b. *title*

c. *platforms*

d. *genres*

e. *sourceUrls*

f. *logoUrl*

4. Update existing GameEye database game data. Existing games in the GameEye database shall be updated from IGDB data. Only the following fields shall be updated from the IGDB database:

a. *title*

b. *platforms*

c. *genres*

d. *sourceUrls*

e. *logoUrl*

5. Update ElasticSearch with game *title* fields. Game *title* updates to the ElasticSearch database shall be triggered when either of the following events occur:

a. Clone game data from IGDB to the GameEye database

b. Update existing GameEye database game data

3.1.6.5. Game Endpoints (Jacob). A set of endpoints for retrieving game information shall be created. The following endpoints shall be implemented:

3.1.6.5.1. Game Logo. An endpoint shall be created that returns the logo of the respective game. The following requirements shall be met.

1. Retrieve the game *id* contained in the request.

2. Return the *logoUrl* of the game referenced by the passed *id*

3.1.6.5.2. List of News Articles for Game. An endpoint shall be created that given a game *id* returns all articles that reference the game. Article data is derived from the *games* collection specified in section 3.1.8.3. The following requirements shall be met:

1. Retrieve the game *id* contained in the request.

2. Return an array of articles for the given game *id*. The following data shall be returned for each article:

a. *title*

b. *snippet*

c. *publicationDate*

d. *newsWebsites.logo*

e. *thumbnail*

f. *impactScore*

g. *url*

3.1.6.5.3. Most-Watched Games List. An endpoint shall be created that returns

the most-watched games on the platform. The following requirements shall be met:

1. Query the *games* collection to construct a list of the top 50 games with the highest *watchers* field value.

2. Provide a list of the top 50 game titles along with their number of watchers to the frontend.

3.1.7. User Settings (Jonathan). Users shall customize the content that is shown to them and the notifications that they receive. The following requirements shall be met:

3.1.7.1. Content. Users shall choose what type of content they want to receive.

1. Provide the ability for the user to select the impact score option.

3.1.7.2. Notifications. Users shall be able to customize their notification preferences.

1. Provide the ability for the user to select the notification option.

2. Provide the ability for the user to select the impact score level option.

3.1.8. GameEye Database (Jacob). The GameEye database shall store news website data, user data, game data, and image data. Data shall be stored in four collections in MongoDB using the BSON document format. The BSON document format supports specific data types. Each field shall be provided in the specified data type. Note that each collection shall be named case-sensitively according to the functional requirements. The following database structures shall be implemented:

3.1.8.1. *newsWebsites* Collection. The newsWebsites collection shall store data related to new websites. The web scrapers use data in the newsWebsites collection to retrieve news articles for video games. The following fields and corresponding data types shall be provided:

Name	Data Type	Description
_id	ObjectId	Unique ID generated by MongoDB and may be referred to as *id* due to field mapping between MongoDB and Spring Boot.
name	String	The name of the news website.
logo	Binary	The binary data representation of a logo.
siteUrl	String	The URL of the news website's main page.
rssFeedUrl	String	The URL of the news website's RSS feed.

lastUpdated	Date	The last time the news website database document was updated. Specified in UTC datetime.

3.1.8.2. *users* Collection. The *users* collection shall store data related to users. User data includes the authentication data. Personal data for following video games are stored in the *users* collection. The *users* collection shall store personal preferences for each user. The *users* collection shall store single-layer fields and multi-layer fields. The following database structures shall be implemented:

3.1.8.2.1. Single-Layer Fields. The *users* collection shall contain the following single-layer fields and corresponding data types:

Name	**Data Type**	**Description**
_id	String	Unique ID set to the user's firebase ID and may be referred to as id due to field mapping between MongoDB and Spring Boot.
status	String	Status of the user. Set logically as an enumerated type of "active" or "inactive."
plan	String	Plan type for the user.

3.1.8.2.2. Multi-Layer Fields. The *users* collection shall contain the following multi-layer fields and corresponding data types:

1. *preferences* – Object
   a. *contentPreferences* - Object
      i. *showArchivedResources* – Boolean
      ii. *showImpactScores* – Boolean
      iii. *impactScores* – Boolean Array
   b. *notificationPreferences* – Object
      i. *showArticleResources* – Boolean
      ii. *showImageResources* – Boolean
2. *watchList* - Object Array; each object shall contain the following fields:
   a. *gameId* – String
   b. *notificationCount* – Integer
   c. *resourceNotifications* – Object
      i. *articleNotifications* – Object

1. *count* – Integer
2. *articleIds* – String Array

ii. *imageNotifications* – Object

1. *count* – Integer
2. *imageIds* – String Array

3.1.8.3. *games* Collection. The *games* collection shall store data gathered from web scraping. Data from external databases such as IGDB shall be stored in the *games* collection. The *games* collection shall store single-layer fields and multi-layer fields. The following database structures shall be implemented:

3.1.8.3.1. Single-Layer Fields. The *games* collection shall contain the following single-layer field**s:**

Name	**Data Type**	**Description**
_id	ObjectId	Unique ID generated by MongoDB and may be referred to as id due to field mapping between MongoDB and Spring Boot.
igdbId	String	Unique ID set by the IGDB endpoint that identifies the IGDB game entry.
title	String	Title of the game derived from IGDB.
watchers	Integer	Number of users watching the game.
platforms	String Array	Array of platforms on which a game is available. Derived from IGDB.
status	String	Status of the game such as “Released.”
lastUpdated	Date	The last time the game database document was updated. Specified in UTC datetime.
genres	String Array	Array of game genres to which a game belongs. Derived from IGDB.
logoUrl	String	URL of the game logo that is derived from IGDB.

3.1.8.3.2. Multi-Layer Fields. The *games* collection shall contain the following multi-layer fields:

1. *sourceUrls* – Object
   a. *publisherUrl* – String
   b. *steamUrl* – String
   c. *subRedditUrl* – String
   d. *twitterUrl* – String
2. *resources* – Object
   a. *images* – Object Array
      i. *_id* – ObjectId
      ii. *title* – String
      iii. *imageId* – String
      iv. *lastUpdated* – Date
   b. *articles* – Object Array
      i. *_id* – ObjectId
      ii. *title* – String
      iii. *url* – String
      iv. *newsWebsite* – DBRef to the newsWebsites collection
      v. *thumbnail* – DBRef to the images collection
      vi. *snippet* – String
      vii. *publicationDate* – Date
      viii. *lastUpdated* – Date
      ix. *impactScore* – Integer

3.1.8.4. *images* Collection. The *images* collection shall store images for news articles, news website logos, and games. Documents in the games collection reference the images collection for stored images via foreign key. The following fields and corresponding data types shall be provided:

Name	Data Type	Description
_id	ObjectId	Unique ID generated by MongoDB and may be referred to as id due to field mapping between MongoDB and Spring Boot.
type	String	Type of image such as “logo” or “thumbnail.”
data	Binary	The binary data representation of an image.

3.1.9. Web Scraping (Chris and Brenden). The system shall retrieve news articles from video game news RSS feeds of target news websites. Target news website data is stored in the *newsWebsites* collection. Each news website source listed in section 3.1.9.2 shall be pre-defined in the *newsWebsites* collection. Data from web scraping is inserted into the *games* collection. The following requirements shall be met:

3.1.9.1. Interval. News articles shall be retrieved from the RSS feeds twice a day.

3.1.9.2. Sources. News articles shall be retrieved from the following news sources:

1. IGN

2. GameSpot

3. Eurogamer

4. PC Gamer

5. GameEye Mock News

3.1.9.3. Information. The following elements shall be scraped from each news article in the RSS feed and populated into the *games.resources.articles* nested structure:

1. *title*

2. *snippet*

3. *publicationDate*

4. *url*

5. *thumbnail*

3.1.9.4. Force Scrape. The system shall have functionality for force scraping for demonstration purposes.

3.1.9.5. Organization. News articles shall be organized by which game *title* they are referencing from the *games* collection.

3.1.9.6. Reference Game Search. Each scraped article shall be analyzed to determine which specific game it is referencing. The following requirements shall be met:

1. A search shall be made to ElasticSearch using the article *title* as the query.

2. The search results shall include the top 10 most likely games to be referenced in the article *title*.

3. Each returned game *title* shall be searched in an article's *title*.

4. The longest exactly matching game *title* shall be set as the article's reference game.

5. If there are multiple exact matches, a scraped article shall be set to reference multiple games

3.1.9.7. Data Clean-up. Web scrapers shall delete recently scraped news articles not explicitly related to any games contained within the *games* collection.

3.1.9.8. Duplicates. Web scrapers shall delete any scraped news articles already present in the *games* collection.

3.1.9.9. Database Integration. The set of processed data collected from web scraping will be inserted into GameEye's database. The following requirements shall be met:

1. News articles that reference games not already present in the GameEye database will be added to the *games* collection.

2. Images attached to scraped articles will be stored in the *images* collection.

3. Websites used by web scrapers will be stored in the *newsWebsites* collection.

3.1.10. Watchlist (Adrian). The system shall keep track of the games a user has selected to follow, known as their watchlist. The following requirements shall be met:

3.1.10.1. Add Game. The system shall allow users to add new games to their watchlist. The following requirements shall be met:

1. Provide the user with a search bar to find games they wish to follow.

2. The search bar shall have an auto-completion feature, which displays suggestions beneath the search bar.

3. Add the selected games to the user's watchlist.

3.1.10.2. Remove Game. The system shall allow users to remove games from their watchlist.

3.1.10.3. Watched Games. The system shall display the user's current watchlist. The following requirements shall be met:

3.1.10.3.1. Watchlist Entries. The system shall implement material cards to represent watchlist entries. The following requirements shall be met:

1. Each card shall describe the game it represents with the title and a thumbnail.

2. Each card shall include the notification count of the total amount of new game updates.

3. The user shall be able to access the updates for a specific game by clicking on the corresponding card.

3.1.10.3.2. Game Updates. The system shall organize the update information into specific categories. The following requirements shall be met:

3.1.10.3.2.1. Update Card. The system shall implement material cards to represent each update category. The following requirements shall be met:

1. Each card shall describe the category it represents with a *title* and a *thumbnail*.

2. Each card shall include a notification count of the new game updates for the category it represents.

3.1.10.3.2.2. News Articles. The system shall list the recent articles it has collected along with relevant descriptors. The following requirements shall be met:

1. The title of the article as it appears on the source site.

2. Article impact scores shall be displayed for each listing.

3. Each listing should contain the logo from the site the article was published.

4. Each listing should indicate the amount of time that has passed since the article was published.

5. The user shall be directed to the source site by clicking on an article's designated region.

3.1.11. Most-Watched Games List (Angelos, Adrian, Jonathan). GameEye shall provide a list of the most-watched games on the platform. The following requirements shall be met:

3.1.11.1. Watcher Counter. The backend shall keep track of how many users are watching a particular game. The following requirements shall be met:

1. When a user adds a new game to their watchlist, the watcher counter, *watchers*, for the appropriate game shall be incremented.

2. When a user removes a game from their watchlist, the watcher counter, *watchers*, for the appropriate game shall be decremented.

3.1.11.2. Watcher Counter Sorting. The backend shall query the database to retrieve the games with the highest watcher counters, *watchers*. The following requirements shall be met:

1. The backend shall query the database to retrieve the top 50 most watched games.

2. The game *titles* along with their watcher count shall be available for the frontend.

3.1.12. Cross-platform (Angelos, Adrian, Jonathan). GameEye shall be implemented as a cross-platform web application. The following requirement shall be met:

3.1.12.1. Responsiveness. GameEye shall use responsive web design, making it able to adjust to various screen sizes. The following requirements shall be met:

1. GameEye shall be usable from devices with large-sized screens such as desktop computers

2. GameEye shall be usable from devices with medium-sized screens, such as laptops.

3. GameEye shall be usable from devices with small-sized screens, such as mobile phones.

3.1.12.2. Browser Support. GameEye shall be able to run on major web browsers. The following browsers shall be officially supported:

1. Google Chrome

2. Microsoft Edge

3. Mozilla Firefox

4. Opera

5. Brave

6. Safari

3.1.13. Testing (Everyone). Extensive testing shall be implemented, including with mock data, to ensure that the front-end and back-end GameEye components work as intended. The following requirement shall be met:

3.1.13.1. Backend Testing. Automated tests for the backend shall be implemented to test web scraping, endpoints, services, and integration of the various backend components.

1. Unit tests shall be implemented for isolatable backend components. The following components shall be tested:

a. API endpoints.

b. Services.

2. Integration tests shall be implemented for component interactions. The following interactions shall be tested:

a. Primary backend and database

b. Primary backend and machine learning backend

3.1.13.2. Frontend Testing. Automated tests for the frontend shall be implemented to ensure that frontend services and user flow work as intended.

1. Unit tests shall be implemented for independent frontend services.

2. Automated End-to-End (E2E) tests shall be implemented for the frontend. These tests shall simulate user interaction in a real browser.

3.1.13.3. Mock News Website. A mock news website shall be created that will contain news article titles. The following requirements shall be met

3.1.13.3.1. Articles Page. The mock news website shall display a list of mock news articles. The following requirements shall be met:

1. Each article shall have a title
2. Each article shall have a publication date
3. Each article shall have a short description of its content

3.1.13.3.2. Content Management System. The mock news website shall include a content management system (CMS) for managing mock news articles. The following requirements shall be met:

1. Users shall be able to add new articles
2. Users shall be able to edit existing articles
3. Users shall be able to upload images to use within articles
4. All articles shall be stored on a public GitHub repository