**DATABASE SPECIFICATION**

**1. Databases and Collections**

* **Users**
* **Countries**
* **EconomicData**
* **InvestmentData**
* **News**
* **Tweets**
* **SectorData**
* **SectorInvestmentData**

**2. Collection Specifications**

**2.1 Users Collection**

This collection will store information about users.

Attributes:

* **userId** (Unique Identifier): Auto-generated during user registration.
* **username** (String): Provided by the user during registration.
* **password** (Hashed String): User-provided password, securely hashed.
* **role** (Enum: 'novice', 'professional'): User-defined role. Novices will see simpler, high-level data. Professionals will have access to detailed data.
* **email** (String): User-provided email for communication and account recovery.
* **registeredAt** (Date): Auto-generated registration timestamp.

**2.2 Countries Collection**

This collection will store static data about all countries. This data can be populated from a reliable external source such as the RestCountries API.

Attributes:

* **countryId** (Unique Identifier): Auto-generated unique identifier.
* **name** (String): Country name.
* **continent** (String): Continent the country belongs to.
* **capital** (String): Capital of the country.
* **languages** (Array of Strings): Main languages spoken in the country.

**2.3 EconomicData Collection**

This collection will store economic data for each country. Data will be fetched and updated periodically from the World Bank API.

Attributes:

* **dataId** (Unique Identifier): Auto-generated unique identifier.
* **countryId** (Foreign Key linked to Countries.countryId): Identifier of the country the data belongs to.
* **year** (Integer): Year of the data.
* **GDP** (Float): Gross Domestic Product of the country for the year.
* **inflation** (Float): Inflation rate of the country for the year.
* **unemployment** (Float): Unemployment rate of the country for the year.
* **externalDebt** (Float): External debt of the country for the year.
* **population** (Integer): Population of the country for the year.

**2.4 InvestmentData Collection**

This collection will store investment data for each country, fetched and updated periodically from the World Bank API.

Attributes:

* **dataId** (Unique Identifier): Auto-generated unique identifier.
* **countryId** (Foreign Key linked to Countries.countryId): Identifier of the country the data belongs to.
* **year** (Integer): Year of the data.
* **FDI** (Foreign Direct Investment, Float): Total foreign direct investment received by the country for the year.
* **domesticInvestment** (Float): Total domestic investment for the country for the year.
* **foreignPortfolioInvestment** (Float): Total foreign portfolio investment for the country for the year.

**2.5 News Collection**

This collection will store business news related to each country, fetched from a reliable news API like GNews API.

Attributes:

* **newsId** (Unique Identifier): Auto-generated unique identifier.
* **countryId** (Foreign Key linked to Countries.countryId): Identifier of the country the news is related to.
* **timestamp** (Date): Timestamp of the news article.
* **title** (String): Title of the news article.
* **content** (String): Content of the news article.
* **source** (String): Source of the news article.
* **url** (String): URL of the news article.

**2.6 Tweets Collection**

This collection will store tweets related to the business environment of each country, fetched via Twitter API.

Attributes:

* **tweetId** (Unique Identifier): Auto-generated unique identifier.
* **countryId** (Foreign Key linked to Countries.countryId): Identifier of the country the tweet is related to.
* **timestamp** (Date): Timestamp of the tweet.
* **content** (String): Content of the tweet.
* **username** (String): Username of the tweet author.
* **retweetCount** (Integer): Count of retweets.
* **favoriteCount** (Integer): Count of favorites or likes.
* **sentiment** (Enum: 'positive', 'neutral', 'negative'): Sentiment of the tweet, determined using sentiment analysis algorithms.

**2.7 SectorData Collection**

This collection will store economic data for each sector in every country. Data will be fetched and updated periodically from the World Bank API or similar sources.

Attributes:

* **sectorId** (Unique Identifier): Auto-generated unique identifier.
* **countryId** (Foreign Key linked to Countries.countryId): Identifier of the country the data belongs to.
* **year** (Integer): Year of the data.
* **sectorName** (String): Name of the sector (e.g., "Manufacturing", "Service", "Agriculture").
* **sectorGrowth** (Float): Growth rate of the sector for the year.

**2.8 SectorInvestmentData Collection**

This collection will store investment data for each sector in every country, fetched and updated periodically from multiple sources.

Attributes:

* **dataId** (Unique Identifier): Auto-generated unique identifier.
* **countryId** (Foreign Key linked to Countries.countryId): Identifier of the country the data belongs to.
* **year** (Integer): Year of the data.
* **sectorName** (String): Name of the sector.
* **investmentAmount** (Float): Total investment in the sector for the year.

Implementing these databases will involve the following steps:

1. **Create the database and collections in MongoDB.** This will involve setting up a MongoDB instance and defining the collections with the given schemas.
2. **Fetch and load data into the Countries collection.** You can use the RestCountries API to fetch this data.
3. **Set up scheduled jobs to fetch and update data.** For the **EconomicData**, **InvestmentData**, **SectorData**, and **SectorInvestmentData** collections, you will need to set up jobs to fetch data from the World Bank API and other relevant sources periodically.
4. **Set up integrations with the Twitter and news APIs.** These will fetch and load data into the **Tweets** and **News** collections.
5. **Implement the user registration and login system.** This will involve creating routes to handle user registration and login, storing user data in the **Users** collection, and implementing password hashing for security.
6. **Implement the best business suggestion feature.** This will require building and training a machine learning model, and creating a route to fetch suggestions from this model.
7. **Create API endpoints to fetch and serve data to the front end.** These endpoints will allow the front end to fetch the necessary data from the back end.
8. **Implement data sanitation and validation.** This will ensure that only valid and clean data is stored in your database, helping to maintain data integrity and security.

**REALISTICALLY IMPLEMENTING THE DATABASES**

ChatGPT

Yes, it's realistically possible to implement all these databases and integrate them with your website. However, it requires a combination of skills, including knowledge of MongoDB, API usage, and backend development using Node.js or a similar technology.

Here's a step-by-step guide on how to go about it:

**Step 1: Setting up MongoDB**

The first step is setting up your MongoDB instance. MongoDB Atlas provides a cloud-based solution that you can use. Once you've set up your MongoDB instance, you can define the collections as specified in the database specification.

**Step 2: Setting up the Backend**

You'll need to set up a server that will interact with the MongoDB database and serve data to the front-end of your website. Express.js is commonly used with MongoDB for this purpose.

The backend server should include routes for handling various operations, such as fetching economic data for a specific country, registering a new user, etc. You'll need to define these routes in your Express.js application.

**Step 3: Fetching and Loading Data**

Many of the collections in your database will be populated with data fetched from external sources, such as the World Bank API, RestCountries API, Twitter API, and a news API.

You can fetch and load this data using HTTP requests. In Node.js, you can use libraries like axios to send these requests.

For each API, you'll need to send a request to the relevant endpoint, extract the data from the response, and then insert this data into the appropriate MongoDB collection.

**Step 4: Scheduled Jobs**

Since the economic, investment, and sector data needs to be updated periodically, you'll need to set up scheduled jobs to fetch and update this data. You can do this using a job scheduling library like node-cron.

**Step 5: User Registration and Login**

For the Users collection, you'll need to set up routes in your backend to handle user registration and login.

During registration, you should hash the user's password using a library like bcrypt before storing it in the database.

During login, you'll need to compare the hashed version of the provided password with the stored hashed password to authenticate the user.

**Step 6: Best Business Suggestion**

For the Best Business Suggestion feature, you'll need to build and train a machine learning model using data from your databases. This can be a complex task and may require knowledge of machine learning algorithms and libraries like scikit-learn or TensorFlow.

**Step 7: Front-End Integration**

Finally, your website's front-end will need to send requests to your backend to fetch and display data. You can do this using JavaScript's fetch API or a library like axios.

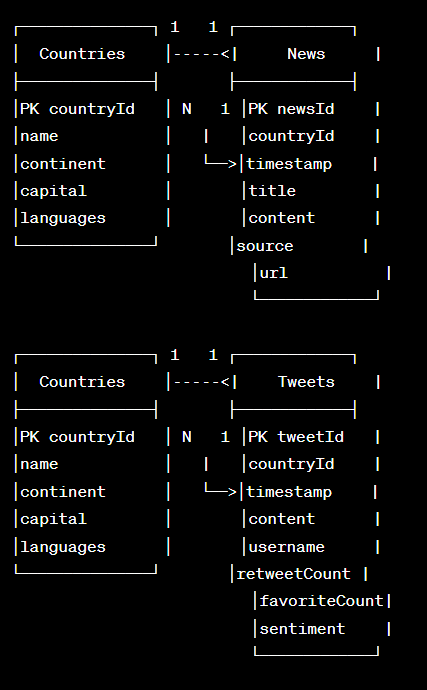
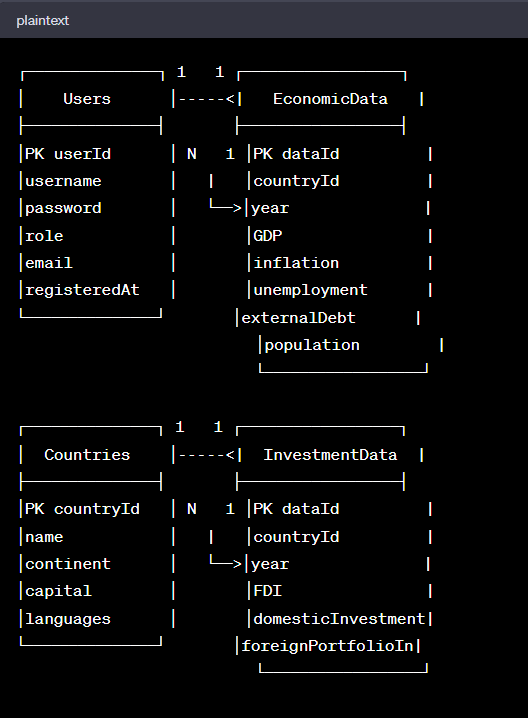
Remember to sanitize and validate the data both at input and output stages to maintain data integrity and security. Use middleware for express.js like **express-validator** for input validation and libraries like **mongo-sanitize** to prevent NoSQL injection attacks.

Also, it's important to consider error handling in your backend application. Proper error handling will make your application robust and easier to debug.

Do note that this is a broad overview and the exact details will depend on various factors including your tech stack, the APIs you're using, and the specifics of your project. You may need to dive deeper into the documentation of the respective technologies and APIs for a thorough understanding.

**ENTITY – RELATIONSHIP DIAGRAM**

**Below is a very detailed Entity-Relationship (ER) Diagram for the Orbit project's database based on the given specifications.**

****

****

**Here's what each component of the ER diagram means:**

* **Each box represents an Entity, which is a collection in your MongoDB database. The name of the Entity (Users, Countries, EconomicData, etc.) is at the top of the box.**
* **Inside each box, you have the list of Attributes of that entity. These are the fields within each document in the collection. The attribute with 'PK' before it is the Primary Key, which uniquely identifies each record in the collection.**
* **Between the boxes, you have Relationships. These are represented as lines connecting the Entities. The relationship is labeled with the cardinality of the relationship:**
  + **'1' means 'one', and 'N' means 'many'.**
  + **So, a '1' next to Users and an 'N' next to EconomicData means that one user can have many EconomicData.**
* **The 'Foreign Key' is shown as an attribute with the same name as the Primary Key from another entity and is used to establish a link between two entities.**

**For this model, it's worth noting that a graphical representation is usually easier to understand, but this textual representation carries all the necessary information. Also, please note that there are no recursive relationships or weak entities in the given database schema. If you need to use such advanced features, you would need to update the model accordingly.**

**Note: Remember that MongoDB, being a NoSQL database, does not enforce relationships like an SQL database would. The relationships here are logical and would need to be managed in your application code.**

**Based on your specifications, here's the ER diagram described:**

1. **Users Entity: The Users entity includes attributes such as userId (Primary Key), username, password, role, email, and registeredAt.**
2. **Countries Entity: This includes attributes like countryId (Primary Key), name, continent, capital, and languages.**
3. **EconomicData Entity: This includes attributes such as dataId (Primary Key), countryId (Foreign Key), year, GDP, inflation, unemployment, externalDebt, and population. The foreign key countryId establishes a relationship with the Countries entity. This relationship is one-to-many as one country can have multiple economic data records.**
4. **InvestmentData Entity: This includes dataId (Primary Key), countryId (Foreign Key), year, FDI, domesticInvestment, and foreignPortfolioInvestment. Similar to the EconomicData entity, the InvestmentData entity has a one-to-many relationship with the Countries entity through the countryId foreign key.**
5. **News Entity: This includes attributes like newsId (Primary Key), countryId (Foreign Key), timestamp, title, content, source, and url. The relationship between the Countries entity and the News entity is again one-to-many based on the countryId foreign key.**
6. **Tweets Entity: This includes tweetId (Primary Key), countryId (Foreign Key), timestamp, content, username, retweetCount, favoriteCount, and sentiment. The Tweets entity has a one-to-many relationship with the Countries entity through the countryId foreign key.**
7. **SectorData Entity: This includes sectorId (Primary Key), countryId (Foreign Key), year, sectorName, and sectorGrowth. The SectorData entity has a one-to-many relationship with the Countries entity based on the countryId foreign key.**
8. **SectorInvestmentData Entity: This includes dataId (Primary Key), countryId (Foreign Key), year, sectorName, and investmentAmount. Similar to the SectorData entity, the SectorInvestmentData entity has a one-to-many relationship with the Countries entity based on the countryId foreign key.**

**For Participation Constraints:**

* **In each one-to-many relationship from the Countries entity to all the other entities (EconomicData, InvestmentData, News, Tweets, SectorData, SectorInvestmentData), the country participation is total, as these data points are related to a country. The other entity participation is partial, as it's possible to have a country with no data point in these databases yet.**
* **For the Users entity, there's no relationship with other entities based on the provided specifications.**

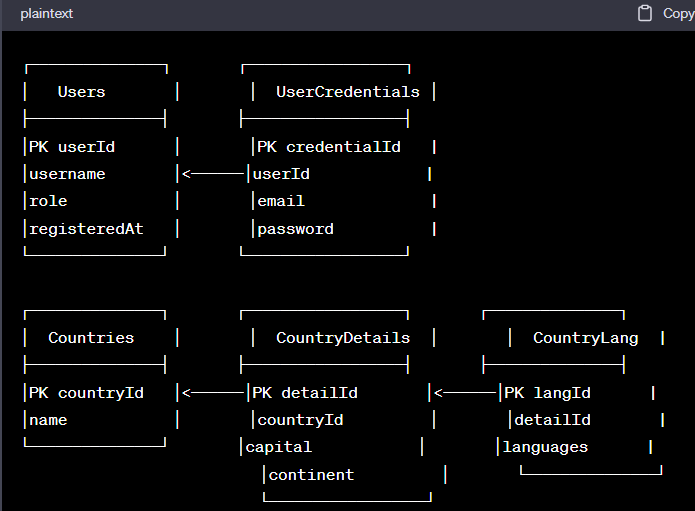
**For Cardinality Constraints:**

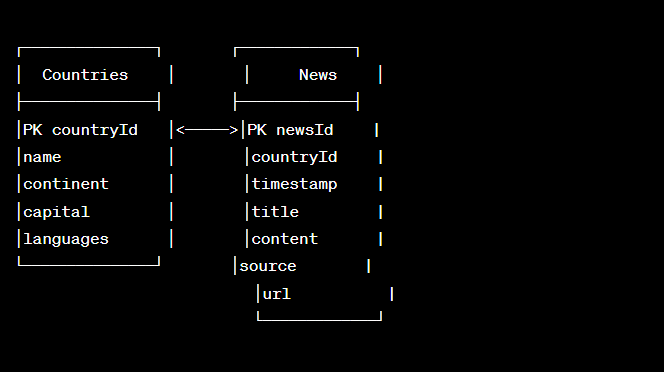
* **All relationships from Countries entity to other entities are one-to-many relationships.**

**Please note, the above description assumes that there's no relationship between users and the rest of the data which is the case based on your current specifications. If there's a need to link users to certain data points (for example, if users can save or comment on certain news or tweets), then the relationships, cardinality, and participation constraints would need to be adjusted accordingly.**

**EXTENDED ENTITY-RELATIONSHIP DIAGRAM (EERD**

**Here's an Extended Entity-Relationship Diagram (EERD) that represents the above specifications in a more detailed way using specialization, generalization, aggregation, and composition. Due to the text-based format, this diagram won't be as intuitive as a graphical one but will represent the same information:**

****

****

**Explanation:**

1. **The Users entity has been specialized into UserCredentials, storing the user's sensitive data separately.**
2. **The Countries entity has been specialized into CountryDetails and CountryLang. CountryDetails holds basic details like capital and continent. CountryLang holds the languages spoken in the country.**
3. **EconomicData and InvestmentData are related to Countries and each other using aggregation, showing the composition of a country's economic profile.**
4. **Similarly, SectorData, SectorInvestmentData, and Tweets are related to Countries, indicating each country's sector data, sector investment data, and tweets respectively.**
5. **The News entity also has a relationship with Countries, indicating that each news article is associated with a specific country.**

**This EERD provides a more detailed perspective on the data model, but remember that MongoDB doesn't enforce relationships at the database level like a relational database does. These relationships need to be managed in your application code.**

**Let's describe the extended entity-relationship diagram (EERD) for the ORBIT project database with focus on generalization, specialization, aggregation, composition, and multiplicity.**

**Users and UserCredentials:**

* **The Users entity can be specialized into the UserCredentials entity.**
* **The Users entity acts as a superclass, with the subclass UserCredentials inheriting the userId attribute.**
* **The UserCredentials entity has attributes for email and password, uniquely associated with a specific user.**
* **The multiplicity between Users and UserCredentials is 1:1 (one-to-one), as each User has one set of UserCredentials and vice versa.**

**Countries, CountryDetails, and CountryLang:**

* **The Countries entity is a superclass and it's specialized into two subclasses: CountryDetails and CountryLang.**
* **Both these subclasses inherit the countryId from the Countries superclass.**
* **The CountryDetails entity has attributes for capital and continent, while CountryLang has an attribute for languages.**
* **The multiplicity between Countries and CountryDetails is 1:1, as each Country has one set of CountryDetails and vice versa. Similarly, the multiplicity between Countries and CountryLang is also 1:1.**

**EconomicData and InvestmentData:**

* **EconomicData and InvestmentData can be seen as composed entities for a country's economic profile, using aggregation.**
* **Both these entities share a relationship with the Countries entity, and the multiplicity between Countries and each of these entities is 1:N (one-to-many), as each country can have multiple sets of economic and investment data (for different years).**
* **Both EconomicData and InvestmentData have a year attribute, indicating the specific year the data corresponds to, along with various other country-specific data attributes.**

**SectorData and SectorInvestmentData:**

* **Similar to EconomicData and InvestmentData, SectorData and SectorInvestmentData can be seen as composed entities representing a country's sector profile.**
* **The relationship between Countries and each of these entities is also 1:N, as each country can have multiple sets of sector data and sector investment data.**
* **Both these entities have a year attribute and a sectorName attribute, indicating the specific year and sector the data corresponds to.**

**Tweets:**

* **The Tweets entity can be seen as a composed entity for a country's social media profile.**
* **The relationship between Countries and Tweets is 1:N, as each country can have multiple related tweets.**
* **The Tweets entity has attributes like timestamp, content, and sentiment, representing the specifics of a tweet and its sentiment analysis.**

**News:**

* **The News entity can be seen as another composed entity representing a country's news profile.**
* **The relationship between Countries and News is 1:N, as each country can have multiple news articles related to it.**
* **The News entity has attributes like timestamp, title, and content, representing the specifics of a news article.**

**In conclusion, the ER diagram for the ORBIT project database uses specialization to segregate sensitive data (UserCredentials), generalization to represent common features (Countries), and aggregation to represent composed entities (EconomicData, InvestmentData, SectorData, SectorInvestmentData, Tweets, and News). Multiplicity is also defined for each of the relationships.**