**Authentication Subsystem**

**Class**: Auth

* **Attributes**:
  + user\_id: integer, private - Unique identifier for the user.
  + session\_token: string, private - Token that maintains the session state across interactions.
  + email\_host: string, private - SMTP server host, derived from config (EMAIL\_HOST).
  + email\_port: integer, private - SMTP server port, derived from config (EMAIL\_PORT).
  + email\_user: string, private - SMTP server username, derived from config (EMAIL\_USER).
  + email\_password: string, private - SMTP server password, derived from config (EMAIL\_PASSWORD).
* **Operations**:
  + verify\_credentials(username, password): boolean, public - Validates user credentials against stored data.
  + send\_verification\_email(email\_address): void, public - Sends an email to verify the user's email address using SMTP settings.

**Monitoring Subsystem**

**Class: PriceMonitor**

* **Attributes**:
  + url: string, private - URL to monitor for price changes.
  + current\_price: float, private - Most recent fetched price.
* **Operations**:
  + check\_price(): float, public - Fetches and returns the current price from the URL.
  + start\_monitoring(frequency): void, public - Begins monitoring the price at specified intervals.
  + stop\_monitoring(): void, public - Stops monitoring the price.

**Class: AvailabilityMonitor**

* **Attributes**:
  + url: string, private - URL to monitor for availability changes.
  + availability\_status: string, private - Latest availability status fetched.
* **Operations**:
  + check\_availability(): string, public - Checks and returns the current availability from the URL.
  + start\_monitoring\_availability(frequency): void, public - Starts monitoring availability at specified intervals.
  + stop\_monitoring\_availability(): void, public - Stops monitoring availability.

**Browser Operation Subsystem**

**Class**: BrowserController

* **Attributes**:
  + browser\_instance: Object, private - Instance of the browser controlled by the bot.
* **Operations**:
  + open\_browser(url): void, public - Opens a browser window and navigates to the specified URL.
  + close\_browser(): void, public - Closes the currently open browser window.
  + login(url, username, password): boolean, public - Logs into a website with provided credentials.

**Data Handling Subsystem**

**Class**: DataHandler

* **Attributes**:
  + database\_connection: Object, private - Connection object for the database.
* **Operations**:
  + save\_data(data): void, public - Saves specified data to the database.
  + retrieve\_data(query): Object, public - Retrieves data from the database based on the query.
  + export\_data\_to\_excel(data): void, public - Exports data to an Excel file.
  + export\_data\_to\_html(data): void, public - Converts and saves data in HTML format.

**Communication Subsystem**

This subsystem facilitates all user interactions and communications, both inbound and outbound.

**Class: MessagingSubsystem**

* **Attributes**:
  + command\_parser: CommandParser, private - An instance of the CommandParser class to parse user commands.
  + response\_templates: dict, private - A dictionary mapping commands to response templates for generating user messages.
* **Operations**:
  + receive\_message(user\_input): string, public - Receives input from the user, parses it to determine the command, and retrieves the appropriate response.
  + send\_message(channel\_id, message): void, public - Sends a message to a specific Discord channel, using the channel ID.
  + format\_response(command, data): string, public - Formats the response based on the command executed and the data retrieved from the system.

**Class: NotificationSubsystem**

* **Attributes**:
  + email\_config: EmailConfig, private - Configuration settings for email, containing host, port, user, and password.
* **Operations**:
  + send\_email(subject, body, recipient): void, public - Sends an email to the specified recipient with a subject and body.
  + notify\_user(channel\_id, message): void, public - Sends a notification message to a user via a specific Discord channel.
  + generate\_email\_body(template\_id, data): string, public - Generates the body of the email based on a template and data, useful for price changes, availability updates, etc.

**Mapping Associations**

1. **Authentication Subsystem**
   * **Auth** associates with **NotificationSubsystem** for sending verification emails.
     + **Association Type**: Composition
     + **Multiplicity**: One Auth may utilize one NotificationSubsystem to handle email functionalities.
2. **Monitoring Subsystem**
   * **PriceMonitor** and **AvailabilityMonitor** should be able to log data using **DataHandler**.
     + **Association Type**: Aggregation
     + **Multiplicity**: One Monitoring class may use one DataHandler to log multiple datasets.
   * Both monitors may need to send notifications through **NotificationSubsystem** when specific thresholds or conditions are met.
     + **Association Type**: Aggregation
     + **Multiplicity**: One Monitoring class can use one NotificationSubsystem for various alert purposes.
3. **Browser Operation Subsystem**
   * **BrowserController** might need to authenticate via the **Auth** class for operations requiring secure access.
     + **Association Type**: Aggregation
     + **Multiplicity**: One BrowserController uses one Auth for managing user sessions and logins.
4. **Data Handling Subsystem**
   * **DataHandler** might be used by almost every other subsystem for data logging and retrieval purposes, establishing a broad association with many classes.
     + **Association Type**: Aggregation
     + **Multiplicity**: Multiple classes (like PriceMonitor, AvailabilityMonitor, etc.) can use one DataHandler.
5. **Communication Subsystem**
   * **MessagingSubsystem** uses **DataHandler** to retrieve data for response formatting.
     + **Association Type**: Aggregation
     + **Multiplicity**: One MessagingSubsystem retrieves and processes multiple data points from DataHandler.
   * **NotificationSubsystem** should interact with **DataHandler** to log notification activities or email statistics.
     + **Association Type**: Aggregation
     + **Multiplicity**: One NotificationSubsystem records multiple logs via DataHandler.

**Mapping Contracts to Exceptions**

**Auth Class**:

* **verify\_credentials(username, password)**:
  + **Contract**: Username and password must not be null or empty.
  + **Exceptions**:
    - InvalidCredentialsException: Thrown if the credentials do not match.
    - NullValueException: Thrown if any parameter is null.
* **send\_verification\_email(email\_address)**:
  + **Contract**: Email address must be valid and not null.
  + **Exceptions**:
    - InvalidEmailException: Thrown if the email format is invalid.
    - EmailSendFailureException: Thrown if the email fails to send.

**Monitoring Classes (PriceMonitor & AvailabilityMonitor)**:

* **check\_price() / check\_availability()**:
  + **Contract**: URL must be accessible and properly formatted.
  + **Exceptions**:
    - URLNotReachableException: Thrown if the URL cannot be accessed.
    - InvalidURLException: Thrown if the URL format is incorrect.

**BrowserController Class**:

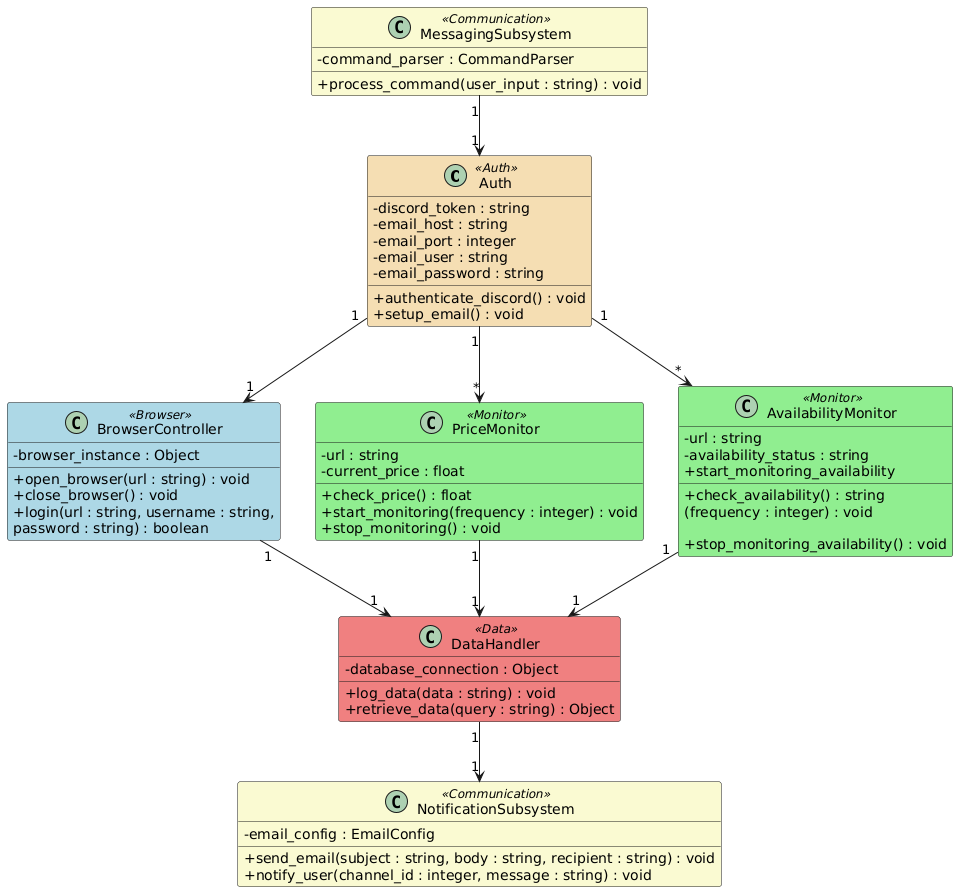
* **open\_browser(url)**:
  + **Contract**: URL must be valid and the browser instance must not already be open.
  + **Exceptions**:
    - BrowserAlreadyOpenException: Thrown if attempting to open a new browser session while one is already active.
    - InvalidURLException: Thrown if the URL format is incorrect.

**DataHandler Class**:

* **save\_data(data)**:
  + **Contract**: Data must not be null and database connection must be active.
  + **Exceptions**:
    - DatabaseConnectionException: Thrown if the connection is inactive.
    - NullDataException: Thrown if null data is passed.

**MessagingSubsystem Class**:

* **send\_message(channel\_id, message)**:
  + **Contract**: Channel ID must exist and message must not be null.
  + **Exceptions**:
    - InvalidChannelException: Thrown if the channel does not exist.
    - NullMessageException: Thrown if the message is null.



@startuml

skinparam classAttributeIconSize 0

skinparam class {

BackgroundColor<<Auth>> Wheat

BackgroundColor<<Monitor>> LightGreen

BackgroundColor<<Browser>> LightBlue

BackgroundColor<<Data>> LightCoral

BackgroundColor<<Communication>> LightGoldenRodYellow

}

' Authentication Subsystem

class Auth <<Auth>> {

- discord\_token : string

- email\_host : string

- email\_port : integer

- email\_user : string

- email\_password : string

+ authenticate\_discord() : void

+ setup\_email() : void

}

' Browser Operation Subsystem

class BrowserController <<Browser>> {

- browser\_instance : Object

+ open\_browser(url : string) : void

+ close\_browser() : void

+ login(url : string, username : string,

password : string) : boolean

}

' Monitoring Subsystem

class PriceMonitor <<Monitor>> {

- url : string

- current\_price : float

+ check\_price() : float

+ start\_monitoring(frequency : integer) : void

+ stop\_monitoring() : void

}

class AvailabilityMonitor <<Monitor>> {

- url : string

- availability\_status : string

+ check\_availability() : string

+ start\_monitoring\_availability

(frequency : integer) : void

+ stop\_monitoring\_availability() : void

}

' Data Handling Subsystem

class DataHandler <<Data>> {

- database\_connection : Object

+ log\_data(data : string) : void

+ retrieve\_data(query : string) : Object

}

' Communication Subsystem

class MessagingSubsystem <<Communication>> {

- command\_parser : CommandParser

+ process\_command(user\_input : string) : void

}

class NotificationSubsystem <<Communication>> {

- email\_config : EmailConfig

+ send\_email(subject : string, body : string, recipient : string) : void

+ notify\_user(channel\_id : integer, message : string) : void

}

' Associations

Auth "1" --> "1" BrowserController

Auth "1" --> "\*" PriceMonitor

Auth "1" --> "\*" AvailabilityMonitor

BrowserController "1" --> "1" DataHandler

PriceMonitor "1" --> "1" DataHandler

AvailabilityMonitor "1" --> "1" DataHandler

DataHandler "1" --> "1" NotificationSubsystem

MessagingSubsystem "1" --> "1" Auth

@enduml