**Discord Bot Automation Assistant**

**Discord Bot Automation Assistant Chapter 3**

**Oguz Kaan Yildirim**

**307637**

# TABLE OF CONTENTS

[Table of Contents 2](#_Toc178958374)

[CHAPTER THREE: PROJECT ORGANIZATION/STRUCTURE 4](#_Toc178958375)

[1. Project Requirements 4](#_Toc178958376)

[1.1 Use Case Diagram 4](#_Toc178958377)

[1.1.1 Actors 4](#_Toc178958378)

[1.1.2 Use Case Groups 4](#_Toc178958379)

[1.1.3 Extensions 5](#_Toc178958380)

[1.2 Use Case Descriptions 6](#_Toc178958381)

[1.2.1 Account Management 6](#_Toc178958382)

[1.2.1.1 Add Account 6](#_Toc178958383)

[1.2.1.2 Fetch All Accounts 6](#_Toc178958384)

[1.2.1.3 Fetch Account by Website 6](#_Toc178958385)

[1.2.1.4 Delete Account 6](#_Toc178958386)

[1.2.2 Browser Operations 7](#_Toc178958387)

[1.2.2.1 Launch Browser 7](#_Toc178958388)

[1.2.2.2 Close Browser 7](#_Toc178958389)

[1.2.2.3 Navigate to Website 7](#_Toc178958390)

[1.2.3 Login 8](#_Toc178958391)

[1.2.3.1 Login to Website 8](#_Toc178958392)

[1.2.4 Price Monitoring 8](#_Toc178958393)

[1.2.4.1 Get Price 8](#_Toc178958394)

[1.2.4.2 Start Monitoring Price 8](#_Toc178958395)

[1.2.4.3 Stop Monitoring Price 9](#_Toc178958396)

[1.2.5 Availability Monitoring 9](#_Toc178958397)

[1.2.5.1 Check Availability 9](#_Toc178958398)

[1.2.5.2 Start Monitoring Availability 9](#_Toc178958399)

[1.2.5.3 Stop Monitoring Availability 10](#_Toc178958400)

[1.2.6 DataExport 10](#_Toc178958401)

[1.2.6.1 DataExport 10](#_Toc178958402)

[1.2.7 Notifications and Emailing 10](#_Toc178958403)

[1.2.7.1 Receive Email 10](#_Toc178958404)

[1.2.7.2 Chat with Bot 11](#_Toc178958405)

# CHAPTER THREE: PROJECT ORGANIZATION/STRUCTURE

This chapter covers the system design and implementation details of the Discord Bot Automation Assistant. We will start with the project requirements, including the use case diagram and descriptions of the use cases. Then, we will present the architecture, including UML component, deployment, and activity diagrams. The design section will include UML package and class diagrams. Finally, we will discuss the technology stack and frameworks used, followed by a short conclusion.

## Project Requirements

In this section, we will cover the project requirements, including the use case diagram and detailed descriptions of the use cases. We will also integrate relevant parts from assignments to provide a comprehensive understanding.

### Use Case Diagram

#### Actors

* **User**: Represents the person interacting with the bot.
* **ExternalHelpers**: External systems or tools that assist with browser operations (e.g., Selenium).

#### Use Case Groups

* **Account Management**: Covers account-related commands.
* **Browser Operations**: Covers commands related to launching, closing, and navigating the browser.
* **Login**: Dedicated to logging into websites using stored credentials.
* **Price and Availability Monitoring**: Tracks prices and service availability, respectively.
* **Data Export**: Automatically exports data as part of the monitoring process.
* **Notifications & Email**: Sends email notifications with exported data.
* **Chat with Bot**: Serves as the primary communication interface where users send commands and interact with the bot.

#### Extensions

* **Data Export** is an extension of both **Price Monitoring** and **Availability Monitoring**, as exporting happens after data retrieval.
* **Notifications & Email** is an extension of **Data Export**, as emails are sent after exporting data.
* **Chat with Bot** extends other use cases, as it serves as the interface through which all actions are triggered.

A diagram of a diagram

Description automatically generated

### Use Case Descriptions

#### Account Management

This group of use cases handles managing user accounts in the bot.

##### Add Account

* + **Description**: Allows the user to store account credentials for a specific website.
  + **Actor**: User
  + **Precondition**: The user provides valid credentials (username, password, and website).
  + **Postcondition**: The account details are stored in the bot's database for future use.
  + **Command**: !add\_account <username> <password> <website>

##### Fetch All Accounts

* + **Description**: Retrieves all stored accounts for the user.
  + **Actor**: User
  + **Precondition**: The user has accounts stored in the database.
  + **Postcondition**: The bot returns a list of all stored accounts.
  + **Command**: !fetch\_all\_accounts

##### Fetch Account by Website

* + **Description**: Fetches the stored account details for a specific website.
  + **Actor**: User
  + **Precondition**: The user has an account stored for the specified website.
  + **Postcondition**: The bot returns the account details for the specified website.
  + **Command**: !fetch\_account\_by\_website <website>

##### Delete Account

* + **Description**: Removes an account from the database.
  + **Actor**: User
  + **Precondition**: The user specifies the account ID of an existing account.
  + **Postcondition**: The account is deleted from the database.
  + **Command**: !delete\_account <account\_id>

#### Browser Operations

These use cases revolve around the bot automating browser activities.

##### Launch Browser

* + **Description**: Opens a browser window.
  + **Actor**: User
  + **Precondition**: No browser is currently open.
  + **Postcondition**: A browser window is opened, ready for further actions.
  + **Command**: !launch\_browser

##### Close Browser

* + **Description**: Closes an open browser window.
  + **Actor**: User
  + **Precondition**: A browser window is open.
  + **Postcondition**: The browser window is closed.
  + **Command**: !close\_browser

##### Navigate to Website

* + **Description**: Navigates the browser to a specified URL.
  + **Actor**: User
  + **Precondition**: The browser is open, and a valid URL is provided.
  + **Postcondition**: The browser navigates to the provided URL.
  + **Command**: !navigate\_to\_website <url>

#### Login

This use case focuses on logging into a website using stored credentials.

##### Login to Website

* + **Description**: Logs into a website using the credentials stored in the Account Management system.
  + **Actor**: User
  + **Precondition**: The user has stored credentials for the specified website.
  + **Postcondition**: The user is logged into the website within the browser.
  + **Command**: !login <website>

#### Price Monitoring

These use cases allow the bot to track the prices of products and export the data.

##### Get Price

* + **Description**: Fetches the current price of a product from a specified URL.
  + **Actor**: User
  + **Precondition**: A valid product URL is provided.
  + **Postcondition**: The bot returns the current price and logs it in an Excel and HTML file.
  + **Command**: !get\_price <url>

##### Start Monitoring Price

* + **Description**: Starts continuous monitoring of the price for a specified product.
  + **Actor**: User
  + **Precondition**: The user provides a valid URL.
  + **Postcondition**: The bot periodically checks the price and sends notifications if the price changes.
  + **Command**: !start\_monitoring\_price <url>

##### Stop Monitoring Price

* + **Description**: Stops continuous price monitoring.
  + **Actor**: User
  + **Precondition**: The bot is actively monitoring the price.
  + **Postcondition**: The monitoring process is stopped.
  + **Command**: !stop\_monitoring\_price

#### Availability Monitoring

These use cases monitor the availability of services (e.g., restaurant reservations).

##### Check Availability

* + **Description**: Checks the availability of a service for a specific date.
  + **Actor**: User
  + **Precondition**: The user provides a valid URL and date.
  + **Postcondition**: The bot returns availability information and logs it in an Excel and HTML file.
  + **Command**: !check\_availability <url> <date>

##### Start Monitoring Availability

* + **Description**: Starts continuous monitoring of availability for a service.
  + **Actor**: User
  + **Precondition**: The user provides a valid URL and date.
  + **Postcondition**: The bot periodically checks availability and sends notifications when it updates.
  + **Command**: !start\_monitoring\_availability <url> <date>

##### Stop Monitoring Availability

* + **Description**: Stops continuous availability monitoring.
  + **Actor**: User
  + **Precondition**: The bot is actively monitoring availability.
  + **Postcondition**: The monitoring process is stopped.
  + **Command**: !stop\_monitoring\_availability

#### DataExport

This use case automatically exports data retrieved from price and availability monitoring into Excel and HTML files.

##### DataExport

* + **Description**: Exports data to Excel and HTML formats after retrieving price or availability information.
  + **Actor**: System (triggered by other use cases)
  + **Precondition**: Price or availability data has been retrieved.
  + **Postcondition**: Data is exported and stored locally.

#### Notifications and Emailing

These use cases handle sending notifications and emails based on monitoring tasks.

##### Receive Email

* + **Description**: Sends an email containing the exported data (HTML or Excel file) to the user.
  + **Actor**: User
  + **Precondition**: The user provides the name of the exported file.
  + **Postcondition**: The specified file is sent to the user's email.
  + **Command**: !receive\_email <file\_name>

##### Chat with Bot

* + **Description**: Enables users to interact with the bot via Discord, sending commands and receiving results or notifications.
  + **Actor**: User
  + **Precondition**: The bot is online and listening for commands.
  + **Postcondition**: The bot responds to the user's commands with relevant results, errors, or notifications.
  + **Command**: !<command> (e.g., !get\_price, !start\_monitoring\_price, etc.)

## Architecture Section

The project **Discord Bot Automation Assistant** is designed around multiple core operations, including account management, browser automation, monitoring price and availability, data export, and user notifications. Each of these operations relies on different components structured into entity, control, and boundary objects.

### Account Management Subsystem

* **Entity:** This subsystem interacts with the database to manage user accounts. It uses AccountDAO to perform database operations like adding, deleting, and fetching accounts.
* **Control:** The AccountControl class handles the business logic related to account management. It processes commands from the boundary layer and invokes methods in the entity layer to manipulate data.
* **Boundary:** The AccountBoundary class receives commands from the user via Discord and interacts with the control layer. For example, commands like !add\_account or !fetch\_all\_accounts pass through the boundary and are processed by the control.

### Browser Automation Subsystem

* **Entity:** This subsystem, centered around BrowserEntity, uses Selenium to handle browser operations like opening, navigating, and closing browser windows.
* **Control:** BrowserControl manages the browser-related logic, including login processes and navigation. It acts as a middle layer between the boundary and entity, coordinating actions that require browser interaction.
* **Boundary:** The BrowserBoundary class listens for Discord commands such as !launch\_browser or !navigate\_to\_website, passing them to the control layer for processing.

### Price Monitoring Subsystem

* **Entity:** PriceEntity handles scraping product prices from websites using Selenium. It retrieves price information based on CSS selectors.
* **Control:** PriceControl orchestrates the price-monitoring tasks, including handling commands to start or stop monitoring prices. It also exports results to files and sends email notifications when price changes occur.
* **Boundary:** The PriceBoundary class listens for commands like !get\_price or !start\_monitoring\_price, triggering control operations to retrieve price information or begin price monitoring.

### Availability Monitoring Subsystem

* **Entity:** AvailabilityEntity interacts with websites to check for service availability, again using Selenium. It simulates user actions like selecting dates or navigating pages to determine availability.
* **Control:** AvailabilityControl processes commands related to availability checking and monitoring. It continuously monitors availability and sends email notifications when updates occur.
* **Boundary:** The AvailabilityBoundary class handles user commands like !check\_availability or !start\_monitoring\_availability, passing them to the control layer for processing.

### Data Export and Notification Subsystem

* **Entity:** This subsystem doesn't directly require a distinct entity class but instead utilizes utility classes like ExportUtils and email\_utils to handle exporting data to Excel or HTML formats and sending notifications.
* **Control:** Various control classes, such as PriceControl and AvailabilityControl, manage the export and notification process, triggering exports when new data is retrieved and emailing the results to users.
* **Boundary:** The boundary layer for this subsystem primarily interacts with users through existing boundaries like PriceBoundary and AvailabilityBoundary, which include commands for exporting data and receiving notifications.

### 6. Bot Communication Subsystem

* **Entity:** No direct entity objects are involved in bot communication.
* **Control:** BotControl manages commands related to bot operations such as !project\_help and !stop\_bot. It coordinates the bot’s communication and shutdown processes.
* **Boundary:** BotBoundary listens for user commands like !project\_help or !stop\_bot, interacting with the control layer to provide help or terminate the bot.