https://github.com/Gruppe1-Fulya/price_tracker/tree/main

https://www.youtube.com/watch?v=YVjtXFjKhYQ

# Price Tracker - Kundengeschichte

## Was macht diese Software?

**01** Ermöglichen Sie Benutzern, ein Konto unter Verwendung ihres Vor- und Nachnamens, ihrer E-Mail-Adresse und ihres Passworts zu erstellen.

**02** Ermöglichen Sie Benutzern, sich mit ihrer E-Mail-Adresse und ihrem Passwort in ihr Konto einzuloggen.

**03** Ermöglichen Sie Benutzern, Produkte, die sie verfolgen möchten, hinzuzufügen, indem sie URLs von beliebten Einkaufsseiten wie Hepsiburada.com, trendyol.com und amazon.com.tr eingeben.

**04** Zeigen Sie die Preise der verfolgten Produkte in einem grafischen Format an, um Benutzern zu ermöglichen, Preisentwicklungen im Laufe der Zeit zu visualisieren.

**05** Ermöglichen Sie Benutzern, Bedingungen für die von ihnen verfolgten Produkte festzulegen, z. B. eine Benachrichtigung zu erhalten, wenn der Preis unter einen bestimmten Betrag fällt.

**06** Lassen Sie das System diese Produkte im bestimmten Zeitpunkt überprüfen und die Preise aufzeichnen und vergleichen.

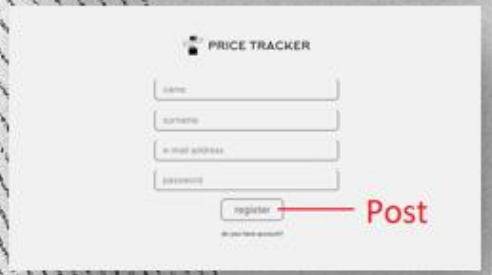
# Use Case - POST

Client

POST

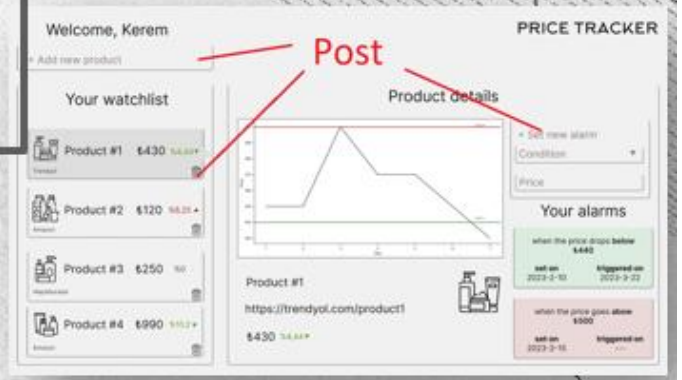
-Wenn ein Benutzer ein neues Konto erstellt, wird eine POST-Anfrage gesendet, um eine neue Benutzerentität in der Datenbank zu erstellen.


-Wenn ein Benutzer ein neues Produkt zu seiner Watchlist hinzufügt, wird eine POST-Anfrage gesendet, um einen neuen Watchlist in der Datenbank zu erstellen.




-Wenn dieses Produkt nicht existiert, wird ein neues Produkt in der Datenbank erstellt. Der Preis wird regelmäßig überprüft und in die Preisliste-Entität in der Datenbank aufgenommen.

-Wenn ein Benutzer seinen Alarm aktualisiert oder erzeugt, wird eine POST-Anfrage gesendet, um ihn in der Datenbank zu aktualisieren.




POST

Server

**Funktion**: Registrierung
**Eingaben**: Vorname, Nachname, E-Mail-Adresse, Passwort
**Verarbeitungsschritte**:

- Prüfe, ob die E-Mail-Adresse bereits in der Datenbank vorhanden ist.
- Wenn nicht, wird ein neuer Benutzer mit den eingegebenen Daten in der Datenbank erstellt.
- Der Benutzer wird zur Login-Seite weitergeleitet.
- Falls die E-Mail-Adresse bereits vorhanden ist, wird eine Fehlermeldung ausgegeben.

**Ausgaben**: Der Benutzer wird zur Login-Seite weitergeleitet oder eine Fehlermeldung wird angezeigt.

# USER CONTROLLER

createUser - POST


| User |
| --- |
| -int id |
| -String email |
| -String name |
| -String surname |
| -String password |
| -List<Watchlist> watchlists |

Es prüft anhand der angegebenen E-Mail-Adresse, ob der Benutzer bereits existiert. Wenn der Benutzer existiert, wird eine Konfliktantwort mit dem HTTP-Statuscode 409 zurückgegeben. Wenn der Benutzer nicht existiert, erstellt der Code einen neuen Benutzer mit den bereitgestellten Details, einschließlich eines Kennworts, das normalerweise aus Sicherheitsgründen verschlüsselt werden sollte. Es hat sich bewährt, Benutzerkennwörter in einem verschlüsselten Format zu speichern, um sie vor unbefugtem Zugriff zu schützen. Nachdem der neue Benutzer erstellt wurde, gibt der Code eine Erfolgsantwort im JSON-Format mit dem HTTP-Statuscode 200 (OK) zurück.

Request body

```
{
   "email": "anakin@gmail.com",
   "name": "anakin",
   "surname": " skywalker ",
   "password": "padme"
}
```

Antwort

```
{
   "id": 2,
   "email": " anakin@gmail.com",
   "name": "anakin",
   "surname": "skywalker",
   "password": "$2a$10$86i4LVG/FpO3n5Pujlv9Le88T1A3uUN7dBMBDZQEb5EkWP2697pK."
}
```

# SYSTEMTESTFÄLLE - US_REGISTRIERUNG

| | |
|---|---|
| **System Test Case ID Name** | S_TC_US_1 Test Create new user with POST-createUser service |
| **Preconditions** | • User web service is given:<br>  ○ Request: POST createUser(User user),<br>  ○ Response: A User with User attributes (Password is encrypted)<br>  • int id // unique <generated><br>  • String email<br>  • String name<br>  • String surname<br>  • String password<br>• App port number = (default) 8080<br>• The "users/register" endpoint is accessible.<br>• The password encoder is properly configured.<br>• The user service is properly configured and connected to a database.<br>• System is up and running! |
| **Test Steps** | 1. Send a POST request to the "/register" endpoint with a valid user object in the request body.<br>2. Verify that the response status code is 201 (Created).<br>3. Verify that the response body contains the saved user object.<br>4. Send another POST request to the "/register" endpoint with the same email address in the request body.<br>5. Verify that the response status code is 409 (Conflict).<br>6. Verify that the response body contains the message "User already exists". |
| **Post-Conditions** | 1. The user object is saved in the database. |

# SYSTEMTESTFÄLLE - US_REGISTRIERUNG

| | |
|---|---|
| **Test Data** | Patrick, Bateman, patrick-bateman@gmail.com, sigma |
| **Expected Result** | US User<br>• id = <generated><br>• US email = "patrick-bateman@gmail.com"<br>• US name = "Patrick"<br>• US surname = "Bateman"<br>• US password = "$2a$10$1nJrnduis3tPbCSyX7EccuT6I2Dac0O2s/G3/yo9GZkVWv.Q0cMZS" |
| **Actual Result** | US User<br>• id = <generated><br>• US email = "patrick-bateman@gmail.com"<br>• US name = "Patrick"<br>• US surname = "Bateman"<br>• US password = "$2a$10$1nJrnduis3tPbCSyX7EccuT6I2Dac0O2s/G3/yo9GZkVWv.Q0cMZS" |
| **Verdict (Pass/Fail)** | Pass |
| **Verified UC & Req. IDs** | US_Registrierung |

| | |
|---|---|
| **Component Integration Test Case ID Name** | CI_TC_US_1 Test provide User database interface |
| **Preconditions** | • Database Table “User” includes a row for<br>  • id = unique <generated><br>  • email<br>  • name<br>  • surname<br>  • password<br>• US Web service call is mocked! |
| **Test Steps** | 1. Retrieve a user from the database using the getUserByEmail() method.<br>2. Attempt to retrieve a non-existing user from the database using the getUserByEmail() method.<br>3. Create a new user in the database using the createUser() method.<br>4. Attempt to create a user that already exists in the database using the createUser() method.<br>5. Authenticate a user using the login() method.<br>6. Attempt to authenticate a user with an invalid email address using the login() method.<br>7. Attempt to authenticate a user with an invalid password using the login() method. |
| **Post-Conditions** | none |
| **Test Data** | "Patrick", "Bateman", "patrick-bateman@gmail.com", "sigma" |
| **Expected Result** | US User.name = "Patrick"<br>US User.surname ="Bateman"<br>US User.email = "patrick-bateman@gmail.com"<br>US User.password = "sigma" |
| **Actual Result** | US User.name = "Patrick"<br>US User.surname ="Bateman"<br>US User.email = "patrick-bateman@gmail.com"<br>US User.password = "sigma" |
| **Verdict (Pass/Fail)** | Pass |
| **Verified UC & Req. IDs** | US_Registrierung, US_Login, US_getUser |

# Price Tracker - Kundengeschichte

## Was macht diese Software?

**01** Ermöglichen Sie Benutzern, ein Konto unter Verwendung ihres Vor- und Nachnamens, ihrer E-Mail-Adresse und ihres Passworts zu erstellen.

**02** Ermöglichen Sie Benutzern, sich mit ihrer E-Mail-Adresse und ihrem Passwort in ihr Konto einzuloggen.

**03** Ermöglichen Sie Benutzern, Produkte, die sie verfolgen möchten, hinzuzufügen, indem sie URLs von beliebten Einkaufsseiten wie Hepsiburada.com, trendyol.com und amazon.com.tr eingeben.

**04** Zeigen Sie die Preise der verfolgten Produkte in einem grafischen Format an, um Benutzern zu ermöglichen, Preisentwicklungen im Laufe der Zeit zu visualisieren.

**05** Ermöglichen Sie Benutzern, Bedingungen für die von ihnen verfolgten Produkte festzulegen, z. B. eine Benachrichtigung zu erhalten, wenn der Preis unter einen bestimmten Betrag fällt.

**06** Lassen Sie das System diese Produkte im bestimmten Zeitpunkt überprüfen und die Preise aufzeichnen und vergleichen.

# Use Case - POST

Client

POST

-Wenn ein Benutzer ein neues Konto erstellt, wird eine POST-Anfrage gesendet, um eine neue Benutzerentität in der Datenbank zu erstellen.

-Wenn ein Benutzer ein neues Produkt zu seiner Watchlist hinzufügt, wird eine POST-Anfrage gesendet, um einen neuen Watchlist-Posten in der Datenbank zu erstellen.

-Wenn dieses Produkt nicht existiert, wird ein neues Produkt in der Datenbank erstellt. Der Preis wird regelmäßig überprüft und in die Preisliste-Entität in der Datenbank aufgenommen.

-Wenn ein Benutzer seinen Alarm aktualisiert, wird eine POST-Anfrage gesendet, um ihn in der Datenbank zu aktualisieren.




POST

Server

**Funktion**: Produkt hinzufügen
**Eingaben:** URL einer beliebten Einkaufsseite
**Verarbeitungsschritte:**

- Prüfe, ob die URL bereits in der Datenbank vorhanden ist.
- Wenn nicht, wird ein neues Produkt mit den Daten aus der URL in der Datenbank erstellt.
- Das Produkt wird zur Watchlist des Benutzers hinzugefügt.

**Ausgaben:** Das Produkt wird zur Watchlist des Benutzers hinzugefügt.

# WATCHLIST CONTROLLER

## createWatchlistForUser - POST


Es nimmt E-Mail-, Namens-, URL- und Bildparameter, erstellt ein neues Produktobjekt und prüft, ob ein Benutzer mit der angegebenen E-Mail-Adresse im System vorhanden ist. Wenn der Benutzer existiert, wird geprüft, ob ein Produkt mit der angegebenen URL existiert. Wenn es nicht vorhanden ist, erstellt es ein neues Product-Objekt und gibt eine Antwort mit einem vom watchlistService erstellten Watchlist-Objekt zurück. Wenn das Produkt bereits vorhanden ist, wird das vorhandene Produktobjekt zurückgegeben. Wenn der Benutzer nicht existiert, wird eine NOT_FOUND-Antwort zurückgegeben. Die Antwort wird in ein ResponseEntity-Objekt eingeschlossen, das den HTTP-Statuscode je nach Ergebnis der Operation auf CREATED oder NOT_FOUND setzt.

Request parameters
email=....
&name=...
&url=...
&image=...

Antwort

```
{
  "id": 6,
  "product": {
    "id": 3,
    "name": "Bargello ERKEK ....",
    "image": "https://cdn.dsmcdn.com/ty96/....jpg",
    "url": "https://www.trendyol.com/bargello/erkek-par..."
  },
  "alarm": null
}
```

**Watchlist**

- int id
- Product product
- Alarm alarm
- User user

**Product**

- int id
- String name
- String image
- String url
- List<Watchlist> watchlists
- List<Price> prices

# SYSTEMTESTFÄLLE - WS_PRODUKTHINZUFÜGEN

| | |
|---|---|
| **System Test Case ID Name** | S_TC_WS_1 Test Add product to user's watchlist with POST-createWatchlistForUser service |
| **Preconditions** | • Watchlist web service is given:<br>  ○ Request: POST createWatchlistForUser(String email, String name, String url, String image),<br>  ○ Response: A Watchlist (alarm is null)<br>  • int id // unique <generated><br>  • Product product(int id, String name, String image, String url)<br>  • Alarm alarm // null<br>• App port number = (default) 8080<br>• The "/watchlists/add-product" endpoint is accessible.<br>• The user service and product service are properly configured and connected to a database.<br>• There is an existing user in the database with a valid email address.<br>• System is up and running! |
| **Test Steps** | 1. Send a POST request to the "/add-product" endpoint with valid email, name, url, and image parameters in the request body.<br>2. Verify that the response status code is 201 (Created).<br>3. Verify that the response body contains the created watchlist object.<br>4. Send another POST request to the "/add-product" endpoint with an invalid email or url parameter in the request body.<br>5. Verify that the response status code is 404 (Not Found) or 409 (Conflict).<br>6. Verify that the response body contains the appropriate error message. |
| **Post-Conditions** | The watchlist object is created and associated with the user and product in the database. |

# SYSTEMTESTFÄLLE - WS_PRODUKTHINZUFÜGEN

| | |
|---|---|
| **Test Data** | patrick-bateman@gmail.com, "Puma ESS Cap Siyah Şapka", https://cdn.dsmcdn.com/ty315/product/media/images/20220201/23/40833697/15920472/2/2_org_zoom.jpg, "https://www.trendyol.com/puma/ess-cap-siyah-sapka-p-3806547" |
| **Expected Result** | WS Watchlist<br>• id = <generated><br>• Product = product(2, "Puma ESS Cap Siyah Şapka", "https://cdn.dsmcdn.com/ty315/product/media/images/20220201/23/40833697/15920472/2/2_org_zoom.jpg", "https://www.trendyol.com/puma/ess-cap-siyah-sapka-p-3806547")<br>• Alarm = null |
| **Actual Result** | WS Watchlist<br>• id = <generated><br>• Product = product(2, "Puma ESS Cap Siyah Şapka", "https://cdn.dsmcdn.com/ty315/product/media/images/20220201/23/40833697/15920472/2/2_org_zoom.jpg", "https://www.trendyol.com/puma/ess-cap-siyah-sapka-p-3806547")<br>• Alarm = null |
| **Verdict (Pass/Fail)** | Pass |
| **Verified UC & Req. IDs** | WS_produktHinzufügen |

| | |
|---|---|
| **Component Integration Test Case ID Name** | CI_TC_WS_1 Test provide Watchlist database interface |
| **Preconditions** | • Database Table “Watchlist” includes a row for<br>  • id = unique <generated><br>  • alarm_id<br>  • product_id<br>  • user_id<br>• WS Web service call is mocked! |
| **Test Steps** | **Next page |
| **Post-Conditions** | none |
| **Test Data** | 1, 1, 1 |
| **Expected Result** | WS Watchlist.alarm_id = 1<br>WS Watchlist.product_id = 1<br>WS Watchlist.user_id = 1 |
| **Actual Result** | WS Watchlist.alarm_id = 1<br>WS Watchlist.product_id = 1<br>WS Watchlist.user_id = 1 |
| **Verdict (Pass/Fail)** | Pass |
| **Verified UC & Req. IDs** | WS_produktHinzufügen, WS_EntferneProdukt, WS_ErstelleAlarm, WS_loadWatchlist, WS_EntferneAlarm |

| | |
|---|---|
| **Test Steps** | 1. Set up the test environment, including the necessary dependencies and configurations.<br>2. Prepare the required test data, such as user details, product information, and watchlist records.<br>3. Instantiate the WatchlistController class and initialize the required dependencies (WatchlistService, UserService, ProductService).<br>4. Invoke the "getWatchlistsForUser" method of the WatchlistController, passing a valid user ID as a path parameter.<br>5. Verify that the returned ResponseEntity contains the expected HTTP status code (FOUND) and the list of watchlists for the user.<br>6. Invoke the "createWatchlistForUser" method of the WatchlistController, providing the necessary parameters (email, name, URL, image).<br>7. Verify that the returned ResponseEntity contains the expected HTTP status code (CREATED) and the created watchlist object.<br>8. Invoke the "deleteWatchlist" method of the WatchlistController, passing a valid watchlist ID as a path parameter.<br>9. Verify that the returned ResponseEntity contains the expected HTTP status code (OK) and the success message indicating the deletion.<br>10. Invoke the "setAlarm" method of the WatchlistController, passing a valid Alarm object as the request body.<br>11. Verify that the returned ResponseEntity contains the expected HTTP status code (OK) and the ID of the created alarm.<br>12. Invoke the "deleteAlarm" method of the WatchlistController, passing a valid watchlist ID as a path parameter.<br>13. Verify that the returned ResponseEntity contains the expected HTTP status code (OK) and the success message indicating the alarm deletion. |

# Price Tracker - Kundengeschichte

## Was macht diese Software?

**01** Ermöglichen Sie Benutzern, ein Konto unter Verwendung ihres Vor- und Nachnamens, ihrer E-Mail-Adresse und ihres Passworts zu erstellen.

**02** Ermöglichen Sie Benutzern, sich mit ihrer E-Mail-Adresse und ihrem Passwort in ihr Konto einzuloggen.

**03** Ermöglichen Sie Benutzern, Produkte, die sie verfolgen möchten, hinzuzufügen, indem sie URLs von beliebten Einkaufsseiten wie Hepsiburada.com, trendyol.com und amazon.com.tr eingeben.

**04** Zeigen Sie die Preise der verfolgten Produkte in einem grafischen Format an, um Benutzern zu ermöglichen, Preisentwicklungen im Laufe der Zeit zu visualisieren.

**05** Ermöglichen Sie Benutzern, Bedingungen für die von ihnen verfolgten Produkte festzulegen, z. B. eine Benachrichtigung zu erhalten, wenn der Preis unter einen bestimmten Betrag fällt.

**06** Lassen Sie das System diese Produkte im bestimmten Zeitpunkt überprüfen und die Preise aufzeichnen und vergleichen.

# Use Case - GET

Server

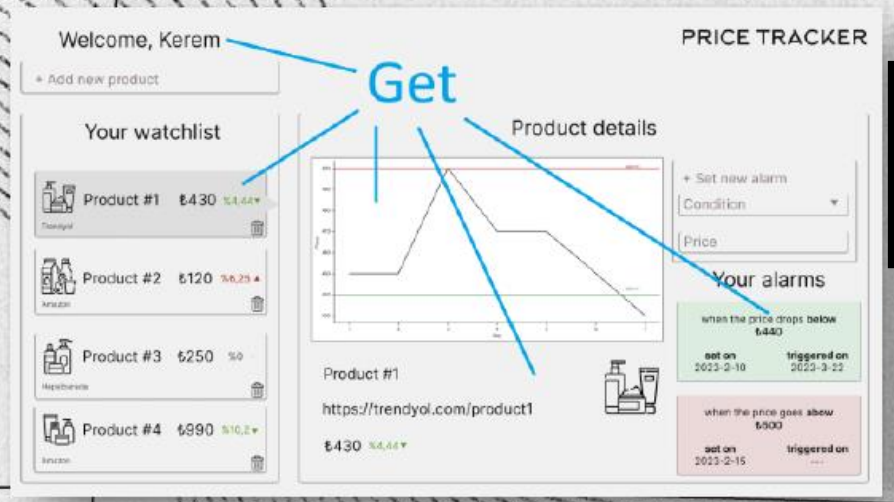
GET

- Wenn sich der Benutzer anmelden möchte, werden Passwort und E-Mail-Informationen mit GET überprüft.
- Beim Laden der Schnittstelle wird die Watchlist des Benutzers mit GET abgerufen.
- Beim Laden der Schnittstelle werden die Alarme der Produkte mit einer GET-Anfrage gebracht.

- Wenn die Schnittstelle geladen wird, wird die registrierte Preishistorie der Produkte mit einer GET-Anfrage gebracht.

GET

Client



**Funktion**: Preisverlauf anzeigen
**Eingaben**: Produkt aus der Watchlist
**Verarbeitungsschritte**:

- Wenn der Benutzer keine Watchlist hat, werden die in der Datenbank registrierten Opportunity-Produkte angezeigt.
- Wenn der Benutzer eine Watchlist hat Preisdaten des ausgewählten Produkts werden aus der Datenbank abgerufen.
- Die Preisentwicklung wird in einem grafischen Format angezeigt.

**Ausgaben**: Der Preisverlauf des ausgewählten Produkts wird angezeigt.

# PRICE CONTROLLER

## listPrices - GET


Es nimmt einen id-Parameter und verwendet die findAlarmById-Methode des alarmService-Objekts, um das Alarm-Objekt mit der angegebenen ID zu finden. Wenn das Alarm-Objekt gefunden wird, gibt es eine Antwort mit dem HTTP-Statuscode OK (200) und dem Alarm-Objekt im Antworttext zurück. Wenn das Alarm-Objekt nicht gefunden wird, gibt es eine Antwort mit dem HTTP-Statuscode NOT_FOUND zurück. Es nimmt einen product_id-Parameter und verwendet die getPricesForLast7Days-Methode des priceService-Objekts, um eine Liste von Price-Objekten für das angegebene Produkt für die letzten 7 Tage abzurufen. Er gibt dann eine Antwort mit einem HTTP-Statuscode von FOUND (302) zurück.

Antwort

```
[
  {
    "id": 35,
    "date": "2023-05-04",
    "price": 137.0
  },
  {
    "id": 34,
    "date": "2023-05-03",
    "price": 137.0
  }, ...
]
```

**Price**

- int id
- LocalDate date
- double price
- Product product

# SYSTEMTESTFÄLLE - PS_PREISVERLAUFANZEIGEN

| | |
|---|---|
| **System Test Case ID Name** | S_TC_PS_2 Test load last 7 days price data for selected product with GET-listPrices service |
| **Preconditions** | • Price web service is given:<br>  ○ Request: GET listPrices(int product_id),<br>  ○ Response: A list of Prices with attributes id, date and price<br>  • int id // unique <generated><br>  • String date<br>  • Double price<br>• App port number = (default) 8080<br>• The "/price/price-list/{product_id}" endpoint is accessible.<br>• The price service is properly configured and connected to a database.<br>• There is an existing product in the database with a valid ID.<br>• System is up and running! |
| **Test Steps** | 1. Send a GET request to the "/price/price-list/{product_id}" endpoint with a valid product ID as a path variable.<br>2. Verify that the response status code is 302 (Found).<br>3. Verify that the response body contains a list of prices for the last 7 days associated with the product. |
| **Post-Conditions** | The list of prices associated with the product is retrieved from the database. |

# SYSTEMTESTFÄLLE - PS_PREISVERLAUFANZEIGEN

| | |
|---|---|
| **Test Data** | None |
| **Expected Result** | PS Price<br>• id = <generated><br>• date = "2023-05-04"<br>• price = 127.0 |
| **Actual Result** | PS Price<br>• id = <generated><br>• date = "2023-05-04"<br>• price = 127.0 |
| **Verdict (Pass/Fail)** | Pass |
| **Verified UC & Req. IDs** | PS_preisverlaufAnzeigen |

| | |
|---|---|
| **Component Integration Test Case ID Name** | CI_TC_PS_1 Test provide Price database interface |
| **Preconditions** | • Database Table “Price” includes a row for<br>  • id = unique <generated><br>  • date<br>  • price<br>  • product_id<br>• PS Web service call is mocked! |
| **Test Steps** | 1. Call the startChecking() function in the PriceController class.<br>2. Verify that the returned ResponseEntity object has a status code of 200 OK.<br>3. Call the listPrices(int product_id) function in the PriceController class with a valid product ID.<br>4. Verify that the returned ResponseEntity object has a status code of 302 FOUND.<br>5. Verify that the returned ResponseEntity object contains a list of prices for the specified product within the last 7 days.<br>6. Call the updatePrice(int product_id) function in the PriceController class with a valid product ID.<br>7. Verify that the returned ResponseEntity object has a status code of 200 OK.<br>8. Verify that the PriceService class has updated the price for the specified product in the Price database. |
| **Post-Conditions** | The price-related operations should be executed successfully without any errors. |
| **Test Data** | ”2023-05-23”, 450, 1 |
| **Expected Result** | PS Price.date = ”2023-05-23”<br>PS Price.price = 450<br>PS Price.product_id = 1 |
| **Actual Result** | PS Price.date = ”2023-05-23”<br>PS Price.price = 450<br>PS Price.product_id = 1 |
| **Verdict (Pass/Fail)** | Pass |
| **Verified UC & Req. IDs** | PS_alarmFestlegen, PS_preisverlaufAnzeigen |