SYSTEMTEST SPEZIFIKATION PRICE TRACKER

Metin Kerem Öztürk, e200503057@stud.tau.edu.tr

Yavuz Selim Tunçer, e200503001@stud.tau.edu.tr

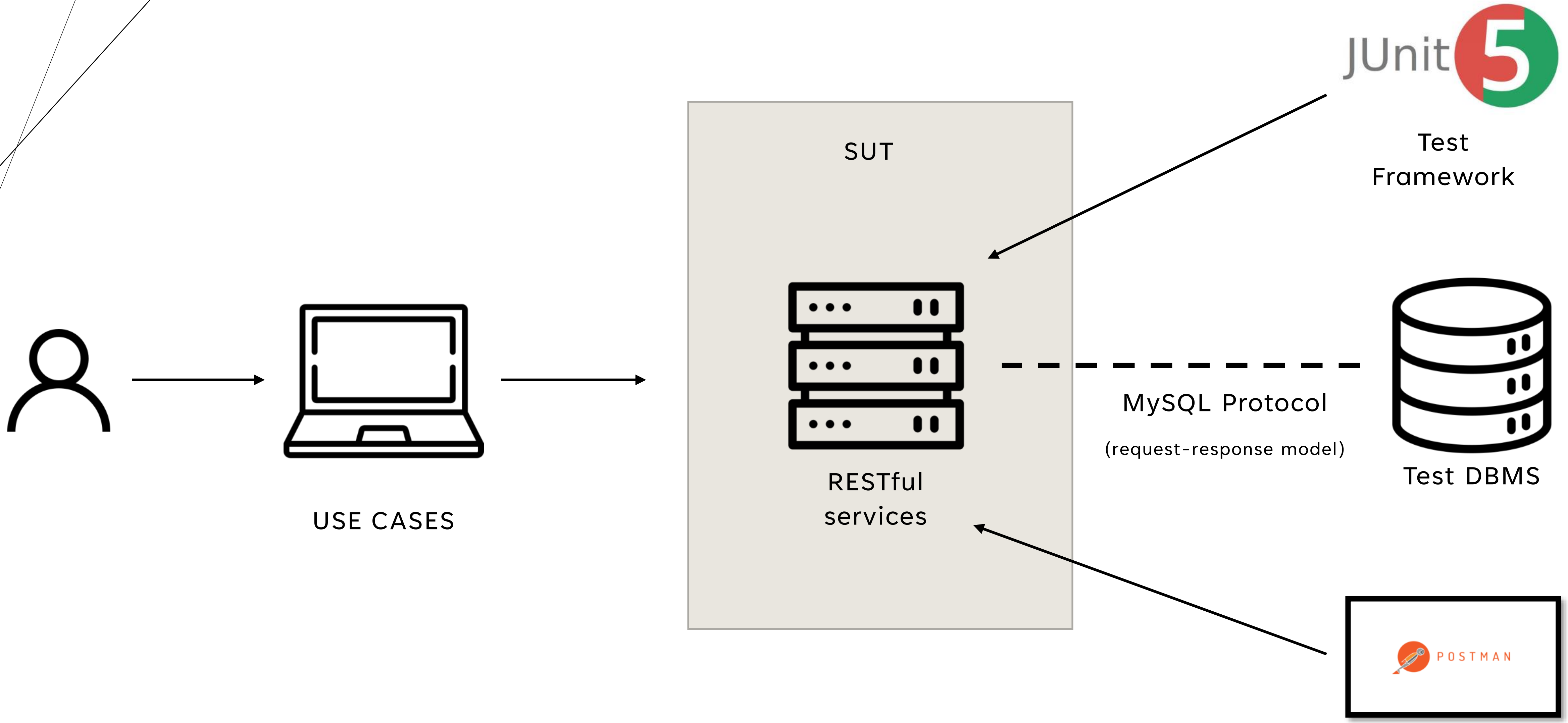
Stakeholder: DI. Ömer Karacan, omer.karacan@tau.edu.tr

TÜRK-ALMAN ÜNİVERSİTESİ
TÜRKISCH-DEUTSCHE UNIVERSITÄT

PRICE TRACKER

INHALTSVERZEICHNIS

SYSTEMÜBERBLICK



EINLEITUNG

Das Systemtestspezifikationsdokument umfasst eine breite Palette von Testszenarien und -fällen, um eine umfassende Abdeckung der Systemfunktionen sicherzustellen. Diese Tests untersuchen verschiedene Aspekte, darunter Eingabevalidierung, funktionale Arbeitsabläufe, Fehlerbehandlung, Leistung, Sicherheit und Datenbankkonnektivität. Um eine umfassende Bewertung des Systems zu gewährleisten, werden wir sowohl Postman als auch JUnit in unseren Testprozess integrieren. Postman wird sich hauptsächlich auf API-Tests konzentrieren, während JUnit zum Testen der Datenbankkonnektivität verwendet wird.

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: ○ Request: POST createUser(User user), ○ Response: A User with User attributes (Password is encrypted) • int id // unique <generated> • String email • String name • String surname • String password • App port number = (default) 8080 • The "users/register" endpoint is accessible. • The password encoder is properly configured. • The user service is properly configured and connected to a database. • 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. 2. Verify that the response status code is 201 (Created). 3. Verify that the response body contains the saved user object. 4. Send another POST request to the "/register" endpoint with the same email address in the request body. 5. Verify that the response status code is 409 (Conflict). 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 • id = <generated> • US email = "patrick-bateman@gmail.com" • US name = "Patrick" • US surname = "Bateman" • US password = "$2a$10$1nJrnduis3tPbCSyX7EccuT6I2Dac0O2s/G3/yo9GZkVWv.Q0cMZS"
Actual Result	US User • id = <generated> • US email = "patrick-bateman@gmail.com" • US name = "Patrick" • US surname = "Bateman" • US password = "$2a$10$1nJrnduis3tPbCSyX7EccuT6I2Dac0O2s/G3/yo9GZkVWv.Q0cMZS"
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	US_Registrierung

SYSTEMTESTFÄLLE - US_LOGIN

System Test Case ID Name	S_TC_US_2 Test Login to the user's account POST-login service
Preconditions	• User web service is given: ○ Request: POST login(String email, String password), ○ Response: A User with User attributes (Password is encrypted) • int id // unique <generated> • String email • String name • String surname • String password • App port number = (default) 8080 • The "users/login" endpoint is accessible. • The password encoder is properly configured. • The user service is properly configured and connected to a database. • There is an existing user in the database with a valid email and password. • System is up and running!
Test Steps	1. Send a POST request to the "/login" endpoint with valid email and password credentials in the request body. 2. Verify that the response status code is 200 (OK). 3. Verify that the response body contains the existing user object. 4. Send another POST request to the "/login" endpoint with an invalid email or password in the request body. 5. Verify that the response status code is 401 (Unauthorized). 6. Verify that the response body contains the message "Invalid email or password".
Post-Conditions	1. The user object is retrieved from the database using the email address.

SYSTEMTESTFÄLLE - US_LOGIN

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

SYSTEMTESTFÄLLE - US_GETUSER

System Test Case ID Name	S_TC_US_3 Test Fetch the user's details with GET-getUserByEmail service
Preconditions	• User web service is given: ○ Request: GET getUserByEmail(String email), ○ Response: A User with User attributes (Password is null) • int id // unique <generated> • String email • String name • String surname • String password // null • App port number = (default) 8080 • The "/users/details/{email}" endpoint is accessible. • The user service is properly configured and connected to a database. • There is an existing user in the database with a valid email address. • System is up and running!
Test Steps	1. Send a GET request to the "/details/{email}" endpoint with a valid email address as a path variable. 2. Verify that the response status code is 200 (OK). 3. Verify that the response body contains the user object with the specified email address. 4. Send another GET request to the "/details/{email}" endpoint with an invalid email address as a path variable. 5. Verify that the response status code is 404 (Not Found). 6. Verify that the response body contains the message "User not found!".
Post-Conditions	The user object is retrieved from the database using the email address.

SYSTEMTESTFÄLLE - US_GETUSER

Test Data	**None**
Expected Result	US User • id = <generated> • US email = "patrick-bateman@gmail.com" • US name = "Patrick" • US surname = "Bateman" • US password = null
Actual Result	US User • id = <generated> • US email = "patrick-bateman@gmail.com" • US name = "Patrick" • US surname = "Bateman" • US password = null
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	US_getUser

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: ○ Request: POST createWatchlistForUser(String email, String name, String url, String image), ○ Response: A Watchlist (alarm is null) • int id // unique <generated> • Product product(int id, String name, String image, String url) • Alarm alarm // null • App port number = (default) 8080 • The "/watchlists/add-product" endpoint is accessible. • The user service and product service are properly configured and connected to a database. • There is an existing user in the database with a valid email address. • 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. 2. Verify that the response status code is 201 (Created). 3. Verify that the response body contains the created watchlist object. 4. Send another POST request to the "/add-product" endpoint with an invalid email or url parameter in the request body. 5. Verify that the response status code is 404 (Not Found) or 409 (Conflict). 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 • id = <generated> • 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") • Alarm = null
Actual Result	WS Watchlist • id = <generated> • 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") • Alarm = null
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	WS_produktHinzufügen

SYSTEMTESTFÄLLE - WS_ENTFERNEPRODUKT

System Test Case ID Name	S_TC_WS_2 Test remove product from user’s watchlist with POST-deleteWatchlist service
Preconditions	• Watchlist web service is given: ○ Request: POST deleteWatchlist(int id), ○ Response: A response message • String "Product deleted." • App port number = (default) 8080 • The "/watchlists/remove/{watchlist_id}" endpoint is accessible. • The watchlist service is properly configured and connected to a database. • There is an existing watchlist in the database with a valid ID. • System is up and running!
Test Steps	1. Send a POST request to the "/watchlists/remove/{watchlist_id}" endpoint with a valid watchlist ID as a path variable. 2. Verify that the response status code is 200 (OK). 3. Verify that the response body contains the message "Product deleted."
Post-Conditions	1. The watchlist object is deleted from the database. 2. The watchlist object cannot be retrieved, updated, or deleted from the database.

SYSTEMTESTFÄLLE - WS_ENTFERNEPRODUKT

Test Data	None
Expected Result	WS Watchlist • Product deleted.
Actual Result	WS Watchlist • Product deleted.
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	WS_EntferneProdukt

SYSTEMTESTFÄLLE - WS_ERSTELLEALARM

System Test Case ID Name	S_TC_WS_3 Test create new alarm for user's watchlist with POST-setAlarm service
Preconditions	• Watchlist web service is given: ○ Request: POST setAlarm(int productId, int watchlist_id, String date_created, String condition, double target_price), ○ Response: An integer alarm id • int alarm id ○ Where date_created equals to today's date. • App port number = (default) 8080 • The "/watchlists/set-alarm" endpoint is accessible. • The watchlist service is properly configured and connected to a database. • There is an existing watchlist in the database with a valid ID. • System is up and running!
Test Steps	1. Send a POST request to the "/watchlists/set-alarm" endpoint with a valid alarm object in the request body. 2. Verify that the response status code is 200 (OK). 3. Verify that the response body contains the ID of the created alarm.
Post-Conditions	1. The alarm object is created and associated with the watchlist in the database.

SYSTEMTESTFÄLLE - WS_ERSTELLEALARM

Test Data	None
Expected Result	WS Watchlist • 14
Actual Result	WS Watchlist • 14
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	WS_ErstelleAlarm

SYSTEMTESTFÄLLE - WS_LOADWATCHLIST

System Test Case ID Name	S_TC_WS_4 Test List products details of user’s watchlist with GET-getWatchlistsForUser service
Preconditions	• Watchlist web service is given: ○ Request: GET getWatchlistsForUser(int id), ○ Response: A list of Watchlists with id, product and alarm details • int id // unique <generated> • Product product(int id, String name, String image, String url) • Alarm alarm(int id, int productId, int watchlist_id, String date_created, String condition, double target_price, String date_triggered) • App port number = (default) 8080 • There is an existing user in the database with a valid ID. • The watchlist service is properly configured and connected to a database. • The "/watchlists/load/{user_id}" endpoint is accessible. • System is up and running!
Test Steps	1. Send a GET request to the "/watchlists/load/{user_id}" endpoint with a valid user ID as a path variable. 2. Verify that the response status code is 302 (Found). 3. Verify that the response body contains a list of watchlists associated with the user. 4. Send another GET request to the "/load/{user_id}" endpoint with an invalid user ID as a path variable. 5. Verify that the response status code is 404 (Not Found).
Post-Conditions	The list of watchlists associated with the user is retrieved from the database.

SYSTEMTESTFÄLLE - WS_LOADWATCHLIST

Test Data	None
Expected Result	WS Watchlist • id = <generated> • 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") • Alarm = alarm(4, 2, 2, "2023-05-24", "BELOW_TARGET", 270.0, "2023-05-26")
Actual Result	WS Watchlist • id = <generated> • 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") • Alarm = alarm(4, 2, 2, "2023-05-24", "BELOW_TARGET", 270.0, "2023-05-26")
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	WS_loadWatchlist

SYSTEMTESTFÄLLE - WS_ENTFERNEALARM

System Test Case ID Name	S_TC_WS_5 Test remove alarm from user’s watchlist with POST-deleteAlarm service
Preconditions	• Watchlist web service is given: ○ Request: POST deleteAlarm(int watchlist_id), ○ Response: A response message • String "Alarm deleted!" • App port number = (default) 8080 • The "/watchlists/remove-alarm/{watchlist_id}" endpoint is accessible. • The watchlist service is properly configured and connected to a database. • There is an existing watchlist in the database with a valid ID. • System is up and running!
Test Steps	1. Send a POST request to the "/watchlists/remove-alarm/{watchlist_id}" endpoint with a valid watchlist ID as a path variable. 2. Verify that the response status code is 200 (OK). 3. Verify that the response body contains the message "Alarm deleted!".
Post-Conditions	1. The alarm associated with the watchlist is deleted from the database. 2. The alarm object cannot be retrieved, updated, or deleted from the database.

SYSTEMTESTFÄLLE - WS_ENTFERNEALARM

Test Data	None
Expected Result	WS Watchlist • Alarm deleted!
Actual Result	WS Watchlist • Alarm deleted!
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	WS_EntferneAlarm

SYSTEMTESTFÄLLE - PS_ALARMFESTLEGEN

System Test Case ID Name	S_TC_PS_1 Test start updating prices and checking alarm conditions with POST-startChecking service
Preconditions	• Price web service is given: ○ Request: POST startChecking(), ○ Response: A response message • String "Started" • App port number = (default) 8080 • The "/price/start" endpoint is accessible. • Amazon.com.tr, hepsiburada.com and trendyol.com are live. • The price service is properly configured and connected to a database. • System is up and running!
Test Steps	1. Send a POST request to the "/price/start" endpoint. 2. Verify that the response status code is 200 (OK). 3. Verify that the response body contains the message "Started". 4. Check the database to ensure that the prices have been updated.
Post-Conditions	The prices associated with the products are updated in the database.

SYSTEMTESTFÄLLE – PS_ALARMFESTLEGEN

Test Data	None
Expected Result	PS Price • Started
Actual Result	PS Price • Started
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	PS_alarmFestlegen

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: ○ Request: GET listPrices(int product_id), ○ Response: A list of Prices with attributes id, date and price • int id // unique <generated> • String date • Double price • App port number = (default) 8080 • The "/price/price-list/{product_id}" endpoint is accessible. • The price service is properly configured and connected to a database. • There is an existing product in the database with a valid ID. • 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. 2. Verify that the response status code is 302 (Found). 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 • id = <generated> • date = "2023-05-04" • price = 127.0
Actual Result	PS Price • id = <generated> • date = "2023-05-04" • price = 127.0
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	PS_preisverlaufAnzeigen

SYSTEMTESTFÄLLE - AS_LAODALARM

System Test Case ID Name	S_TC_AS_1 Test get details of selected alarm with GET-loadAlarm service
Preconditions	• Alarm web service is given: ○ Request: GET loadAlarm(int id), ○ Response: An Alarm with all attributes • int id // unique <generated> • int productId • int watchlist_id • String date_created • String condition • Double target_price • String date_triggered • App port number = (default) 8080 • The "/alarm/load-alarm/{id}" endpoint is accessible. • There is an existing alarm in the database with a valid ID. • The alarm service is properly configured and connected to a database. • System is up and running!
Test Steps	1. Send a GET request to the "/alarm/load-alarm/{id}" endpoint with a valid alarm ID as a path variable. 2. Verify that the response status code is 200 (OK). 3. Verify that the response body contains the alarm object associated with the ID. 4. Send another GET request to the "/alarm/load-alarm/{id}" endpoint with an invalid alarm ID as a path variable. 5. Verify that the response status code is 404 (Not Found).
Post-Conditions	The alarm object associated with the ID is retrieved from the database.

SYSTEMTESTFÄLLE – AS_LOADALARM

Test Data	6
Expected Result	AS Alarm • id = <generated> • productId = 1 • watchlist_id = 1 • date_created = "2023-05-10" • condition = "BELOW_TARGET" • target_price = 150.0 • date_triggered = null
Actual Result	AS Alarm • id = <generated> • productId = 1 • watchlist_id = 1 • date_created = "2023-05-10" • condition = "BELOW_TARGET" • target_price = 150.0 • date_triggered = null
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	AS_loadAlarm

Component Integration Test Case ID Name	CI_TC_US_1 Test provide User database interface
Preconditions	• Database Table “User” includes a row for • id = unique <generated> • email • name • surname • password • US Web service call is mocked!
Test Steps	1. Retrieve a user from the database using the getUserByEmail() method. 2. Attempt to retrieve a non-existing user from the database using the getUserByEmail() method. 3. Create a new user in the database using the createUser() method. 4. Attempt to create a user that already exists in the database using the createUser() method. 5. Authenticate a user using the login() method. 6. Attempt to authenticate a user with an invalid email address using the login() method. 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" US User.surname ="Bateman" US User.email = "patrick-bateman@gmail.com" US User.password = "sigma"
Actual Result	US User.name = "Patrick" US User.surname ="Bateman" US User.email = "patrick-bateman@gmail.com" US User.password = "sigma"
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	US_Registrierung, US_Login, US_getUser

Component Integration Test Case ID Name	CI_TC_WS_1 Test provide Watchlist database interface
Preconditions	• Database Table “Watchlist” includes a row for • id = unique <generated> • alarm_id • product_id • user_id • 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 WS Watchlist.product_id = 1 WS Watchlist.user_id = 1
Actual Result	WS Watchlist.alarm_id = 1 WS Watchlist.product_id = 1 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. 2. Prepare the required test data, such as user details, product information, and watchlist records. 3. Instantiate the WatchlistController class and initialize the required dependencies (WatchlistService, UserService, ProductService). 4. Invoke the "getWatchlistsForUser" method of the WatchlistController, passing a valid user ID as a path parameter. 5. Verify that the returned ResponseEntity contains the expected HTTP status code (FOUND) and the list of watchlists for the user. 6. Invoke the "createWatchlistForUser" method of the WatchlistController, providing the necessary parameters (email, name, URL, image). 7. Verify that the returned ResponseEntity contains the expected HTTP status code (CREATED) and the created watchlist object. 8. Invoke the "deleteWatchlist" method of the WatchlistController, passing a valid watchlist ID as a path parameter. 9. Verify that the returned ResponseEntity contains the expected HTTP status code (OK) and the success message indicating the deletion. 10. Invoke the "setAlarm" method of the WatchlistController, passing a valid Alarm object as the request body. 11. Verify that the returned ResponseEntity contains the expected HTTP status code (OK) and the ID of the created alarm. 12. Invoke the "deleteAlarm" method of the WatchlistController, passing a valid watchlist ID as a path parameter. 13. Verify that the returned ResponseEntity contains the expected HTTP status code (OK) and the success message indicating the alarm deletion.

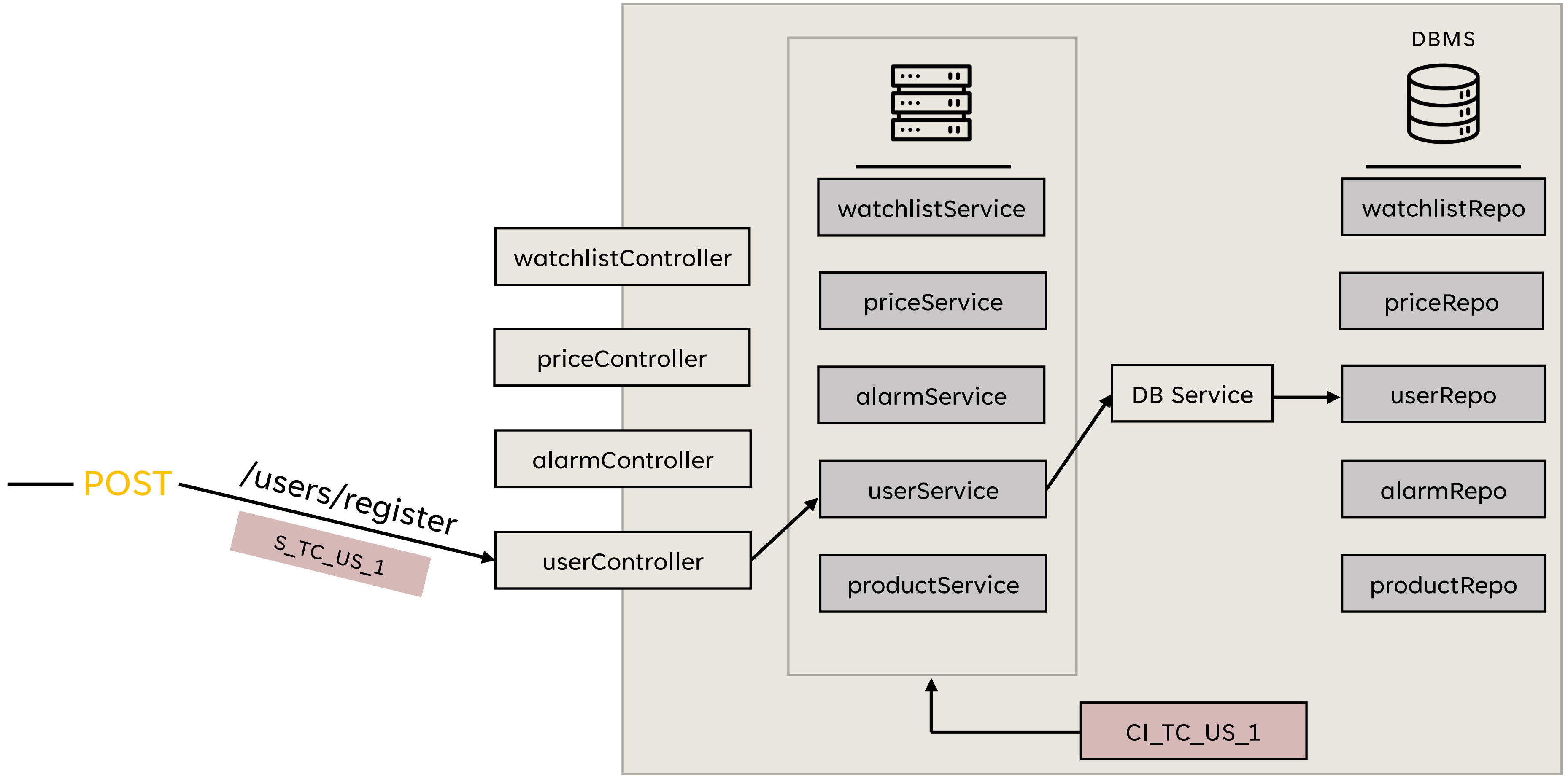
Component Integration Test Case ID Name	CI_TC_AS_1 Test provide Alarm database interface
Preconditions	• Database Table “Alarm” includes a row for • id = unique <generated> • condit(condition) • date_created • date_triggered • product_id • target_price • watchlist_id • AS Web service call is mocked!
Test Steps	1. Set up the test environment, including the necessary dependencies and configurations. 2. Prepare the required test data, such as alarm details and existing alarm records. 3. Invoke the "loadAlarm" method of the AlarmController, passing a valid alarm ID as a path parameter. 4. Verify that the returned ResponseEntity contains the expected HTTP status code (OK) and the alarm object. 5. If the alarm is found (not null), validate its attributes against the expected values. 6. If the alarm is not found (null), validate that the response contains the appropriate HTTP status code (Not Found).
Post-Conditions	The alarm-related operations should be executed successfully without any errors.
Test Data	"BELOW_TARGET", "2023-05-24", "2023-05-24", 2, 270, 2
Expected Result	AS Alarm.condit = "BELOW_TARGET" AS Alarm. date_created = "2023-05-24" AS Alarm. date_triggered = "2023-05-24" AS Alarm. product_id = 2 AS Alarm. target_price = 270 AS Alarm. watchlist_id = 2
Actual Result	AS Alarm.condit = "BELOW_TARGET" AS Alarm. date_created = "2023-05-24" AS Alarm. date_triggered = "2023-05-24" AS Alarm. product_id = 2 AS Alarm. target_price = 270 AS Alarm. watchlist_id = 2
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	AS_loadAlarm

Component Integration Test Case ID Name	CI_TC_PS_1 Test provide Price database interface
Preconditions	• Database Table "Price" includes a row for • id = unique <generated> • date • price • product_id • PS Web service call is mocked!
Test Steps	1. Call the startChecking() function in the PriceController class. 2. Verify that the returned ResponseEntity object has a status code of 200 OK. 3. Call the listPrices(int product_id) function in the PriceController class with a valid product ID. 4. Verify that the returned ResponseEntity object has a status code of 302 FOUND. 5. Verify that the returned ResponseEntity object contains a list of prices for the specified product within the last 7 days. 6. Call the updatePrice(int product_id) function in the PriceController class with a valid product ID. 7. Verify that the returned ResponseEntity object has a status code of 200 OK. 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" PS Price.price = 450 PS Price.product_id = 1
Actual Result	PS Price.date = "2023-05-23" PS Price.price = 450 PS Price.product_id = 1
Verdict (Pass/Fail)	Pass
Verified UC & Req. IDs	PS_alarmFestlegen, PS_preisverlaufAnzeigen

RÜCKVERFOLGBARKEIT DER ANFORDERUNGEN

Use Case Registrierung

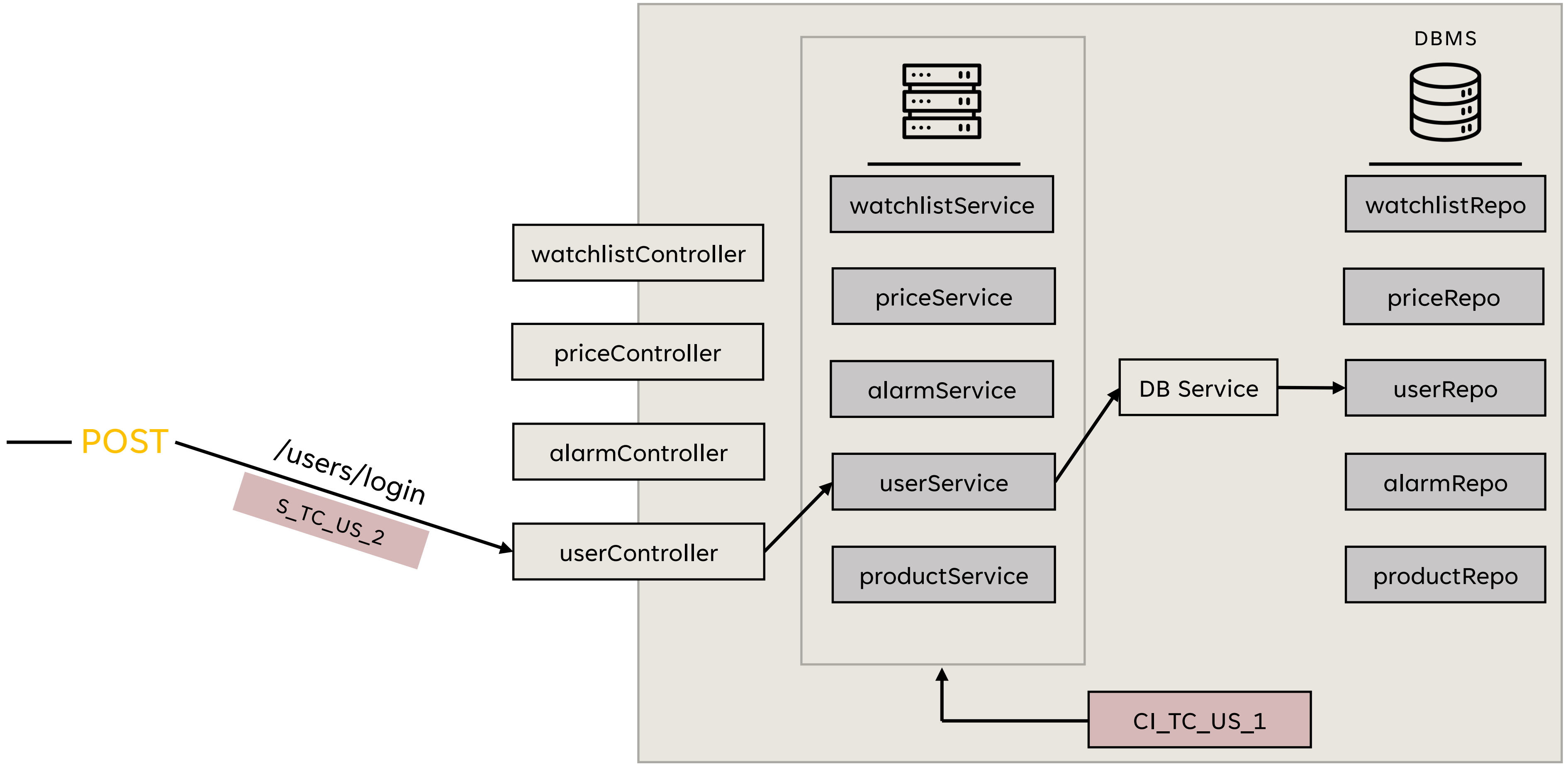
Funktion: Registrierung
Eingaben: Vorname, Nachname, E-Mail-Adresse, Passwort
Verarbeitungsschritte:
• Prüfe, ob die E-Mail-Adresse bereits in der Datenbank vorhanden ist.
• Wenn nicht, wird ein neuer Benutzer mit den eingegebenen Daten in der Datenbank erstellt.
• Der Benutzer wird zur Login-Seite weitergeleitet.
• 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.



RÜCKVERFOLGBARKEIT DER ANFORDERUNGEN

Use Case Login

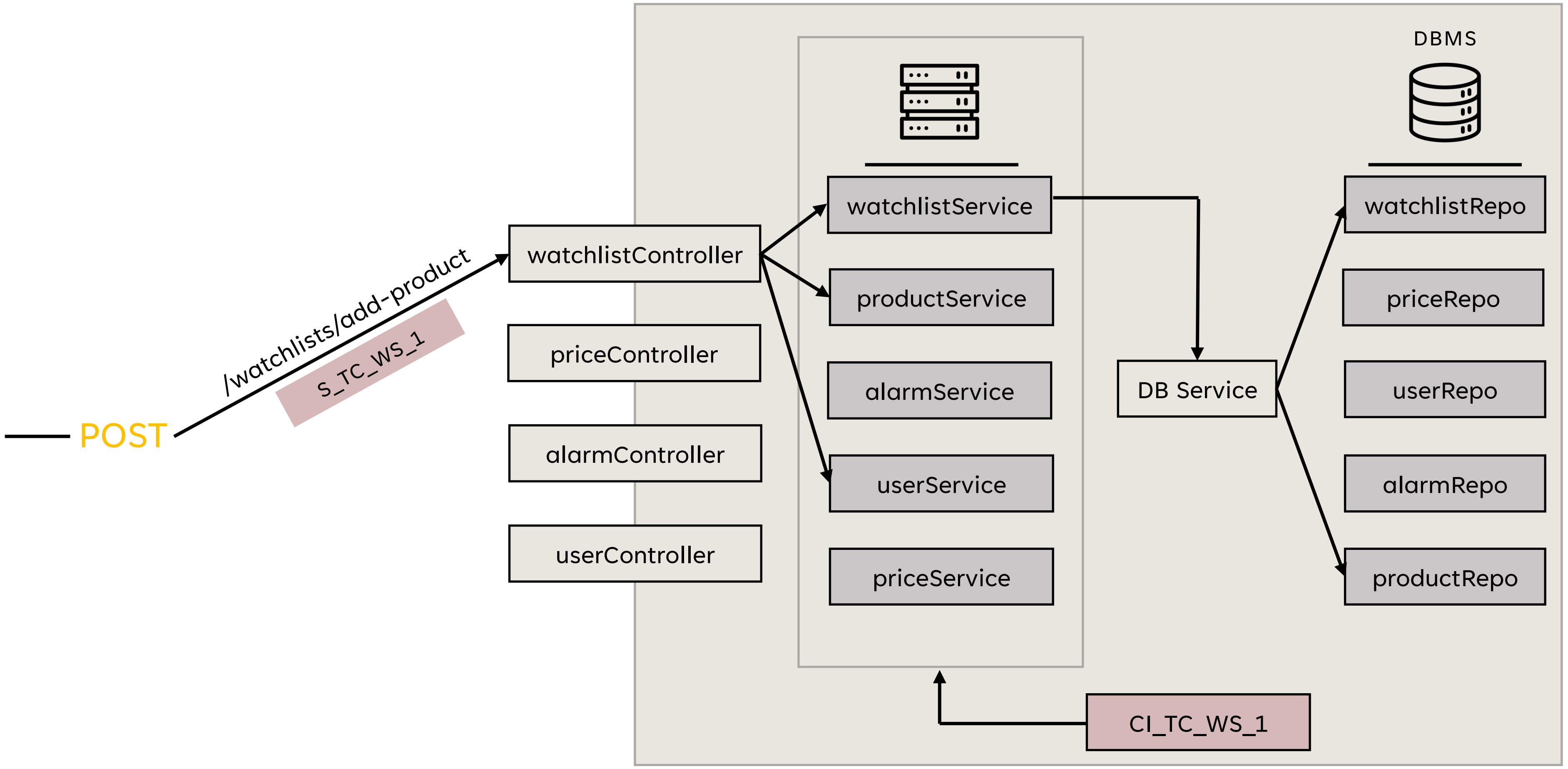
Funktion: Login
Eingaben: E-Mail-Adresse, Passwort
Verarbeitungsschritte:
• Prüfe, ob die E-Mail-Adresse und das Passwort in der Datenbank vorhanden sind.
• Wenn ja, wird der Benutzer zur Hauptseite weitergeleitet.
• Falls die E-Mail-Adresse oder das Passwort falsch sind, wird eine Fehlermeldung ausgegeben.
Ausgaben: Der Benutzer wird zur Hauptseite weitergeleitet oder eine Fehlermeldung wird angezeigt.



RÜCKVERFOLGBARKEIT DER ANFORDERUNGEN

Use Case Produkt hinzufügen

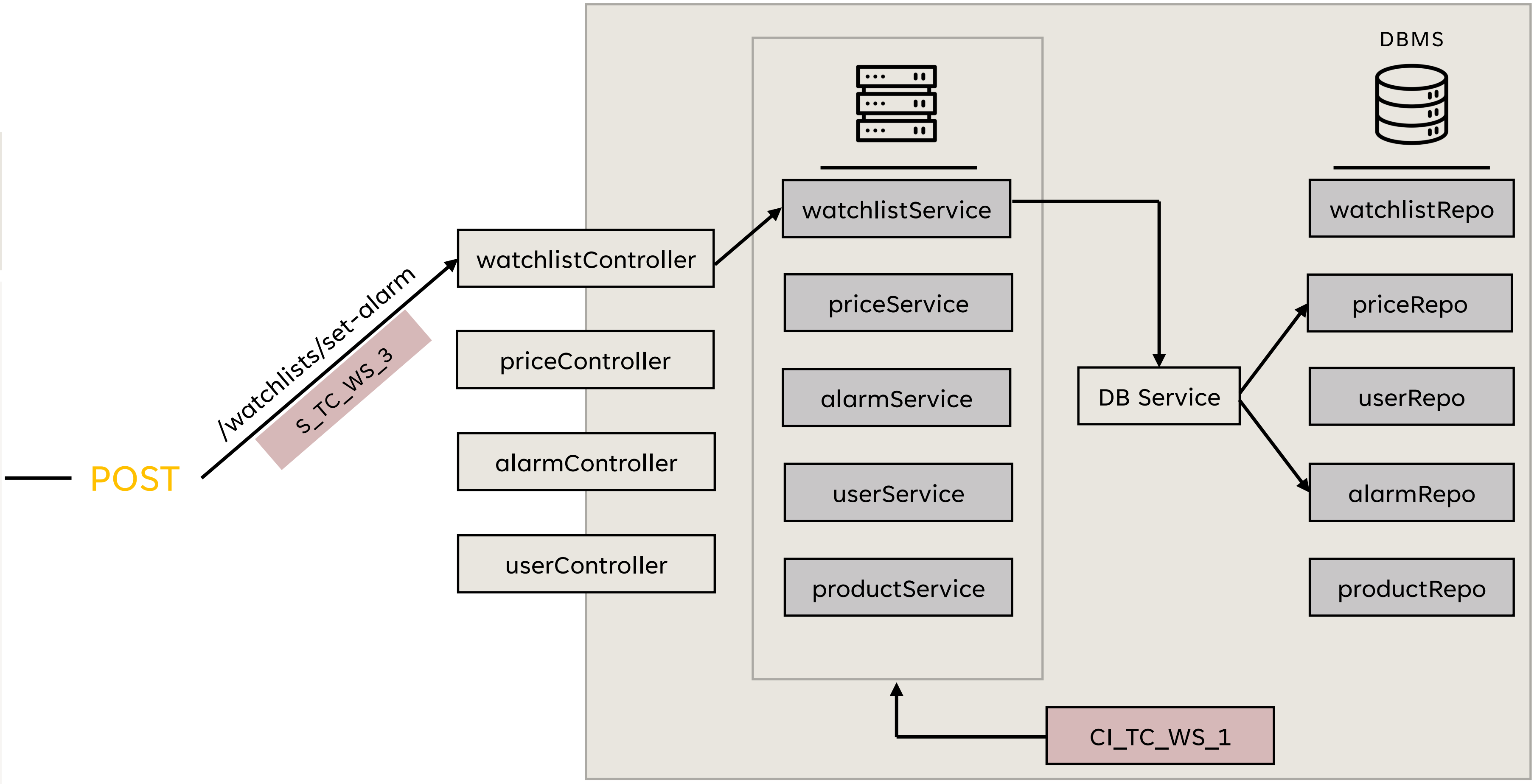
Funktion: Produkt hinzufügen
Eingaben: URL einer beliebten Einkaufsseite
Verarbeitungsschritte:
• Prüfe, ob die URL bereits in der Datenbank vorhanden ist.
• Wenn nicht, wird ein neues Produkt mit den Daten aus der URL in der Datenbank erstellt.
• Das Produkt wird zur Watchlist des Benutzers hinzugefügt.
Ausgaben: Das Produkt wird zur Watchlist des Benutzers hinzugefügt.



RÜCKVERFOLGBARKEIT DER ANFORDERUNGEN

Use Case Erstelle Alarm

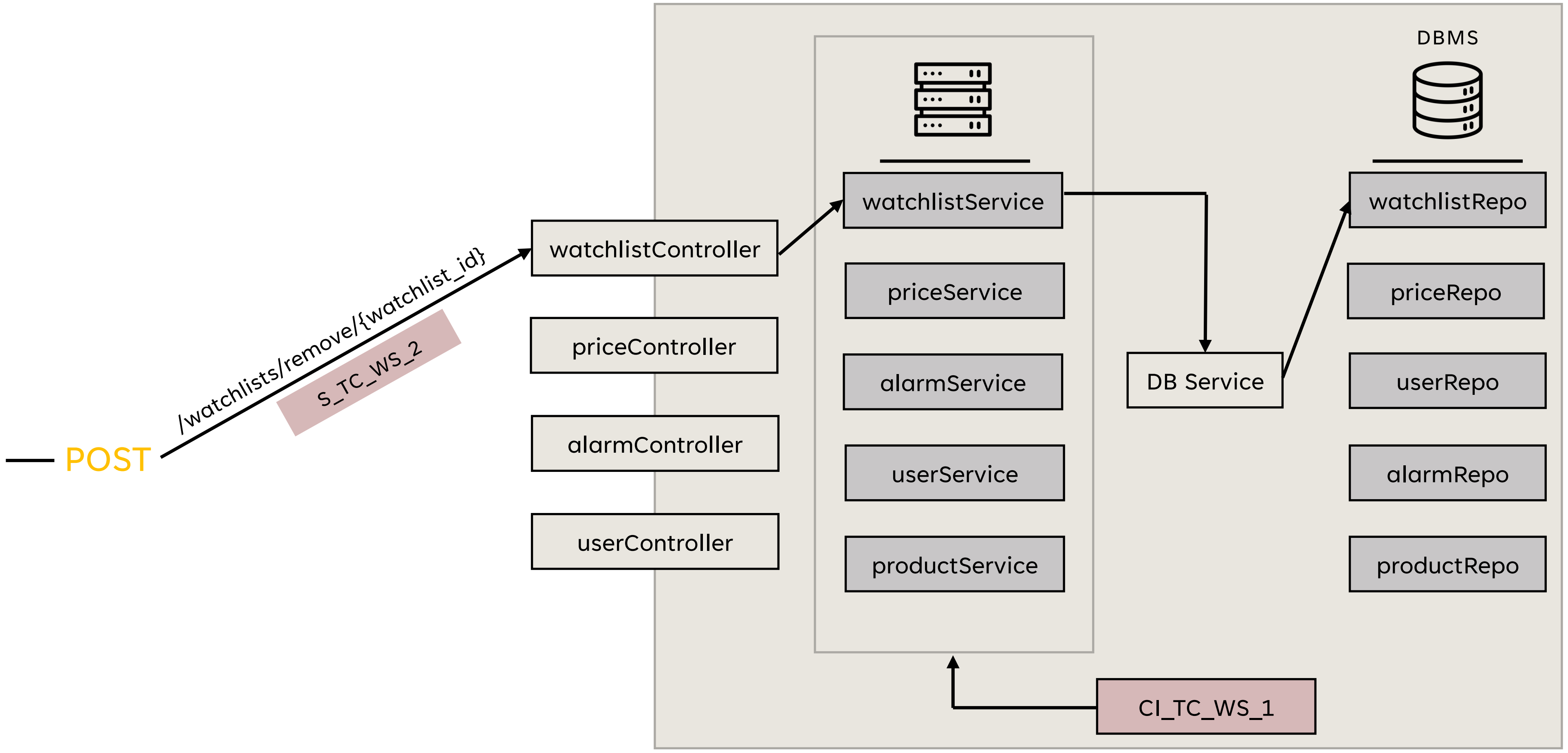
Funktion: Erstelle Alarm
Eingaben: Benutzer-ID, Produkt-ID, Bedingung und Ziel-Preis
Verarbeitungsschritte:
• Überprüfe, ob der Benutzer bereits einen Alarm für das Produkt und die Bedingung erstellt hat
• Füge einen neuen Alarm hinzu, falls keine Übereinstimmungen gefunden wurden, und speichere die Alarmdaten in der Datenbank
Ausgaben: Bestätigungsmeldung, dass der Alarm erfolgreich erstellt wurde, oder Fehlermeldung, wenn der Benutzer bereits einen Alarm für das Produkt und die Bedingung hat oder der Benutzer oder das Produkt nicht existieren.



RÜCKVERFOLGBARKEIT DER ANFORDERUNGEN

Use Case Entferne Produkt aus der Watchlist

Funktion: Entferne Produkt aus der Watchlist
Eingaben: Benutzer-ID und Produkt-ID
Verarbeitungsschritte:
• Überprüfe, ob der Benutzer existiert und das Produkt auf seiner Watchlist hat
• Entferne das Produkt aus der Watchlist des Benutzers
Ausgaben: Bestätigungsmeldung, dass das Produkt erfolgreich aus der Watchlist entfernt wurde, oder Fehlermeldung, wenn der Benutzer das Produkt nicht haben.

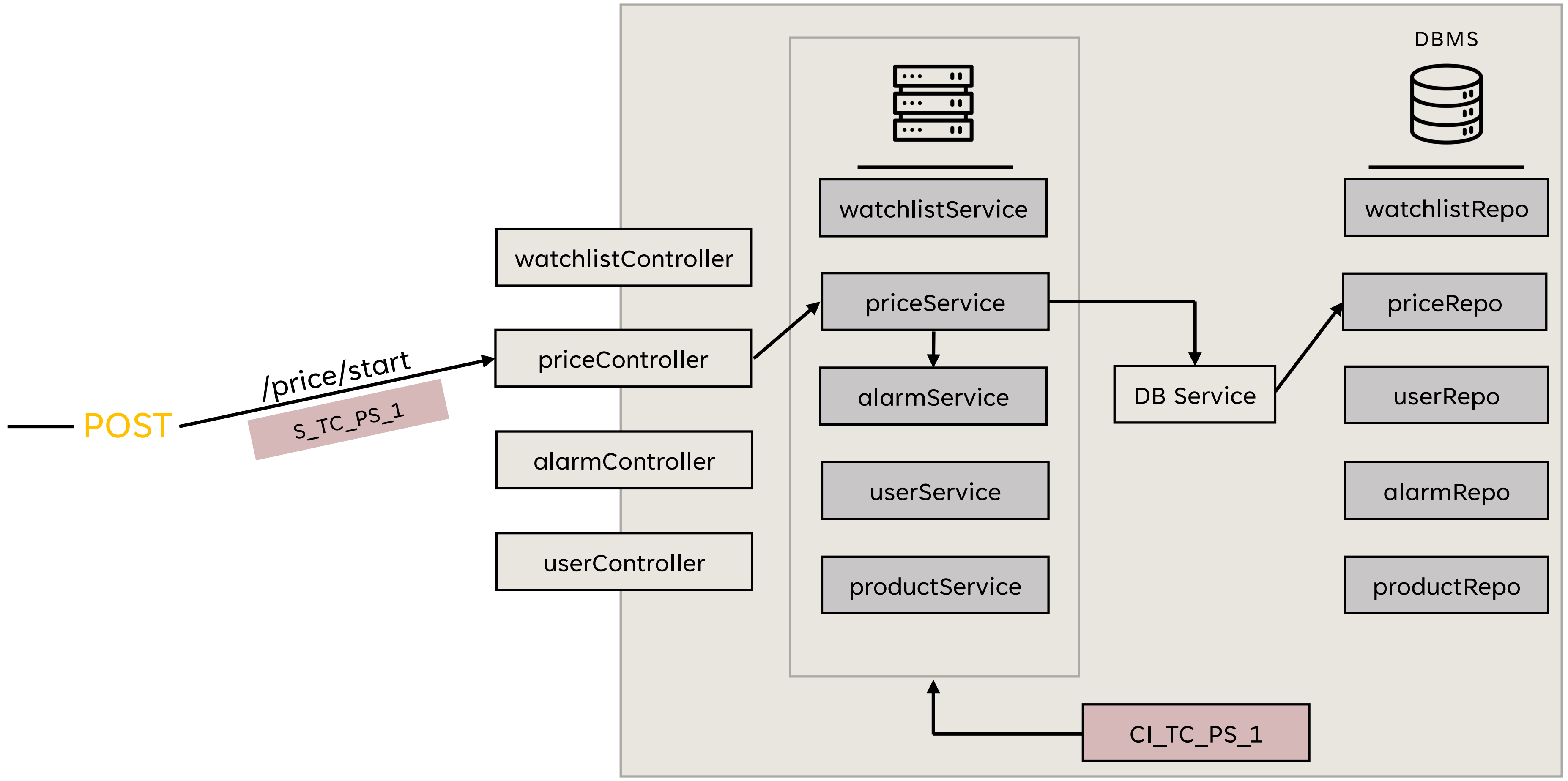


RÜCKVERFOLGBARKEIT DER ANFORDERUNGEN

Use Case Alarm festlegen

Funktion: Alarm festlegen
Eingabe: Benutzer-ID, Produkt-ID, Preislimit
Verarbeitungsschritte:
- Der Alarm wird für das ausgewählte Produkt mit dem angegebenen Preislimit in der Datenbank gespeichert.
- Das System überprüft regelmäßig den Preis des Produkts und sendet eine E-Mail an den Benutzer, wenn der Preis unter das Limit fällt.

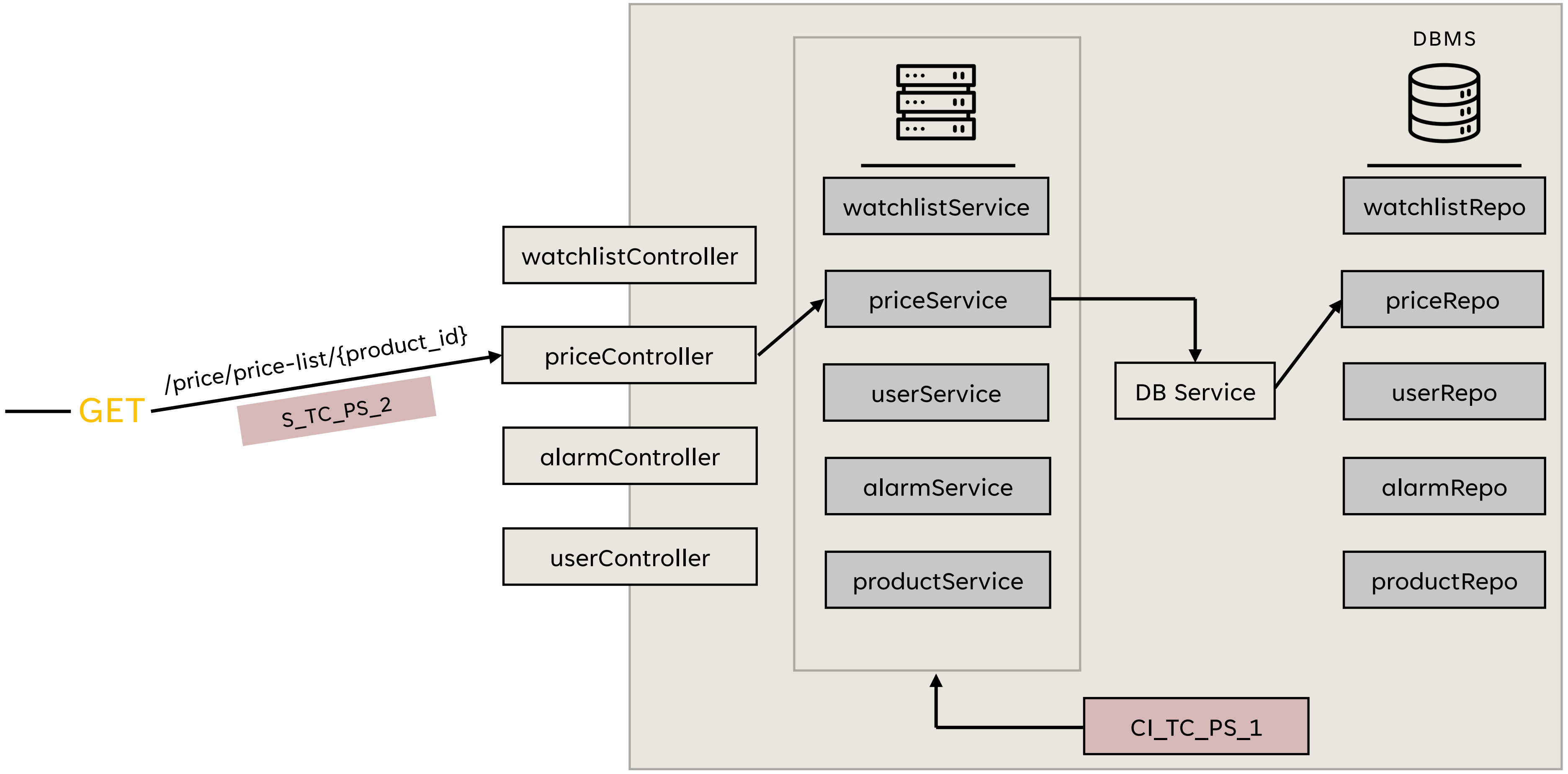
Ausgaben: Der Alarm wird erfolgreich festgelegt und der Benutzer wird per E-Mail benachrichtigt, wenn das Preislimit unterschritten wird.



RÜCKVERFOLGBARKEIT DER ANFORDERUNGEN

Use Case Preisverlauf anzeigen

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.


GET
/price/price-list/{product_id}
S_TC_PS_2
watchlistController
priceController
alarmController
userController
watchlistService
priceService
userService
alarmService
productService
DB Service
DBMS
watchlistRepo
priceRepo
userRepo
alarmRepo
productRepo
CI_TC_PS_1