BIRDWATCH

The Diversity of Life from Space

D6.4 Technical Report on Testing

Deliverable for the Horizon Europe Project BirdWatch

Version 1.0

Funded by the European Union

Legal Disclaimer

This document reflects only the views of the author(s). Neither the European Global Navigation Satellite Systems Agency (EUSPA) nor the European Commission is in any way responsible for any use that may be made of the information it contains. The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The below referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.
This document and the information contained within may not be copied, used, or disclosed, entirely or partially, outside of the BirdWatch consortium without prior permission of the project partners in written form.
© **2026** by BirdWatch Consortium.

Funded by
the European Union

Document Information

GA Number	101082634		**Type of Action**	Horizon-IA
Full Title	BirdWatch - a Copernicus-based service for the improvement of habitat suitability of farmland birds via satellite-enabled monitoring, evaluation and optimisation of CAP greening measures			
Project Acronym	BirdWatch			
Start Date	February 1st, 2023		**Duration**	36 Months
Project URL	https://birdwatch-europe.org/			
Deliverable	D6.4 Technical Report on Testing			
Work Package	WP6			
Project Month of Delivery	**Contractual**	M33	**Actual**	M36(+1)
Nature	R	**Dissemination Level**		PU
Lead Beneficiary	LUP			
Responsible Author	Ursula Plötz, LUP			
Contributions from	Laura Panner, LUP Nastasja Scholz, LUP			

Funded by
the European Union

History of Changes

Version	Issue Date	Stage	Description	Comments	Contributor
1.0	06.02.2026	Draft	First draft		Laura Panner Ursula Plötz

Funded by
the European Union

Funded by
the European Union

Introduction

BirdWatch's aim is to provide an EU-wide service supporting the monitoring and improvement of farmland habitat suitability for bird species which breed or forage on agricultural land.

The BirdWatch service will consist of a geospatial data-based monitoring service which evaluates the habitat suitability of farmland parcels for specific bird species as well as of an optimisation workflow, serving as a decision-support for the identification of appropriate policy-supported agri-environmental measures.

This deliverable summarises the consortium's testing activities and subsequent updates to the platform to increase performance and security, a crucial step to prepare the platform for the upcoming launch.

More specifically, the deliverable

- describes the exploration of potential platform vulnerabilities
- reflects on the potential ways to exploit these vulnerabilities
- describes the measures to reduce these vulnerabilities
- summarises identified issues with layer and database performance
- lists the measures to increase layer and database performance
- describes the platform tests conducted within the consortium before the demonstration activities started
- summarises the changes that were implemented in response to the consortium's feedback
- briefly reflects on value of the feedback from the stakeholder demonstrations

Funded by
the European Union

1 Internal Testing

1.1 Security Tests

In March and October 2025, the consortium partner National Paying Agency (NPA) conducted security tests, the results of which were subsequently shared with LUP.
Since no information on their methodology to conduct the security tests is available, no statement can be made regarding this.

To verify the effectiveness of the measures implemented based on the test results, manual API requests were performed using *Postman*[1]. *Postman* allowed us to validate the correct functionality and security of the API endpoints (also see Appendix).

1.1.1 Test results

Cross-site scripting

A potential vulnerability in the area of *Reflected Cross-Site Scripting* (XSS) was identified. This vulnerability could allow malicious script code to be injected via manipulated URL parameters and reflected directly in the web application's response without sufficient validation or sanitization.

SQL injection

A potential vulnerability related to *SQL injection* was identified. It appears that user-controlled inputs are being incorporated into SQL queries without sufficient filtering. This was indicated by error messages triggered by special characters, as well as measurable delays in server responses when using database functions like `pg_sleep`. As a result, an attacker might be able to manipulate the query structure and execute unauthorized SQL commands on the PostgreSQL database.

Ruby code injection

A potential code injection vulnerability was identified, where user-controlled inputs appear to be incorporated into dynamically executed Ruby code without sufficient validation. This was indicated by a measurable delay in server response after injecting specific inputs, suggesting that the injected code was executed.

[1] https://www.postman.com/

Funded by
the European Union

Python code injection

A potential code injection vulnerability was identified, where user-controlled inputs appear to be embedded in dynamically executed Python code without sufficient validation. This was indicated by a measurable delay in server response after injecting specific inputs, suggesting that the injected code was executed.

OS command injection

A potential *OS command injection* vulnerability was identified, where user-controlled inputs appear to be incorporated into operating system commands without sufficient validation. This was indicated by a measurable delay in server response after injecting special shell characters and a command like `ping`, suggesting that the injected command was executed.

Cross-Site Request Forgery

A potential *Cross-Site Request Forgery* (CSRF) vulnerability was identified, where an attacker could trick an authenticated user into unknowingly performing unintended actions within the web application.

Web server without HSTS / TLS cookie without secure flag set

A potential vulnerability was identified where the web application does not consistently enforce encrypted connections via *SSL/TLS.* This could allow an attacker to manipulate network traffic and downgrade HTTPS connections to unencrypted HTTP connections.

Vulnerable Bootstrap library version

It was found that the web application uses *Bootstrap* version 4.6.2, which has documented security vulnerabilities.

Client-side HTTP Parameter Pollution "HPP"

A potential *URL injection/parameter* manipulation vulnerability was identified, where user-controlled inputs appear to be incorporated into URL query strings without sufficient filtering and are reflected unchanged in the HTML response.

XML injection

A potential *XML/SOAP injection* vulnerability was identified, where user-controlled inputs may be unsafely embedded in server-side processed XML documents or SOAP messages.

TLS certificate issue

Several potential issues were identified with the *TLS certificate* used on the server, including:

- Missing hostname verification
- Untrusted or expired certificate
- Expired certificate chain

1.1.2 Measures

Cross-site scripting

- **X-Content-Type-Options: nosniff** Prevents the browser from interpreting content differently than specified (e.g., MIME type sniffing).
- **Content-Security-Policy:** Restricts which script sources are allowed to execute (e.g., only defined domains) and can significantly reduce the exploitability of XSS attacks.
- **Implemented Input Filter:** Inputs are sanitized/normalized server-side, which can make it harder to inject simple payloads.

SQL injection

Prepared statements were implemented to prevent SQL injection attacks. They strictly separate user input from SQL query processing, making it significantly more difficult to manipulate the query structure or execute unauthorized SQL commands on a PostgreSQL database.

Ruby code injection

- **Type Safety:** Only explicitly allow parameters and convert them to their expected types.
- **Escaping:** Special characters are converted so that they are no longer interpreted as code, but only as plain text.
- **Implemented Input Filter:** Inputs are sanitized/normalized server-side, which can make simple payloads more difficult to execute.

Python code injection

- **Type Safety:** Only explicitly allow parameters and convert them to their expected types.
- **Avoiding `eval()` and `exec()`:** Dynamic code execution using user-controlled inputs should be completely avoided.

- **Implemented Input Filter:** Inputs are sanitized/normalized server-side, which can make simple payloads more difficult to execute.

OS command injection

- **Type Safety:** Only explicitly allow parameters and convert them to their expected types.
- **Implemented Input Filter:** Inputs are sanitized/normalized server-side, which can make simple payloads more difficult to execute.

Cross-Site Request Forgery

- **CSRF Tokens:** Security-sensitive requests are only executed if they originate from the legitimate application and are not triggered by manipulated external websites.
- **Cookie Parameters** `SameSite=Lax`: Reduces the risk of CSRF attacks by restricting the automatic sending of authentication cookies in cross-site requests by the browser.
- **Header: Referrer-Policy = strict-origin-when-cross-origin:** Reduces the amount of referrer information in cross-site requests by transmitting only the origin, thereby limiting potential information leaks and enhancing overall security.

Web server without HSTS / TLS cookie without secure flag set

- **Secure Session Cookies:** Session cookies are marked with the Secure flag, ensuring they are only transmitted over encrypted HTTPS connections and cannot be exposed during unencrypted HTTP transmission
- **HSTS (HTTP Strict Transport Security):** The Strict-Transport-Security header is implemented to force browsers to use HTTPS exclusively, making downgrade attacks more difficult.
- **HTTPS as the Default for Internal Connections:** Using https:// as the default for generating internal links promotes the use of encrypted connections and reduces unintended HTTP access.

Vulnerable Bootstrap library version

The project was migrated to *Vite*[2], and all libraries—including *Bootstrap*—were updated to their latest versions. This addresses known vulnerabilities and enhances the security of the components in use.

[2] https://vite.dev/

Funded by
the European Union

Client-side HTTP Parameter Pollution "HPP"

- **Type Safety:** Only explicitly allow parameters and convert them to their expected types.
- **Implemented Input Filter:** Inputs are sanitized/normalized server-side, which can make simple payloads more difficult to execute.
- **Escaping:** Special characters are converted so that they are no longer interpreted as code, but only as plain text.

XML injection

- **Type Safety:** Only explicitly allow parameters and convert them to their expected types.
- **Implemented Input Filter:** Inputs are sanitized/normalized server-side, which can make simple payloads more difficult to execute.
- **Prepared Statements**: User inputs are processed strictly separate from the SQL query structure, significantly reducing the risk of database query manipulation.

1.1.3 Result

The implemented measures comprehensively address all identified security vulnerabilities and significantly reduce the associated risks. Particularly effective are the following:

Prepared Statements effectively prevent SQL injection by strictly separating user input from SQL query logic. The introduction of HSTS (HTTP Strict Transport Security) and Secure Cookies ensures that all communications are encrypted, preventing data transmission over insecure channels.

Another key element of the security strategy is the use of server-side input filters combined with strict type safety. These measures effectively mitigate code and command injection risks by ensuring that user inputs are always validated, sanitized, and processed in their intended format. To counter Cross-Site Request Forgery (CSRF), CSRF tokens have been implemented, guaranteeing that security-sensitive actions can only be triggered from within the legitimate application.

Overall, the combination of technical solutions—such as those mentioned above—and organizational processes results in a substantially enhanced security level across the systems. These steps not only protect against the specific threats identified but also strengthen resilience against future security risks.

1.2 Layer Performance Tests

Multiple tests were conducted to measure the performance of pre-generated and on-the-fly generated layer tiles. These tests were performed across all project regions at various zoom levels.

1.2.1 Test Results

Issues were identified with on-the-fly tile generation at low zoom levels (zoomed out). Some tiles failed to generate and display due to timeouts while other requests took over 60 seconds to complete. The tests were executed using console log outputs in the browser. The testing code is still present in the application and can be re-run at any time by uncommenting the relevant sections.

1.2.2 Measures

- **Transition from WMS (on-the-fly generation) to WMTS (pre-generated) Layers**
- **Fixed Cached and Rendered Zoom Levels**: Layers are only displayed between zoom levels 6 and 15, with pre-rendering for zoom levels 6–14.
- **Restricted Layer Requests in the Frontend:** Layer requests are now limited to the project area only.
- **Layer Setting Adjustments:** updateWhileAnimating = false updateWhileInteracting = false These changes reduce unnecessary requests and improve performance.

1.2.3 Result

Overall, the migration to WMTS, combined with the accompanying optimizations, has resulted in a significantly faster and more stable mapping service. Load times for map tiles have been reduced from sometimes over 60 seconds to under 1 second, while the error rate has been virtually eliminated. The average response time for most layers now lies in the low millisecond range, significantly enhancing user-friendliness and efficiency.

These improvements ensure a seamless and reliable user experience, sustainably increasing the quality of the application.

Funded by
the European Union

1.3 Database Performance Tests

Multiple tests were conducted to evaluate database performance. These included frequently used queries such as parcel information retrieval, geometry comparison queries, and data aggregation queries.

1.3.1 Test Results

Significant potential for optimizing query execution times was identified in several areas. On average, parcel information queries took 10 seconds, but could exceed 20 seconds, while queries involving data aggregation reached up to 35 seconds.

Note: The parcel IDs used in the detailed breakdown of these tests (see the performance report in Deliverable D6.3) may no longer be in use or may belong to different regions, as the underlying data and some query implementations (e.g., regional filtering) have evolved over time.

GiST Index Impact Assessment

To measure the impact of the GiST index, you can:

1. Temporarily drop the index:
   ```
   DROP INDEX index_name;
   ```
2. Conduct performance measurements.
3. Recreate the index:
   ```
   REATE INDEX index_name ON table_name USING GIST (geom);
   ```
4. Perform a comparative measurement to evaluate the difference.

This approach will help quantify the index's effect on query performance.

1.3.2 Measures

- **GiST Index for All Geometry Tables:** Ensure a GiST index is applied to all tables containing geometric data to accelerate spatial queries.
- **Pre-Aggregate Data**: Pre-calculate data aggregations and query only the results to reduce runtime computation.

Funded by
the European Union

- **Refactor Parcel Information Queries:** Optimize the logic and structure of parcel information queries for improved efficiency.
- **Table Maintenance with `VACUUM ANALYZE`:** Clean up tables and update statistical information for the query planner to enhance performance.
- **Reorganize Table Storage with `CLUSTER`:** Use `CLUSTER` to physically reorder tables based on an index, reducing access times and improving query performance.
- **Region-Specific Query Filtering:** Incorporate regional constraints in queries to limit data processing to relevant areas only.

1.3.3 Result

The implemented optimizations have fundamentally improved the performance of database queries in the application. By introducing a GiST index, query speeds were increased 1,000-fold, resulting in a massive acceleration of data access.

Another critical step was the pre-aggregation of data and its storage in incremental aggregate tables. This reduced response times from several seconds to milliseconds, relieving the database of computationally intensive real-time calculations and ensuring stable system performance, even under high loads. Resources are now used more efficiently, further enhancing the overall performance of the system.

These measures result in a significantly more responsive application, enabling a smooth and efficient user experience. Users benefit from near-instant data queries, which significantly boosts productivity and satisfaction in daily operations.

1.4 Login Performance Tests

Performance tests confirmed stable and fast login times. On September 10, 2025, 10 successful logins were evaluated (100% success rate), with a median duration of ~392 ms (P95 ~415 ms).

The majority of the time was spent on authentication, while the database update (last_login) contributed negligibly. Overall, the sample demonstrates a highly performant and reliable login process.

Funded by
the European Union

2 Consortium-internal Tests

After each development round, an internal test (carried out by LUP) was first performed to ensure direct functionality and basic usability requirements.

The starting point for a test run is the defined main user flow (see D6.2 - First implementation of web-based platform) and the feature specifications (see D2.4 - User and System Requirements Specification).

All functional starting points were always addressed in a structured manner and each option that could be selected at user decision points were completed. All resulting bugs and tasks were recorded in an internal bug and issue tracking system and followed up.

This was followed by two test rounds (March and September 2025) involving the project consortium in which comprehensive user tests were conducted in our four project regions (Flanders, Lithuania, South Tyrol, and Brandenburg, Germany).

The goal was to systematically evaluate the functionality, performance, and user-friendliness of selected platform components. Seven participants from the project consortium took part in the tests, bringing diverse roles and perspectives.

To ensure comparable and structured feedback, a standardized "Internal Testing Template" was used. This template established minimum requirements for collecting feedback and ensured the comparability of results.

2.1 Test Procedure

The tests were carried out by seven consortium members across the four regions. Each participant followed a structured approach using the "Internal Testing Template" to document their findings.

The process included the following steps:

1. **Referencing**
    - **Reference to D2.4** (document or milestone reference)
    - **Platform component** (e.g., "Map View," "Registration")
    - **Feature name and description** (e.g., "Retrieve Parcel Information" or "Layer Selection")

Funded by
the European Union

2. **Test Execution:** Participants conducted the tests independently in their respective work environments. They were asked to document the following in the template
    - **Functionality:** *"Does the feature work as expected?" (Yes/No)*
    - **Performance:** Subjective assessment of speed and responsiveness (e.g., "fast," "delayed," or "acceptable").
    - **Error Messages:** Detailed description of any bugs or unexpected behavior, including screenshots or log excerpts if applicable.
    - **Improvement Suggestions:** Open text field for ideas on optimization or adjustments.
    - **Test Notes:** A brief description of how the test was conducted (e.g., "tested step-by-step workflow" or "validated data with sample parcels").
3. **Technical Conditions:** To ensure the reproducibility of results, the technical environments of the testers were recorded
    - **Device type**(e.g. laptop, desktop PC)
    - **Operating system** (primarily Windows 11)
    - **Browser and version** (mainly Google Chrome and Mozilla Firefox, using the latest versions)
    - **Test date**

2.2 Overview of Test Environments

Most tests were conducted on the following systems:

- **Device:** Laptop (predominant)
- **Operating System:** Windows 11
- **Browser:** Chrome (most frequently used) and Firefox
- **Time frame:** September 2025 (exact dates noted in the respective feedback forms)

The diversity of locations and devices allowed for the consideration of region-specific characteristics (e.g., network conditions or local datasets) and the identification of potential cross-platform differences.

2.3 Purpose of the Template

The “Internal Testing Template” serves several key objectives:

- **Standardization:** Provides a uniform structure for all feedback, enabling consistent evaluation of the data.

Funded by
the European Union

- **Traceability:** Documentation of the test environment (device, OS, browser) allows issues to be reproduced systematically.
- **Efficiency:** Clear categories (e.g., Bug, Performance Comment) simplify the prioritization of adjustments.
- **Transparency:** Test notes help clarify whether deviations are due to user error, technical limitations or actual bugs.

2.4 Test Results

The evaluation of the collected feedback forms yielded valuable insights.

Through systematic analysis of the test results, specific bugs were identified, performance bottlenecks were pinpointed, and user-driven improvement suggestions were captured. These findings have been strategically integrated into ongoing development and optimization processes to continuously enhance the platform.

Bugs and Errors

- **Localization issues:** Several localization issues were identified, including incorrect or missing translations
- **Area Calculation issue:** Issue with incorrectly calculated areas for parcels and regions.
- **Geodata Upload issue:** Challenges arose across project regions due to the use of different coordinate reference systems (CRS) and GIS software solutions. Each region often relies on its own standard projections and different GIS tools (such as ArcGIS, QGIS, or others) embed these projections differently in the geodata file metadata. This inconsistency led to complications in data processing and integration.
- **Layer issues:** Issue with incorrect availability of bird layers across regions.
- **Issues with provided data**:
    - The shape-file for South Tyrol was not the correct one
    - Parcel IDs for Germany could not use the "official" parcel IDs which are used by farmers - this could not be implemented because the required data is not available for all federal states of Germany. Therefore, IDs generated by the Thünen Institute[3] were implemented (which are also the base for the optimization calculations).
    - Parcel IDs for South Tyrol: No complete data set is available for South Tyrol in which all parcels have a unique ID. To solve this issue for the web platform, LUP created its own parcel IDs to be able to display all parcels in the dataset and ensure their findability.

[3] https://atlas.thuenen.de/atlanten/agraratlas

Funded by the European Union

- **Zoom Availability issues:** Zoom in on the orthophoto data did not work (this was fixed by creating additional pre-calculated layers)
- **Performance issues**: Performance speed was experienced as too slow (this was addressed in Deliverable D6.3 and in chapter 1.2 above).

Recommendations for Improvement

The following are the recommendations of test users:

- **Background Maps:** Satellite background maps for all regions.
- **Add data on special funding zones** for Flanders (e.g., zones in which specific bird species are protected)
- **Statistics:** Change the majority bird layer to an average-based representation to improve its informative value.
- **Educational aspect**: Better explain the terms displayed (as for example “Habitat suitability”)
- **Visual simplification**: Rounding values or display classes in traffic-light colors.
- **Optimization:** As part of the platform’s further development, the current raster-based display is being replaced with cohesive, vector-based parcel layers that specifically show only those parcels with concrete optimization potential. Furthermore, enhance the optimization view by adding detailed descriptions and visual highlighting, and rework its user flow for improved clarity and usability.
- **Habitat suitability layer South Tyrol**: Filter the map to display habitat suitability exclusively for farmland areas in South Tyrol.
- **Comparability of status and optimization:** Optimization results were first displayed as raster data (because the original results of the calculation were raster data). To improve comparability they were then also changed to vector based visualization in the platform.
- **Year:** Add the year for which the data is presented.
- **Accessibility:** Changes of colors and maps for better visibility, including for color blindness.

2.5 Measures

Resolution of All Identified Bugs

All technical issues reported during testing and user feedback have been analyzed and successfully resolved. This included, in particular:

- Correction of projection issues during geodata uploads (e.g., inconsistencies caused by differences in the regional CRS).
- Resolution of incorrect area calculations for parcels and regions.

Funded by
the European Union

- Fix for inconsistent availability of bird layers across different project regions.

Implementation of Improvement Suggestions

Most of the user feedback points have been integrated into the platform, including:

- **Revised Optimization View:** The optimization view has been enhanced with optimized data visualization, including clear descriptions, visual highlighting, and a redesigned user flow for improved usability (for example when looking at a parcel, then only display comparison of optimization results in habitat suitability for this parcel, when a bird species is selected, then only display the measure that should support this bird species).
- Adjustment of parcel layer visualization (transition from raster to vector-based layers, focusing on optimization-relevant areas).
- **Educational aspect**: Implemented a small "?"-Button to the technical terms which links to the FAQ where more explanations can be found. Additionally, the year is now added as information.
- **Simplification of displayed values:** The numerical values have been rounded (i.e., no decimal places).
- Display of the Average instead of the Majority of all available habitat suitability layers per region.
- **Background Maps:** Added satellite background maps for all regions
- **Accessibility:**
    - Change of color palette to tones that can be better distinguished by people with color blindness
    - Color of background map changed to grey tones (for better visibility of the parcels in green tones/good habitat suitability)

3 Stakeholder demonstration feedback

The demonstration activities (WP7) increased the user base of the first demo platform and therefore also led to isolated feedback on functions.

There was a little less feedback on registration, which resulted in a bug report on the activation link and otherwise only required some support (password reset).

The demonstration activities and feedback are described in detail in D7.3 and D7.4.

From the standpoint of platform testing and quality assurance, the stakeholder testing activities provided beneficial input regarding general usability and for identifying remaining bugs.

However, quality assurance requires focussed testers from the target regions with different backgrounds and structured test reports, which went beyond the goal of the stakeholder demonstration activities (which are described in detail in Deliverables D7.2, D7.3 and D7.4).

Funded by
the European Union

Conclusion

The comprehensive testing phase has successfully confirmed that all identified bugs and technical issues have been resolved, ensuring the platform's stability and reliability.

The implementation of user feedback and optimization suggestions—such as enhanced layer visualization, improved data representation (e.g., transitioning bird layers from "majority" to "average"), and performance upgrades—has significantly improved both usability and functionality.

The platform now provides a more intuitive user experience, with clear descriptions, visual highlighting, and streamlined workflows that enable users to make data-driven decisions efficiently. Performance improvements, such as pre-rendered tiles (WMTS) and optimized data queries, further contribute to a seamless experience, even when handling large datasets or multiple regions.

While the current version meets the defined requirements, continuous monitoring and user feedback will remain essential to identify further opportunities for refinement. Future updates will focus on scalability, additional optimization criteria, and integration of new datasets to support evolving project needs.

This report concludes that the platform is ready for deployment and fully aligned with the project's technical and functional goals.

Funded by
the European Union

Appendix

Postman Data


```
{
  "info": {
    "_postman_id": "a5b3cc47-05b3-4620-8adf-47bd3c829a80",
    "name": "REST API basics: CRUD, test & variable",
      "description": "# 🚀 Get started here\n\nThis template guides you through CRUD operations (GET, POST, PUT, DELETE), variables, and tests.\n\n## 🔖 **How to use this template**\n\n#### **Step 1: Send requests**\n\nRESTful APIs allow you to perform CRUD operations using the POST, GET, PUT, and DELETE HTTP methods.\n\nThis collection contains each of these [request](https://learning.postman.com/docs/sending-requests/requests/) types. Open each request and click \"Send\" to see what happens.\n\n#### **Step 2: View responses**\n\nObserve the response tab for status code (200 OK), response time, and size.\n\n#### **Step 3: Send new Body data**\n\nUpdate or add new data in \"Body\" in the POST request. Typically, Body data is also used in PUT request.\n\n```\n{\n    \"name\": \"Add your name in the body\"\n}\n\n```\n\n#### **Step 4: Update the variable**\n\nVariables enable you to store and reuse values in Postman. We have created a [variable](https://learning.postman.com/docs/sending-requests/variables/) called `base_url` with the sample request [https://postman-api-learner.glitch.me](https://postman-api-learner.glitch.me). Replace it with your API endpoint to customize this collection.\n\n#### **Step 5: Add tests in the \"Scripts\" tab**\n\nAdding tests to your requests can help you confirm that your API is working as expected. You can write test scripts in JavaScript and view the output in the \"Test Results\" tab.\n\n<img src=\"https://content.pstmn.io/fa30ea0a-373d-4545-a668-e7b283cca343/aW1hZ2Uu
```

```
cG5n\" alt=\"\" height=\"1530\" width=\"2162\">\n\n## 💪 Pro tips\n\n- Use folders to group related requests and organize the collection.\n    \n- Add more [scripts](https://learning.postman.com/docs/writing-scripts/intro-to-scripts/) to verify if the API works as expected and execute workflows.\n    \n\n## 💡Related templates\n\n[API testing basics](https://go.postman.co/redirect/workspace?type=personal&collectionTemplateId=e9a37a28-055b-49cd-8c7e-97494a21eb54&sourceTemplateId=ddb19591-3097-41cf-82af-c84273e56719)\n[API documentation](https://go.postman.co/redirect/workspace?type=personal&collectionTemplateId=e9c28f47-1253-44af-a2f3-20dce4da1f18&sourceTemplateId=ddb19591-3097-41cf-82af-c84273e56719)\n[Authorization methods](https://go.postman.co/redirect/workspace?type=personal&collectionTemplateId=31a9a6ed-4cdf-4ced-984c-d12c9aec1c27&sourceTemplateId=ddb19591-3097-41cf-82af-c84273e56719)",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
    "_exporter_id": "45128374",
    "_collection_link": "https://laura-2561201.postman.co/workspace/Laura's-Workspace~7bbaa92b-ae3a-4fc4-820a-36cd1f8bf9aa/collection/45128374-a5b3cc47-05b3-4620-8adf-47bd3c829a80?action=share&source=collection_link&creator=45128374"
  },
  "item": [
    {
      "name": "Test ok",
      "event": [
        {
          "listen": "test",
          "script": {
```

```
          "exec": [
            "pm.test(\"Successful POST request\", function () {",
            "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
            "});",
            ""
          ],
          "type": "text/javascript",
          "packages": {}
        }
      }
    ],
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Accept-Language",
          "value": "de-DE,de;q=0.9,en;q=0.8",
          "type": "text"
        }
      ],
      "body": {
        "mode": "raw",
        "raw": "{\n  \"parcel_id\": \"ABC123\"\n}",
        "options": {
          "raw": {
```


Funded by
the European Union

```
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK SQL Injection",
      "event": [
        {
          "listen": "test",
```

```
        "script": {
          "exec": [
            "pm.test(\"Successful POST request\", function () {",
            "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
            "});",
            ""
          ],
          "type": "text/javascript",
          "packages": {}
        }
      }
    ],
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Accept-Language",
          "value": "de-DE,de;q=0.9,en;q=0.8",
          "type": "text"
        }
      ],
      "body": {
        "mode": "raw",
        "raw": "{\n  \"parcel_id\": \"ABC123' OR '1'='1\"\t\n}",
        "options": {
```

```
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK SQL Injection2",
      "event": [
        {
```

```
        "listen": "test",
        "script": {
          "exec": [
            "pm.test(\"Successful POST request\", function () {",
            "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
            "});",
            ""
          ],
          "type": "text/javascript",
          "packages": {}
        }
      }
    ],
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Accept-Language",
          "value": "de-DE,de;q=0.9,en;q=0.8",
          "type": "text"
        }
      ],
      "body": {
        "mode": "raw",
        "raw": "{\n  \"parcel_id\": \"' OR 1=1 -- \"\n}",
```

```
          "options": {
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK Ruby Injection",
      "event": [
```

```
    {
      "listen": "test",
      "script": {
        "exec": [
          "pm.test(\"Successful POST request\", function () {",
          "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
          "});",
          ""
        ],
        "type": "text/javascript",
        "packages": {}
      }
    }
  ],
  "request": {
    "method": "POST",
    "header": [
      {
        "key": "Accept-Language",
        "value": "de-DE,de;q=0.9,en;q=0.8",
        "type": "text"
      }
    ],
    "body": {
      "mode": "raw",
```

```
          "raw": "{\n  \"parcel_id\": \"+sleep(5.to_i)+\"\n}",
          "options": {
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK Python Injection",
```

```
  "event": [
    {
      "listen": "test",
      "script": {
        "exec": [
          "pm.test(\"Successful POST request\", function () {",
          "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
          "});",
          ""
        ],
        "type": "text/javascript",
        "packages": {}
      }
    }
  ],
  "request": {
    "method": "POST",
    "header": [
      {
        "key": "Accept-Language",
        "value": "de-DE,de;q=0.9,en;q=0.8",
        "type": "text"
      }
    ],
    "body": {
```

```
          "mode": "raw",
              "raw": "{\n   \"parcel_id\": \t\"__import__('os').system('sleep
5')\"\n}",
          "options": {
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API
via the request body. This request submits JSON data, and the data is
reflected in the response.\n\nA successful POST request typically returns a
`200 OK` or `201 Created` response code."
      },
      "response": []
    },
```

```
{
  "name": "NOK Shell Injection",
  "event": [
    {
      "listen": "test",
      "script": {
        "exec": [
          "pm.test(\"Successful POST request\", function () {",
          "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
          "});",
          ""
        ],
        "type": "text/javascript",
        "packages": {}
      }
    }
  ],
  "request": {
    "method": "POST",
    "header": [
      {
        "key": "Accept-Language",
        "value": "de-DE,de;q=0.9,en;q=0.8",
        "type": "text"
      }
```

Funded by
the European Union

```
        ],
        "body": {
          "mode": "raw",
          "raw": "{\n  \"parcel_id\": \t\"ABC123 && sleep 5\"\t\n}",
          "options": {
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
```

```
  },
  {
    "name": "NOK Shell Injection2",
    "event": [
      {
        "listen": "test",
        "script": {
          "exec": [
            "pm.test(\"Successful POST request\", function () {",
            "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
            "});",
            ""
          ],
          "type": "text/javascript",
          "packages": {}
        }
      }
    ],
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Accept-Language",
          "value": "de-DE,de;q=0.9,en;q=0.8",
          "type": "text"
```

```
        }
      ],
      "body": {
        "mode": "raw",
        "raw": "{\n  \"parcel_id\": \"ABC123; rm -rf /\"\t\n}",
        "options": {
          "raw": {
            "language": "json"
          }
        }
      },
      "url": {
        "raw": "{{base_url}}/api/getEnvParameters.php",
        "host": [
          "{{base_url}}"
        ],
        "path": [
          "api",
          "getEnvParameters.php"
        ]
      },
        "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
    },
```

```
    "response": []
  },
  {
    "name": "NOK Code Injection",
    "event": [
      {
        "listen": "test",
        "script": {
          "exec": [
            "pm.test(\"Successful POST request\", function () {",
            "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
            "});",
            ""
          ],
          "type": "text/javascript",
          "packages": {}
        }
      }
    ],
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Accept-Language",
          "value": "de-DE,de;q=0.9,en;q=0.8",
```

```
            "type": "text"
          }
        ],
        "body": {
          "mode": "raw",
          "raw": "{\n  \"parcel_id\": \t\"' + eval('danger') + '\"\t\n}",
          "options": {
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
```

```
    },
    "response": []
  },
  {
    "name": "NOK Broken JSON",
    "event": [
      {
        "listen": "test",
        "script": {
          "exec": [
            "pm.test(\"Successful POST request\", function () {",
            "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
            "});",
            ""
          ],
          "type": "text/javascript",
          "packages": {}
        }
      }
    ],
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Accept-Language",
```

```
            "value": "de-DE,de;q=0.9,en;q=0.8",
            "type": "text"
          }
        ],
        "body": {
          "mode": "raw",
          "raw": "{\n  \"parcel_id\": ABC123'\n}",
          "options": {
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{base_url}}/api/getEnvParameters.php",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "api",
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API
via the request body. This request submits JSON data, and the data is
```


Funded by
the European Union

```
reflected in the response.\n\nA successful POST request typically returns a
`200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK Empty",
      "event": [
        {
          "listen": "test",
          "script": {
            "exec": [
              "pm.test(\"Successful POST request\", function () {",
              "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
              "});",
              ""
            ],
            "type": "text/javascript",
            "packages": {}
          }
        }
      ],
      "request": {
        "method": "POST",
        "header": [
```

```
      {
        "key": "Accept-Language",
        "value": "de-DE,de;q=0.9,en;q=0.8",
        "type": "text"
      }
    ],
    "body": {
      "mode": "raw",
      "raw": "{\n  \"parcel_id\": \t\"\"\n}",
      "options": {
        "raw": {
          "language": "json"
        }
      }
    },
    "url": {
      "raw": "{{base_url}}/api/getEnvParameters.php",
      "host": [
        "{{base_url}}"
      ],
      "path": [
        "api",
        "getEnvParameters.php"
      ]
    },
```

```
        "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK Umlaute",
      "event": [
        {
          "listen": "test",
          "script": {
            "exec": [
              "pm.test(\"Successful POST request\", function () {",
              "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
              "});",
              ""
            ],
            "type": "text/javascript",
            "packages": {}
          }
        }
      ],
      "request": {
        "method": "POST",
```

```
    "header": [
      {
        "key": "Accept-Language",
        "value": "de-DE,de;q=0.9,en;q=0.8",
        "type": "text"
      }
    ],
    "body": {
      "mode": "raw",
      "raw": "{\n  \"parcel_id\": \t\"äöüß\"\n}",
      "options": {
        "raw": {
          "language": "json"
        }
      }
    },
    "url": {
      "raw": "{{base_url}}/api/getEnvParameters.php",
      "host": [
        "{{base_url}}"
      ],
      "path": [
        "api",
        "getEnvParameters.php"
      ]
```

```
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK HTML Injection",
      "event": [
        {
          "listen": "test",
          "script": {
            "exec": [
              "pm.test(\"Successful POST request\", function () {",
              "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
              "});",
              ""
            ],
            "type": "text/javascript",
            "packages": {}
          }
        }
      ],
      "request": {
```


Funded by
the European Union

```
"method": "POST",
"header": [
  {
    "key": "Accept-Language",
    "value": "de-DE,de;q=0.9,en;q=0.8",
    "type": "text"
  }
],
"body": {
  "mode": "raw",
  "raw": "{\n  \"parcel_id\": \t\"<script>alert(1)</script>\"\t\n}",
  "options": {
    "raw": {
      "language": "json"
    }
  }
},
"url": {
  "raw": "{{base_url}}/api/getEnvParameters.php",
  "host": [
    "{{base_url}}"
  ],
  "path": [
    "api",
    "getEnvParameters.php"
```

```
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    },
    {
      "name": "NOK Whitespaces",
      "event": [
        {
          "listen": "test",
          "script": {
            "exec": [
              "pm.test(\"Successful POST request\", function () {",
              "    pm.expect(pm.response.code).to.be.oneOf([200, 201]);",
              "});",
              ""
            ],
            "type": "text/javascript",
            "packages": {}
          }
        }
      ],
```

```
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Accept-Language",
          "value": "de-DE,de;q=0.9,en;q=0.8",
          "type": "text"
        }
      ],
      "body": {
        "mode": "raw",
        "raw": "{\n  \"parcel_id\": \t\" ABC123 \"\n}",
        "options": {
          "raw": {
            "language": "json"
          }
        }
      },
      "url": {
        "raw": "{{base_url}}/api/getEnvParameters.php",
        "host": [
          "{{base_url}}"
        ],
        "path": [
          "api",
```

```
            "getEnvParameters.php"
          ]
        },
          "description": "This is a POST request, submitting data to an API via the request body. This request submits JSON data, and the data is reflected in the response.\n\nA successful POST request typically returns a `200 OK` or `201 Created` response code."
      },
      "response": []
    }
  ],
  "event": [
    {
      "listen": "prerequest",
      "script": {
        "type": "text/javascript",
        "exec": [
          ""
        ]
      }
    },
    {
      "listen": "test",
      "script": {
        "type": "text/javascript",
        "exec": [
```

```
            ""
          ]
        }
      }
    ],
    "variable": [
      {
        "key": "id",
        "value": "1"
      },
      {
        "key": "base_url",
        "value": "https://postman-rest-api-learner.glitch.me/"
      }
    ]
}
```