Provision for acceptance

Dear Prof. Illes-Seifert,

Dear Prof. Oetzel,

As contractually agreed, you will receive the following deliverables for acceptance by 30.06.2025:

System: vulnerabilitydashboard,

version 1.0.0 (can be found in the file package.json),

timestamp: 2025-06-30 23:59:00.

Technical documentation (artifact, version 1.0.0, 2025-06-21 15:06:00) with the following content:

Introduction

Requirement specification

Architectural documentation

Test documentation

Acceptance documentation

User documentation (optional)

Project report

Alternatively, other documents that are part of your delivery can be listed here.

If applicable: We draw your attention to the following known defects:

No.	Defect/Artifact	Description
1	Todoapp- sonar sast artifacts	This is not of a defect rather than a suggestion. The artifacts are totally fine and is saved in as .json, however, as we used sonarqube we would suggest to look at the analysis result via the sonarqube UI for better view. The project is public just follow the link :). todoapp - Issues - SonarQube Community Build
2		

We look forward to a successful acceptance. You will also find the acceptance report enclosed.

Yours sincerely

Team 14. GitGuardians.

<Guardiola, Antonio Huesa.

Karnavat, Aadit.

Farag, Beshoy.>

Word template
Project report

Study program

Software Design

TH Aschaffenburg

21.06.2025

Authors:
First name: Aadit
Matriculation number: 22 76 878

First name: Antonio
Matriculation number: 22 92 339

First name: Beshoy
Matriculation number: 22 83 841

Team name: Team 14

ASCHAFFENBURG UNIVERSITY OF TECHNOLOGY

FACULTY OF ENGINEERING AND COMPUTER SCIENCE

WÜRZBURGER STRASSE 45

D-63743 ASCHAFFENBURG

Table of contents

1 Project approach (Written by Beshoy)

1.1 SCRUM flow

As a starting point: all the events related to agile (not the development) took place at the lecture room C3 - 103-102

We start with the planning; where we set the sprint goal in an understandable for every team member way.

Then we decide on what can be done to achieve it and format it in a user story way lead by the project owner and reviewed then agreed upon by the development team.

We then do a magic estimation for them, priortize them, and the scrum master asks the team memeber to pull the user stories, some of these user stories then should have a strict deadline for how important they are.

After the first week of the sprint, the weekly scrum takes place where the team members say what they have done, what they will do and what obstacles they faced, depending on what problem they have faced and what was not finished as a result, a refinement can be done in this case.

At the end of the sprint a review and retrospective is done.
Communication each 1-2 days was done in any possible way.

1.2 Procedure for SCRUM events

Describe the SCRUM events that you have carried out in the team. In doing so, go into the practiced procedure and the most important
goals and timeboxing:

1.2.1 **Sprint**

Description:

In my opinion working in sprints help break down the project into small pieces (sprints), then we break these sprints (user stories), and iterate. This helps us work organized and make it easy to achieve our goals.

The further we went into sprints the better and faster our team has progressed, it was exponential.

On average a user story would be broken down into 2-3 tasks

1.2.2 **Daily Scrum**

Description:

Since the 2nd sprint, our entire team has consistently participated in a weekly Scrum meeting held every Wednesday. These meetings typically lasted about 15 minutes and followed the three key questions:

- What did you accomplish this week?
- What will you work on next week?
- What obstacles are you facing?

Full team participation was ensured during these sessions. In addition to the weekly meetings, daily online communication was encouraged and soon became a habit, with team members automatically updating each other every 1-2 days.

By the 3rd sprint, the team had developed a strong culture of openness and collaboration.

One of the greatest benefits of this regular exchange was the valuable honest but respectful technical feedback shared and the team members reaching out for help.

1.2.3 **Sprint Planning 1 and 2**

Description:

In our experience, the greatest benefit of agile release planning was crafting the user stories, and pulling the work so we are able to start once we relaese the sprint plan document.

- **Sprint 1:** Setting up the virtual machine, getting the pipeline ready to accept jobs, figuring out the goals of the project.
- **Sprint 2:** Running the same security checks made for the pipeline locally and achieve the results.
- **Sprint 3:** CI/CD pipelines implementation of security tools for both of the apps in gitlab server.
- **Sprint 4**: Finish the pipeline development since some security checks were not complete and faced with issues, and start developing the dashboard (setu up the project for pulling the reports from the gitlab artifacts API)
- **Sprint 5:** the documentation is done for the project, the dashboard shows the reports and meets the deliverables.

Cutting user stories: first the user story where searched on a techincal level then split into acheivable tasks.

Acceptance Criteria: Acceptance criteria were decided on before the start of the user story, by the project owner with the developer or the excuter of the crossponding user story.

Changes made in the planninng: After the first 2 sprint the sprints, we added the set priorty for each user story, and to try to set deadlines for certain tasks.

1.2.4 **Review**

In my opinion the greatest benifit: The team communicates what type of issues they are facing; this provides space for technical knowledge exchange, we then can solve a problem that a certain member faced for few hours in few minutes.

As the sprints go, the user stories get bigger and harder, the acceptance rate was not going up but slightly down (more work was done nevertheless), and after each weekly scrum we would refine our backlog.

By the end of the 3rd sprint the pipeline was running the checks.

1.2.5 **Retrospective**

Description:

In my opinion addressing what went not so well is the most important point, it helps us each address what problems we faced (agile wise) and we solve them, that helps us improve as a team by eliminating our mistakes.

The deadlines and the documentation was something that was mentioned during our retrospective, by time more deadlines were met, and documentation was done.

The documentation and the backlog are kept mor clean and up to date.

It does not change; however, by time it became more easy for the team to communicate what did not go well and what we need to improve.

1.3 SCRUM roles

Scrum master: ensures the team participates and each of them give feedback and contribute to any on going process. Protect the team from the product owner. Moderates meetings, motivates the team.

Product owner: sets the goal and the vision for the product sets the acceptance criteria. Craft the user stories (then reviewed by the team).

Security Champion : member who acts as the bridge between developers and security. Their responsibilities include Advocating for security best practices in sprint planning and retrospectives as well as reviewing and maintaining security scanning tools

1.4 DoD

Insert your DoD here: https://ahuesag.atlassian.net/wiki/x/l4AB

Tasks are reviewed by peers, product owner accepts according to their acceptance criteria.

1.5 Mapping for individual performance (By all team members)

Antonio Guardiola:

Generic:

- Checked docker was installed in VM
- Configure gitlab-runner in VM to run pipeline jobs from gitlab projects

TodoList’s pipeline:

- Custom Image via Dockerfile
- Build stage (build image and build project)
- Publish stage (publish package and container)
- Container scanning stage
- Testing stage (unit testing and integration testing)
- Packaging stage
- Fuzzing stage

JuiceShop’s pipeline:

- Custom Image via Dockerfile
- Build stage (build image and build project)
- Publish stage (publish container)

- Container scanning stage
- Fuzzing stage

Vulnerability dashboard:

- Chose project’s stack
- basic blank application set up
- Settings section
- Fuzzing section
- Container scanning section
- User interface

Aadit Karnavat:

Generic:

- Created folders like reports/ manually when needed to ensure artifacts existed

Both pipelines:

- Implemented the Dependency scanning
- Manually verified pipeline success

Vulnerability Dashboard:

- Developed the Dependency scanning part

Beshoy Farag:

Generic:

- Freed some space on the VM
- Connected gitlab to the virtual machine

TodoList’s pipeline:

- Implemented the SAST SonarQube
- Implemented the GitLeaks
- Did the deploy stage
- Implemented the DAST Stage
- Added the dockerhub login data for authentication for pulling docker Images

JuiceShop’s pipeline:

- Implemented the SAST semgrep
- Implemented the GitLeaks
- Did the deploy stage
- Implemented the DAST Stage
- Added the dockerhub login data for authentication for pulling docker Images

Vulnerability dashboard:

- Implemented the SAST stage
- Implemented the DAST stage

2 Team

2.1 Team name and members (Written by Beshoy)

Team 14, GitGuradians

All team members participated in the backlog refinement the magic estimation and the documentation of single tasks.

Aadit Karnavat (security champion): Ensures security practices are considered, Helps identify security risks during sprint planning.

Antonio Guardiola (product owner): Product owner: sets the goal and the vision for the product sets the acceptance criteria. Craft the user stories (then reviewed by the team).

Beshoy Farag. (scrum master): ensures the team participates and each of them give feedback and contribute to any on going process. Protect the team from the Product owner.

2.2 Team Commitment (Written by Aadit)

Transparency:

Reach out for help when you need it, let the team know if there will be any delay.

Respect & communication:

Communication every 1-2 days online, and attending each lecture.
When a team member does not agree on something they should address Respectfully.

Continuous improvement:

After each sprint during the retrospective each team member should say What we can improve, the team members then follow these suggestions The next sprints.

Shared goal:

Each team member knows what our clear final goal is, each sprint Planning addresses the goal of the sprint in an understandable way

2.3 Team Values (Written by Antonio Huesa Guardiola)

For our team the most important SCRUM values have been focus, courage and commitment.

- Focus, because we must not lose our concentration on our projects‘ development if we want to get them into a finished state.

- Courage, because due to the little size of our team, we need the courage and effort to do the work of multiple roles and learn new things for the completition of the development.

- Commitment, because due to the complexity and challenge of the tasks, we have to ensure that the team is commited to the final goal of the projects.

2.4 Retrospectives (Edited by Aadit)

Sprint 4 Retrospective

Date: [18-06-2025]

What went well:

1. The team successfully completed the sprint work.
2. Team members supported one another by asking for help and clarifications.
3. All security tools were completed ahead of time.

What didn’t go so well:

1. Time was spent on tasks that should have been completed in the previous sprint.

What needs to improve:

1. Optimize pipeline efficiency.
2. Use timeboxing and estimation for better task management.

Sprint 3 Retrospective

Date:[04-06-2025]

What went well:

1. Communication within the team improved.
2. Higher volume of completed work.
3. Greater technical understanding was achieved.

What didn’t go so well:

1. Dependency management was lacking.
2. Deadlines were not clearly defined.
3. Insufficient documentation.

What needs to improve:

1. Better and more consistent documentation.
2. Start work earlier in the sprint.
3. Evaluate and plan tools before implementing.
4. Set and adhere to task timelines.
5. Actively seek feedback during development.

3 Summary (Edited by Antonio Huesa Guardiola)

All agile events were held in room C3 - 103-102. Each sprint began with planning, where goals were set for the crossponding sprint, user stories defined, estimated, and prioritized. Team members then pulled tasks (one of the scrum values not to push tasks), with some given strict deadlines for crucial user stories. Mid-sprint, a weekly scrum addressed progress and blockers, leading to refinements if needed. Each sprint ended with a review and retrospective. The team communicated regularly every 1–2 days mostly online.

The team used structured work into sprints, gradually improving in speed and quality. Sprint planning focused on defining user stories, setting goals, and assigning priorities. Weekly Scrums and regular communication supported progress tracking and collaboration. Communication helped solve technical issues quickly, while retrospectives focused on identifying problems and improving team practices. Over time, the team became more organized, met more deadlines, and maintained better documentation.

The scrum master ensures each member participates and moderates the meeting, protect the team from the product owner. The product owner defines the vision and sets the user stories and its acceptance criteria.

The team values transparency by openly asking for help and informing each other when there are delays. Communication happens on each lecture as well as each 1-2 days online. Any disagreement is handled with respect. Contiuous improvement is made from retrospectives. Everyone works towards a shared goal with focus, commitment and courage, being these the three most important scrum values for our team.

4 Appendix

Unfortunately, our team was formed later in the project phase, and as a result, we were not present for the lecture covering the Lego model formulation. We sincerely apologize for not being able to provide a Lego model for this assignment.

High Priority:

- Sprint 3:
  - Documentation, do it and be consistent
  - Start earlier with the pulled tasks
- Sprint 4:
  - Refine and check the pipeline

Mid Priority:

- Sprint 3:
  - put time line for the task
- Sprint 4:
  - Do timeboxing

Low Priority:

Word template
Technical documentation

Study program

Software Design

TH Aschaffenburg

23.06.2025

Authors:
First name Last name: Beshoy Farag
Matriculation number: 22 83 841

First name Last name: Antonio Huesa Guardiola
Matriculation number: 22 92 339

First name Last name:Aadit Karnavat
Matriculation number: 22 76 878

Team name: Team 14, GitGuardians

ASCHAFFENBURG UNIVERSITY OF APPLIED SCIENCES

FACULTY OF ENGINEERING AND COMPUTER SCIENCE

WÜRZBURGER STRASSE 45

D-63743 ASCHAFFENBURG

Table of contents

1 Introduction

1.1 Purpose (Written by Beshoy Farag)

This document was created by Team 14 (GitGuardians).
For Professor Illes-Seifert and Professor Oetzel.

This document covers all the technical work from the start of the VM set up, development of the pipeline until the dashboard development in detail.

This document is binding for all the stakeholders involved in this project.

1.2 Summary (Written by Beshoy Farag)

Stakeholders:

Product owner, scrum master, DevOps engineer, Security Champ, Securtiy analysts, Web Developers, Vulnerability dashboard users .e.g: other developers of an app that the vulnerability dashboard shows its analysis.

System Title: Vulnerability Dashboard

System Objectives:

The Vulnerability Dashboard is a desktop-based application designed to display the latest security vulnerabilities detected in a web application called juice shop. It achieves this by pulling and analyzing artifacts generated from automated security checks ran via a gitlab CI/CD pipeline. The core objective is to provide developers with immeditae (live) and organized visibility into the security status of their application after each code commit.

System Scope:

Integration with GitLab's API to automaticaclly fetch artifacts (primarily .json but sometimes as html or raw text) from recent CI/CD pipeline runs.

Parsing and analysis of scan artifacts from these security checks stages:

Dependency check – SAST – container scanning – DAST – Fuzzing

Then presenting the results in an organized (sorted according to severtiy way) as well as an overview of the stages.

Out of Scope:

Role-based access; anyone who has the application can see the results, there is no log-in or credentials needed.

Technical and Business Context:

Platform: Desktop Application

Development: Two working CI/CD pipelines for two existing projects, one based in Java with the framework SpringBoot and the other a web page based on javascript, As well as a Node.js and electron based desktop app. The entire process from setting the virtual machine until finishing the desktop-app was split into 5 sprint each of 2 weeks.

Integration: Connects directly into gitlab API to retrieve the artifacts.

1.3 References Standards **and** regulations (optional)

2 Requirements documentation (Written by Beshoy / Edited by Antonio)

2.1 Product vision and goals

Product Vision:

To provide two atomated CI/CD pipelines that check both our projects vulnerabilitites to ease its troubleshooting and development, as well as an automated vulnerability dashboard that shows the security vulnerabilties of the JuiceShop-OWASP project after each commit in an organized way and intiutive way.

Each Sprint goal (as mention before in the project „agile“ report):

- **Sprint 1:** Setting up the virtual machine, getting the pipeline ready to accept jobs, figuring out the goals of the project.
- **Sprint 2:** Running the same security checks made for the pipeline locally and achieve the results.
- **Sprint 3:** CI/CD pipelines implementation of security tools for both of the apps in gitlab server.
- **Sprint 4**: Finish the pipeline development since some security checks were not complete and faced with issues, and start developing the dashboard (setu up the project for pulling the reports from the gitlab artifacts API)
- **Sprint 5:** the documentation is done for the project, the dashboard shows the reports and meets the deliverables.

2.2 Personas (Edited by Aadit)

Persona - 1

Description:

Joe Werner.
Web Developer
Married
Living and working in Darmstadt

My Typical Day:

Make updates and maintain the app
Learn new tools

My goals are:

To make the web app as useable and the features as efficient as possible.

Document and the share the results with the concerned shareholders

My Challenges:

While making changes in the code I might make it vulnerable

Sometimes my team and I face communication problems

I see and hear:

The updates needed by the shareholders

Team members asking for help

These are my wishes:

Some app or interface to show me my vulnerabilities in real time.

To see my vulnerabilities every time I update the app and commit through the pipeline.

Good communication

Documentation of each process

I do and say this:

I develop the web app

“Features are important but they are not good unless secured”

Persona - 2

Description:

Thomas Horn.

Security Analyst

Divorced

Living and working in FFM

My Typical Day:

Reviewing daily security reports and alerts.

Monitoring ongoing vulnerability scans and assessments.

Collaborating with developers and DevOps teams to solve security issues.

Using tools to analyze vulnerabilities, .html and .json reports.

My goals are:

Quickly identify the most critical vulnerabilities affecting the Web App.

Be in sync with security analysis technologies.

Inform the security team about any ongoing vulnerabilities.

My Challenges:

Managing and interpreting large volumes of scan data in different formats.

Keeping track of the latest security status after every code commit.

Lack of a unified tool that automatically pulls the latest vulnerability data.

Difficulty in prioritizing vulnerabilities based on severity .

I see and hear:

Developers discussing new features and tight deadlines.

Alerts and notifications about new vulnerabilities.

These are my wishes:

A user-friendly dashboard that shows all vulnerability data automatically after each pipeline run.

Clear visualization of security risks, sorted by severity and type.

Real-time updates without manual intervention.

Easy access to detailed vulnerability information to aid quick decision-making.

I do and say this:

“I need a reliable tool that shows me the latest security issues without hunting through multiple reports.”

“Prioritizing risks quickly helps me focus on what truly matters.”

“Automation should reduce my workload, not add to it.”

Persona - 3

Description:

Jose Mendes.

DevOps Engineer.

Single

Living and working in Aschaffenburg

My Typical Day:

Managing and maintaining CI/CD pipelines for projects.

Monitoring build statuses, pipeline executions, and artifact generation.

Collaborating with developers and security teams to integrate security tools into pipelines.

Troubleshooting pipeline failures and optimizing automation workflows.

My goals are:

Automate security checks and integrate them seamlessly into the CI/CD pipeline.

Ensure timely availability of security artifacts for analysis after each build.

Maintain pipeline reliability and speed while adding security layers.

My Challenges:

Managing complex integrations between multiple tools .

Handling artifact formats (JSON, HTML).

Keeping up with evolving security tools and best practices.

I see and hear:

pipeline failures.

Requests from developers for faster build times and fewer pipeline disruptions.

Discussions about improving pipeline security and compliance.

Notifications from security analysts about missing or delayed vulnerability data.

These are my wishes:

A dashboard that automatically fetches and displays pipeline security artifacts without manual steps.

Clear feedback on security scan statuses within the pipeline process.

Better collaboration with security and development teams.

I do and say this:

“My goal is to keep our pipelines fast but secure.”

“Automation should save time, not cause extra work.”

“Security is a team effort — everyone must stay informed.”

“If the pipeline breaks, we fix it fast.”

2.3 User stories (Written By Beshoy)

Accepted: As a DevOps engineer, I want to verify connectivity and control between GitLab and my Ubuntu VM, so that the GitLab CI/CD pipeline can be run in the VM.
AC: Success in the access to the VM through the given ssh key has been documented. Gitlab CI/CD runner is able to establish a secure SSH connection to the target VM. The SSH credentials are securely stored using GitLab Ci/CD variables. The pipeline fails clearly if the connection cannot be established. The VM’s SSH logs show a successful connection attempt from the GitLab runner’s IP address or GitLab-hosted runner. Access to those variables is restricted to authorized users. The VM is configured to accept connections from GitLab runners

Accepted: As a DevOps engineer, I want to understand how the configuration file controls the CI/CD pipeline to write my own for the projects
AC: Read the official gitlab documentation for the pipeline configuration. Document the key elements to understand and link the offical documentation for the rest of developers to have a deep dive into it.

Accepted: As a DevOps engineer, I want to understand how the stamentens of the Dockerfile can help me create my own custom Docker image, to pass the enviroments in which both projects have been tested.
AC: Read the official documentation related to the subject.The key elements have been documented and the official documentation has been linked for the rest of developers to have a deep dive into it.

Accepted: As a DevOps engineer, I want to research the target audience for our vulnerability dashboard, so that we can design a user-centric tool.
AC: Stakeholder Identification.The key roles who will use or benefit from the dashboard. Personas Created. At least 1 user persona is developed that represent the primary dashboard users.Each persona includes demographics (role, experience level), needs, goals, and typical use cases.

Accepted: As a DevOps engineer, I want to create the Dockerfile files to use basic custom images for our projects‘ testing.

AC: Create a file named “DockerFile” in both projects. Fill those files with the needed base images and dependencies based on the projects needs Document the process, check that output shows that both projects build properly, indicating that the base images and dependencies do not conflict with the projects.

Accepted: As a DevOps engineer, I want to install gitlab runner in VM, so that GitLab can register and run each stage of our pipelines into the VM.
AC: GitLab Runner is successfully installed on the target virtual machine.The gitlab-runner command is available and returns the expected version.The runner is registered to the correct GitLab instance/project using a valid registration token.The runner appears as active under GitLab →Project → Settings → CI/CD → Runners.The GitLab Runner service is enabled and starts automatically on VM reboot.Verified by restarting the VM and confirming the runner remains active. A test pipeline runs using the installed runner, executing a basic script (e.g., echo Hello from VM runner) and completes successfully. Access to the VM and runner is secured (e.g., via SSH keys, firewall rules).

Accepted: As a DevOps engineer, I want to verify and set up access to the GitLab server from the VM, so that our runner can securely connect and pull repositories to execute the CI/CD jobs of each project.
AC: The virtual machine (VM) has a stable network connection and can reach the GitLab server.Git is installed on the VM and can be used to manually clone repositories.Authentication from the VM to the GitLab server is configured using a secure method. The GitLab Runner is successfully registered from the VM to the intended GitLab project or group using a valid registration token.The registered runner can authenticate and pull project repositories it is assigned to.A test pipeline job using this runner successfully clones the repository and completes a basic task. Runner connectivity persists after VM reboots or network restarts, indicating persistent and reliable configuration.

Accepted: As a DevOps engineer, I want to install docker engine in the VM.
AC: Install docker if it is not already installed in the VM. Verifiy that docker is installed via its command. Verifiy that the docker service is running. Document the process.

Accepted: As a DevOps engineer, I want to Implement the SAST tool learnt into both our pipelines config file to secure our projects.
AC: The selected SAST tool is successfully installed and integrated into the target projects.The SAST tool is configured with baseline rulesets appropriate to the project technology stack. Scan results are clearly visible and accessible in the VM. Identified security issues are categorized by severity. The SAST tool does not significantly increase the pipeline runtime beyond acceptable thresholds.Documentation is provided.Security policies and guidelines are updated to include SAST tool usage and handling of

scan results.The SAST tool can scan all major components of the projects, including backend, frontend, and infrastructure code as applicable.

Accepted: As a DevOps engineer, I want to learn about the chosen SAST tool for further implementation in our CI/CD pipelines
AC: The DevOps engineer has completed official documentation for the chosen SAST tool.The engineer has successfully run sample scans on test projects or codebases using the SAST tool.
The engineer understands the key features, configuration options, and limitations of the SAST tool.A summary report or knowledge document is created detailing how the tool works, how to configure it, and best practices.Potential integration points of the SAST tool into the existing CI/CD pipeline are identified.The engineer is familiar with how to interpret scan results and address common findings.A roadmap or plan for implementing the SAST tool is documented.

Accepted: As a DevOps, I want to choose the most appropiate SAST tool based on the projects needs
AC: A clear list of project requirements and security goals for the SAST tool is documented. Tool evaluations include factors such as supported languages, ease of integration, accuracy, performance, and cost.The selected SAST tool aligns with the technology stack and compliance requirements of the projects.A final recommendation report is created, detailing the evaluation process and justifying the choice.Stakeholders review and approve the chosen tool.Next steps for procurement or implementation are clearly outlined.

Accepted: As a security analyst, i want to run SAST analysis on our juiceshop. so that i catch vulnerabilities in my app's code.
AC: The chosen SAST tool fits our juiceshop security needs SAST analysis results are obtained in a visually appealing way e.g html The process is well documented results are accessible via ubuntu vm or gitlab repo for the team.

Accepted: As a DevOps engineer, I want to learn about how the chosen container scannning tool can be implemented in our CI/CD pipelines
AC: Document the key elements to learn of the official documentation. Link the official documentation of the tool.

Accepted: As a DevOps engineer, I want to implement the container scanning tool into our CI/CD pipelines configuration files, so that the container in which the porjects are run are scanned for vulnerabilities.
AC: Document the process of the implementation. Document the output log of the tool. Verify that the output logs that the tool functions as intended, that the tool is checking for vulnerabilities and does not fail in the search process.

Accepted: As a DevOps engineer, I want to research which container scanner tool fits better our use case, taking into account the requirements of both our projects‘ pipelines.
AC: Document which tool has been selected. Document the reasoning of choosing said tool.

Accepted: As a DevOps engineer, I want to implement the fuzzing tool in the todolist CI/CD pipeline, so that the project can be checked for fuzzing.
AC: The fuzzing dependency has been added to the project. The fuzzing function has been added to the project. The output of the tool has been documented. The documented output does not show any kind of failure in its use. The documented output logs the vulnerabilities that the project may have.

Accepted: As a DevOps engineer, I want to implement the fuzzing tool in the JuiceShop CI/CD pipeline, so that the project can be checked for fuzzing
AC: The fuzzing dependency has been added to the project.The fuzzing module has been added to the project.The fuzzing function has been added to the project. The output of the tool has been documented. The documented output does not show any kind of failure in its use. The documented output logs the vulnerabilities that the project may have.

Accepted: As a DevOps engineer, I want to learn about the specific fuzzing tool for further implementation on our CI/CD juiceshop pipeline.
AC: The Key elements have been documented. The official documentation has been read. The official documentation has been linked.

Accepted: As a security analyst, i want to implement the DAST tool for juiceshop app, so that i can catch running app vulnerabilities
AC: The reasons why you chose this DAST tool. The DAST tool shows the run time vulnerabilities. The results are stored on the ubuntu vm or on gitlab where team members can access it. The entire process is well documented for our team members.

Accepted: As a security analyst, i want to run DAST security checks on my todolist, to catch run time vulnerable behavior of my app.
AC: The reasons why you chose this DAST tool. The DAST tool shows the run time vulnerabilities. The results are stored on the ubuntu vm or on gitlab where team members. can access it.The entire process is well documented for our team members.

Accepted: As a DevOps engineer, I want to learn how to publish my container to share the enviroment of both my CI/CD pipelines and scan that container for vulnerabilitites
AC: The key elements to know have been documented. The official documentation has been read. The official documentation has been linked.

Accepted: As a DevOps engineer I want to learn how to publish a maven project package for the todolist project to register it into the GitLab register, and have a distributable of the project.
AC: Documentation of the key points to understand has been done,
The official documentation has been linked.

Accepted: As a Devops engineer, i want to implement fully the publishing of my both containers, sot that i can scan their vulnerabilities,
AC: The todolist CI/CD pipeline has a variable set to the container that is going to be published (pushed). The JuiceShop CI/CD pipeline has a variable set to the container that is going to be published (pushed)
The todolist CI/CD pipeline's container is pushed to the container registry
The JuiceShop CI/CD pipeline's container is pushed to the container registry.
The todolist project has the tag of the pushed container in its registry
The JuiceShop project has the tag of the pushed container in its registry.

Accepted: As a DevOps engineer, I want to publish the package of the todolist application.
AC: the project id has been referenced in the todolist application. The gitlab access token has been created. The output of the stage job has been documented. The output log of the pipeline shows that. The package register of gitlab registers the todolist application last published package.

Accepted: As i DevOps engineer, i want to implement the SAST in my yml file, so that the CI/CD pipeline run the checks automatically for the juiceshop
AC: The .gitlab-ci.yml file is updated to include the chosen SAST tool's scanning job.The SAST scan runs automatically on every pipeline execution.The scan uses the correct configuration and settings for the Juiceshop project.The pipeline job properly fails or warns on findings based on severity thresholds.Scan results are visible and accessible within GitLab's security artifacts.The SAST scan job completes within an acceptable time frame without causing major pipeline delays.The pipeline continues to run other jobs unaffected by the SAST scan.Documentation is updated to describe the SAST integration and how to interpret results.

Accepted: As i DevOps engineer, i want to implement the DAST in my yml file, so that the CI/CD pipeline run the checks automatically for the juiceshop
AC: The .gitlab-ci.yml file includes a DAST scanning job configured for the Juiceshop project. The DAST scan runs automatically on pipeline events such as pushes. The scan targets the correct application URL or environment (e.g., deployed Juiceshop instance). The DAST job completes successfully and reports vulnerabilities found in the web application. The scan results are integrated and visible in Gitlab artifacts. Documentation is updated to describe DAST integration, configuration, and interpreting scan results.

Accepted: As a DevOps engineer, I want to learn the tools used to deploy the juice-shop project
AC: The key elemets to know have been documented. The official documentation has been read. The official documentation has been linked.

Accepted: As a DevOps engineer, I want to learn the tools used to deploy the todolist project
AC: The key elemets to know have been documented. The official documentation has been read. The official documentation has been linked.

Accepted: As a DevOps engineer, I want to implement the deployment to the todolist project.
AC: The output log has been documented the output log has been proven to have no issues, indicating that the deployment runs without errors the output has been proven to log the success of the deployment.

Accepted: As a DevOps engineer, I want to implement the deployment in the juice-shop project.
AC: The output log has been documented.the output log has been proven to have no issues, indicating that the deployment runs without errors. The output has been proven to log the success of the deployment.

Accepted: As a DevOps engineer, I want to implement the dependency scanning tools for our CI/CD pipelines.
AC: the output log has been documented and shows the desired behaviour.

Accepted: As a DevOps engineer, I want to research which scanning tools can be used the specific project
AC: The selected tool is mentioned in the documentation.The reasoning of why the specified tool has been chosen has been documented.

Accepted: As a DevOps engineer, I want to learn about the chosen scanning tool for further implementation in our CI/CD pipelines
AC: The key elements to learn has been documented. The Official documentation has been linked.

Accepted: As a DevOps engineer, I want to check if the output of the security tools can be formated into a json file
AC: The documentation in relation to the specified tools has been checked, looking for an option in its output format. The option that formats the output into a json file has been documented. The official documentation about the specified tool and command has been linked.

Accepted: As a DevOps engineer, I want to check if the output can automatically be stored in an specific and accessible place for our dashboard

AC: Research about gitlab artifacts, and how they can be extracted. Research of what kind of authorisation might be required. Document the process.

Accepted: As a DevOps engineer, I want to ensure the Juice Shop Pipeline is refined, so that the vulnerability dashboard can obtain all the reports needed without issue
AC: Check that the stages share their given artifacts to the following requiring stages. Check that the sec. tools reports are given in JSON format in the juice pipeline Document the errors found, if any.

Accepted: As a DevOps engineer, I want to process the security reports fetched from GitLab API so they can be displayed meaningfully in the dashboard.

AC: JSON reports parsed correctly (SAST, SCA, Container, DAST, Fuzzing). Dashboard groups issues by severity, file, tool, and CWE.False positives handled gracefully (e.g., ignored via config). Dashboard visually separates tools and categories.

Accepted: As a DevOps engineer, I want to fetch security reports from GitLab's API using credentials and config stored in the dashboard.

AC: The latest pipeline ID is fetched dynamically. Artifact download API iscalled securely and asynchronously. Fetched data is stored for rendering.

Accepted: As a DevOps engineer, I want the dashboard to display errors such as failed API connections or missing reports so users are aware of issues.

AC: UI displays clear, user-friendly error messages.Covers: invalid token, missing artifacts, empty pipelines. Messages disappear when issue is resolved.

Accepted: As a DevOps engineer, I want to complete the final documentation for delivery, so other teams can install, configure, and use the dashboard independently.

AC: README includes: purpose,personas, setup guide, Screenshot of working dashboard. List of supported tools and CI/CD integration.

Accepted: As a developer, i want to develop the SAST report page, so that i can show it on the Vulnerability dashboard

AC: The report from the pipeline is reachable as a .json The artifacts are pulled successfully from the API of gitlab. the function of sast in the render.js can filter the report and organize it this is applicable for both the sast semgrep and the gitleaks each one of them has its own button to show it when clicked.

Accepted: As a developer, i want to develop the DAST report page, so that i can show it on the Vulnerability dashboard

AC: The report from the pipeline is reachable as a .json at the end point of the gitlab API.The artifacts are pulled successfully from the API of gitlab. in the render.js.the function of dast in the render.js can filter the report and organize it. this is applicable for both the sast semgrep and the gitleaks.each one of them has its own button to show it when clicked. the solution and the error area at the table are equal in width.

Accepted: As a developer, I want to implement the Fuzzing section of the vulnerability dashboard

AC: The report from the pipeline is reachable at the end point of the gitlab API. The artifacts are pulled successfully from the API of gitlab. in the render.js. The function of fuzzing in the render.js can process the report. The dashboard shows the last pipeline output log of the fuzzing tool

Accepted: As a developers, I want to implement the container scanning section of the Vulnerabilty dashboard

AC: The report from the pipeline is reachable as a .json at the end point of the gitlab API. The artifacts are pulled successfully from the API of gitlab. in the render.js. the function of container scanning in the render.js can filter the report and organize it. the dashboard shows the last pipeline output log of the cont-scanning tool.

2.4 Dictionary of terms (Written by Aadit)

Artifacts: Output files (like JSON or HTML reports) from CI/CD pipelines for sharing work between pipeline stages or obtaining those files for informational purposes.
CI/CD: Continuous Integration/Continuous Deployment — automated process of building and testing code.
SAST: Static Application Security Testing — analyzing source code for vulnerabilities without running it.
DAST: Dynamic Application Security Testing — analyzing running applications for vulnerabilities.
Fuzzing: A security testing technique that inputs random data to find bugs or vulnerabilities.
VM: Virtual Machine — a software-based emulation of a physical computer used to run isolated environments.
AC: Acceptance Criteria — specific, measurable conditions that a system or feature must meet to be considered complete or successful.
SCA: Software Composition Analysis — scanning third-party components for vulnerabilities.
GitLeaks: Tool to detect secrets and sensitive information in code repositories.
Pipeline: A series of automated steps (build, test, deploy) executed in CI/CD.

3 Architectural documentation

3.1 System architecture and design (Written by Antonio Huesa Guardiola)

3.1.1 CI/CD Pipelines (Written by Aadit)

For both our pipelines, we define the following non-functional requirements:

- Each job communicate with each other the necessary files and data through shareable Gitlab artifacts to make the execution of the pipeline as little time consuming as possible.

- Each job executes the specific security tool through its command using either the official docker image of the tool if exist, or the executable of the tool.

- Each job will give a clear output of the results, showing that the specific task of the job was a success or fail, explaining the cause of it.

Considering those, both pipelines have the same implementation of these requirements:

- The different stages are defined in the configuration file of each pipeline to set an ordered structure of jobs that have to be executed.

- Each job is filled with the needed commands of the tools that should be executed in the job.

- Artifacts are created and shared if necessary.

3.1.2 Vulnerability dashboard (Written by Aadit)

For our vulnerability dashboard, we define the following non-functional requirements:

- The System gives constant feedback of the state that it is in, displaying the content requested, displaying a loading animation in case the content cannot be displayed yet and changing buttons scale or color by the user interaction.

- The System fetches the data of the security tools using the GitLab API, searching for the last pipeline run, obtaining its jobs, and fetching each specific job's artifacts.

- The System process the jobs' artifacts (parsing them into a JSON if possible to ease the process), filtering all its important content, for example the type of error, severity, description, etc. displaying such content maintaining that organization.

- The System allows the user to set its target configuration in a specific section of the dashboard, where all the sensible details, such as the GitLab project's access token, are hidden, toggling its visibility via a checkbox next to it. Once the user confirms the configuration pressing the confirmation button, the content filed is saved in a JSON file for its use.

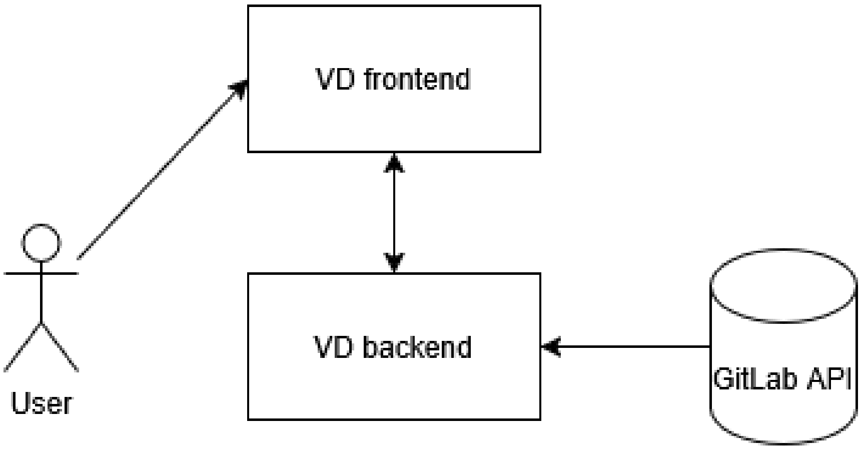
- The System allows selecting which security tool report to see via a series of buttons in the header of applications.

These requirements have been prioritized based on its relevance; the more important the requirement is for the application core functionality, the more priority it has to be implemented.

They have been implemented using the stack chosen, electron, a JavaScript framework that lets developers create desktop applications using Js, HTML and CSS by embedding Chromium and node.js into its binary.

Taking that into account, for the implementation of most requirements, basic JavaScript functions as "fetch()" have been used, in this case to obtain the artifacts from the Gitlab API and process them properly. Other parts of the stack as HTML and CSS have been use to create the basic interface of the application and give clear feedback to the user's actions.

The System architecture is represented in the next scheme:



Pic 1.0 "Vulnerability dashboard system architecture scheme"

- The user interact with the application through the interface, the frontend of the application.

- The frontend and the backend of the application communicate with each other to display the proper content, and, process the required data for it.

- The backend obtains the data mentioned from the GitLab API, fetching the last artifacts of the pipeline if the jobs were succesful.

3.2 Human-machine interface (Written by Aadit Karnavat/ Edited by Beshoy)

Are there any requirements for the MM interface?

3.2.1.1 Interface Requirements

-Accessible layout with buttons for each tool (SAST, DAST, DEPENDENCY SCANNING.)

-Uses JS dynamic rendering (via render.js) for updating pages based on fetched JSON data.

-Allows interactive parsing of reports (severity filtering, etc.)

-Accessible layout with buttons for each tool (SAST, DAST,etc.)

-Uses JS dynamic rendering (via render.js) for updating pages based on fetched JSON data.

-Allows interactive parsing of reports (severity filtering, etc.)

What does the vulnerability dashboard look like? Justify your design decisions.

- High contrast for readability and accesibility
- The choices are broken down into sections (buttons) to not overwhelm the user (cognitive load reduction)
- According to UX rules users used many other websites and expect your website to work the same (the interface is similiar to a simple dashboard) „Jakobs law of UX“
- The header has shadows to make it feel clooser to the user
- Information is grouped into a table relevant according to gestalt principles (proximity and similarity)

4 Test documentation (Written by Aadit)

4.1 Status of the test objectives

4.1.1 Pipeline ToDo List

Test objective	Status / Explanation
Functional correctness of the pipeline	All stages ran successfully from build to deploy to DAST and Fuzzing
Functional correctness of the Vulnerability Dashboard	Reports fetched and parsed correctly in JS

Usability of the technical Documentation	Readable YAML comments and stage logic in CI file

Overall assessment of the achievement of the test objectives:

4.1.2 Pipeline Juice Shop

Test objective	Status / Explanation
Functional correctness of the pipeline	Reports generated via npm audit & retire.js
Functional correctness of the Vulnerability Dashboard	JSON outputs parsed and shown via render.js
Usability of the technical Documentation	Well-documented functions and branch comments

Overall assessment of the achievement of the test objectives:

-Achieved successfully. All CI/CD stages from build to deploy executed as expected. Tools such as SonarQube (SAST), OWASP Dependency-Check (SCA), Gitleaks (Secret Detection), Trivy (Container Scanning), and OWASP ZAP (DAST) were properly integrated. Artifacts were uploaded correctly, and results were fetched without failure.

-Fully functional for stages where reports were available. The SAST and SCA stages were properly represented in the dashboard. However, a placeholder was left for final SAST JSON integration. Overall, the system can visualize vulnerability data effectively and interactively

-The project meets all core test objectives with a high level of integration, functionality, and security coverage. With minor polishing in user experience and documentation, it's ready for final delivery.

4.2 Error status after successful test

4.2.1 Pipeline ToDo List

	Open	**Closed**
Blocking	**0**	**0**
High	**1**	**3**
Normal	**2**	**4**
Low	**0**	**3**
Total	3	10

4.2.2 Pipeline Juice Shop

	Open	**Closed**
Blocking	**0**	**0**
High	**2**	**5**

Normal	**2**	**41**
Low	**1**	**2**
Total	5	11

5 Acceptance documentation

The following templates are to be used in the acceptance procedure:
BZA - Provision for acceptance
Acceptance protocol

6 User documentation (optional) (Written by Aadit/ Edited by Beshoy)

1. Open the index.html from the vulnerability-dashboard folder in your browser. Make sure your EduVPN is on (crucial)

2. Configure GitLab Connection:
-Click on "Settings".

-Enter the following:

-GitLab Personal Access Token

-Project ID (ToDo app or Juice Shop)

-GitLab Host (e.g., https://lv-gitlab.intern.th-ab.de)

3. Viewing Vulnerabilities:

-Select a tool (e.g., SAST, DAST, SCA) from the dashboard.

-The dashboard fetches the latest available report and displays it.

-Use filters (if present) to group by severity or category.

4. Error Handling:

-If the tool report is missing, you will see a warning.

-Ensure pipeline for that tool completed successfully and artifacts are available.

7 Summary (Written by All team members)

This project implemented two complete CI/CD pipelines for:

- ToDo App (Java Spring Boot + Maven)
- OWASP Juice Shop (Node.js + npm)

Security Coverage:

- Secret Detection: Gitleaks
- SAST: Semgrep, SonarQube todolist only)
- SCA: OWASP Dependency-Check, npm audit, retire.js
- Container Scanning: Trivy
- DAST: OWASP ZAP (both pipelines)
- Fuzzing: jsfuzz (Juice Shop), JavaFuzz (ToDo)

Vulnerability Dashboard:

- Fetches and processes JSON reports from GitLab API
- Displays categorized vulnerability info per tool
- Supports user configuration and error feedback
- Built for use by Developers, Security Champions, and Technical Writers

Agile & Teamwork:

- Roles included Scrum Master, Product Owner, Security Champion
- Work organized in epics, user stories, and sprints
- Peer collaboration, code reviews, sprint reviews

Impact:

- Demonstrated secure software development lifecycle (SSDLC)
- Encouraged shift-left security via early detection
- Unified visibility of vulnerabilities for faster triaging

8 Appendix (written by Aadit)

OWASP.org — Tool Documentation

GitLab CI/CD Documentation

Semgrep Registry

npm Audit and Retire.js docs

SonarQube Documentation

Docker & Trivy Scanner Guide

JavaFuzz (https://github.com/fuzzitdev/javafuzz)

OWASP Juice Shop Documentation

Bibliog

List of illustrations (optional)

Acceptance report

The acceptance was carried out on xx.xx.202x. The provision for acceptance took place on time.

The following defects were identified during acceptance.

No.	Defect	Description	Defect category
1			

Defect category

High (1): **No acceptance** can take place unless the defect is remedied.

Medium (2): Can lead to **conditional acceptance**. In the latter case, there would be a conditional acceptance with a request to remedy the defects subsequently by a date to be agreed.

Low (3): Can be remedied during maintenance.

We ask you to remedy the defects of the priority "medium" by xx.xx.202x EOD. Low priority defects will be remedied via the maintenance process.

We hereby issue a

○ Acceptance
○ Conditional acceptance
○ No acceptance.

Yours sincerely

Timea Illes-Seifert and Marie Oetzel