URBANITE

Supporting the decision-making in urban transformation with the use of disruptive technologies

Deliverable D5.3

Integration strategy

Editor(s):	Iñaki Etxaniz, Juncal Alonso
Responsible Partner:	Tecnalia
Status-Version:	Final - v1.0
Date:	30.09.2020
Distribution level (CO, PU):	PU

Project Number:	GA 870338
Project Title:	URBANITE

Title of Deliverable:	Integration strategy
Due Date of Delivery to the EC:	30/09/2020

Workpackage responsible for the Deliverable:	WP5 - URBANITE ecosystem integration and DevOps
Editor(s):	Tecnalia, Engineering
Contributor(s):	Iñaki Etxaniz (Tecnalia), Giuseppe Ciulla (ENG)
Reviewer(s):	Giuseppe Ciulla (ENG)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP6

Abstract:	This document details the strategy and steps to integrate the URBANITE ecosystem. It specifies the different environments that will be generated (development, integration and production) and the tools that will be used for the continuous integration and continuous delivery approach.
Keyword List:	Development environment, integration, testing, requirements, GUI.
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/ The document itself is delivered as a description for the European Commission about the released software, so it is not public.
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	04/05/2020	First draft version	Tecnalia
v0.2	24/05/2020	Included 3. Validation strategy	Tecnalia
v0.3	05/06/2020	Included 2.2 Version Control and task management	Tecnalia
V0.4	25/06/2020	Included 2.3 Deployment management	Tecnalia
v0.5	14/07/2020	Included 4.GUI integration	Engineering
V0.6	30/07/2020	General setup	Tecnalia
V0.7	11/09/2020	Internally reviewed	Engineering
V0.8	16/09/2020	Updates according to internal review	Tecnalia
V1.0	30/09/2020	Ready for submission	Tecnalia

Table of Contents

List of Figures

List of Tables

Terms and Abbreviations

CI/CD	Continuous Integration/Continuous Deployment
DevOps	Development and Operation
DoW	Description of Work
EC	European Commission
KR	Key Result
QA	Quality Assurance
SCM	Source Code Management
Sw	Software
WP	Work Package
GUI	Graphical User Interface

Executive Summary

URBANITE is a collaborative research and innovation action whose outcomes are mainly software-based. These outcomes will be implemented in a collaborative manner by different development teams from various partners. In order to manage the development environments, and the integration of the different software components in on-time releases, the proper DevOps strategy and processes have to be defined and set up.

This document facilitates partner cooperation in the technical work packages, providing guidelines for partners during the development (WP2, WP3, WP4), integration (WP5) and validation (WP6) processes in URBANITE. This deliverable is the result of *Task 5.3 - Continuous Integration and DevOps approach.*

First, the project DevOps methodology and the development, integration and production environments that will be used for managing the development process, are presented and discussed in detail, along with the proposed supporting tools/technologies in each environment. The GitLab, provided by Tecnalia, will be used as version control system, hosting both private and public repositories, whereas Jenkins is selected as the deployment management tool of the project. Jenkins's main functionalities and a comparison with the GiLab CI tools is presented. Docker will be used as the containerization technology to provide infrastructure as specification in URBANITE.

Next, the integration stage is defined, being its main activities Build, Run and Test. This stage focus on compiling the code, managing the interactions among the different components, and performing the integration test reports. The validation of the functional requirements is performed after the integration. Requirements are elicited over the different technical Work Packages, and presented, prioritized and managed in a related deliverable (D5.1 and further).

Finally, the strategy followed to provide a single GUI for URBANITE as a wrapper of the different component is presented. A design system - collection of colour scheme, reusable components with a common style (buttons, icons, fonts), etc. that can be assembled to build several applications is also proposed. The suggested technologies to build the web based GUI are Angular **(**JavaScript framework), Nebular (UI library for Angular), and ngx-admin (dashboard based on Angular), capable of show maps and charts, which is used to present a template for the system dashboard. Two strategies for integrating components in the system are introduced.

The most tangible results of the DevOps methodology and integration tasks in URBANITE will be the DevOps infrastructure (to be released in M12 as deliverable D5.6) and the sequential URBANITE Ecosystem (M15 and following).

1 Introduction

1.1 About this deliverable

This document details the strategy and steps to follow to integrate the URBANITE ecosystem. It is provided as a baseline reference for partners during the development, integration and validation processes in URBANITE. It facilitates partner cooperation in the technical work packages by defining the tools, set of rules and guidelines for the management of the development, integration and delivery of project's results (source code repositories, binary repositories, construction method, etc.). It also presents the strategy for the validation and the prioritization of the functional requirements.

This deliverable is the result of *Task 5.3 - Continuous Integration and DevOps approach.*

1.2 Document structure

The document is organized into four (4) main sections plus a conclusion chapter, with the first section presenting the deliverable's objective and structure.

The second section introduces the different environments and procedures that will be used for software implementations in URBANITE, including the setup of the different settings -namely, development, integration and production,- the description of the various procedures defined for the software life cycle of the project and finally, the tools envisioned to be adopted in each phase. Also, some development conventions and best practices are detailed.

Next, the integration and validation strategy to be followed in URBANITE is described. For this, the approach and processes for the integration, as well as the requirements validation mechanism, are presented.

The fourth section is devoted to Graphical User Interface (GUI) component, explaining which strategy is adopted to provide a unique GUI for the whole URBANITE ecosystem, which are the technologies and tools to be used in the implementation.

Finally, the conclusion section resumes the most relevant points of the document. The document ends with the references and appendixes.

2 DevOps development environment and procedures

2.1 DevOps infrastructure

This section presents the infrastructure and tools planned to be used internally for the development and operation. The DevOps approach requires the deployment of a pipeline that consists of the stages an application goes through, from development to delivery and production.

The URBANITE iterative and incremental approach mandates adopting a development and deployment process able to support it fully. That is why the project will adopt a **DevOps** approach in the development of all KRs. DevOps integrates development and operations into a single-minded entity with common goals: high-quality software, faster releases and improved user satisfaction. DevOps also incorporates several agile principles, methods, and practices, such as continuous delivery, continuous integration, and collaboration [1].

The different KRs, outcomes of URBANITE [2], are composed of several software components that will be implemented by various partners following heterogeneous technologies.

The DevOps approach requires the setup of a development and delivery pipeline that consists of the stages an application goes through from development through production. In URBANITE, the DevOps approach will be structured in three environments, as depicted in the figure 1. These environments are 1) development, 2) integration, 3) production or piloting.

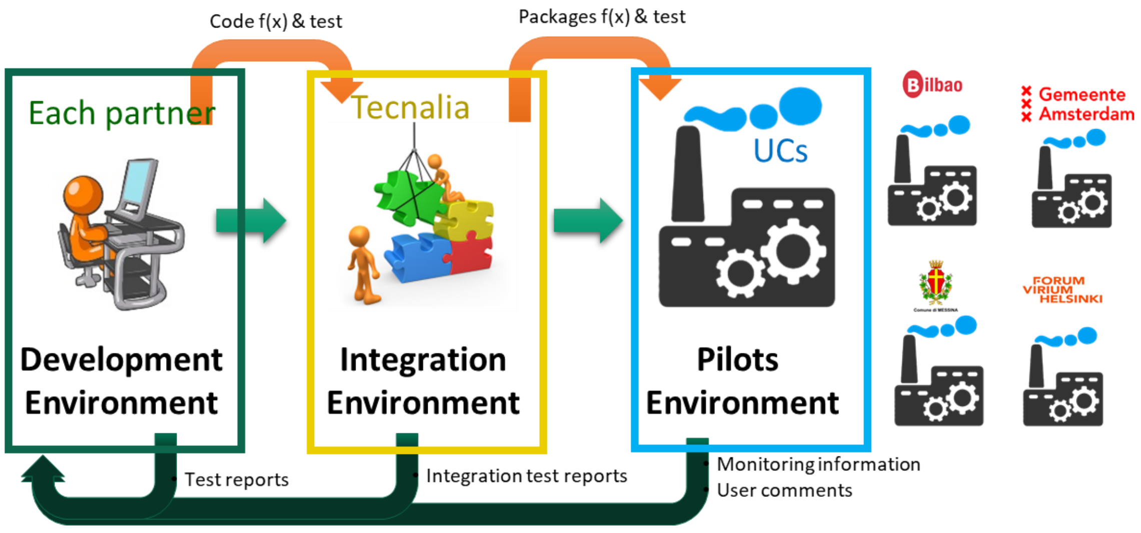


Figure 1. The three environments in URBANITE.

- The *Development* stage envisages the availability of a development environment that provides tools to write and test code and support collaborative development (e.g., source control management, work-item management, collaboration, unit testing, project planning). Possible tools and development approaches are: *GitLab*[[3] or *Git* [4] as the version control system, *Apache Maven* [5] to manage project's builds, reporting and documentation, and containerization technology to have applications running in self-contained units that can be ported across platforms (e.g., Docker [6]).
- The *Integration* stage focuses on compiling the code and performing the unit test and integration test reports. This stage also includes the availability of a common storage mechanism for the binaries created, as well as the assets required to deploy the applications (e.g., configuration files, infrastructure-as-code files, deployment scripts). Possible tools to support this stage are *xUnit* [7] as unit testing framework, *Jenkins* [8] to support continuous integration and *Apache Maven* [5] for building instructions.
- The *Pilots Environments* envisage tools and features for application environment management and provisioning such as *Chef* [9], *Docker*, *Puppet* [10], as well as tools like *Logstash* [11] to speed up the feedback loop; *osTicket* [12] or *Trac* [13] as ticket or issue tracking system.

The DevOps approach that will be followed in URBANITE includes the use of version control tools (e.g. Github or GitLab), continuous integration tools (e.g. Maven for managing dependencies and Jenkins) as well as micro-services components deployment (e.g. Docker containers for easier portability and reconstruction of the solution).

In the following sub-sections (2.2, 3, 4 and 5) the different tools to be used in the URBANITE DevOps infrastructure will be presented, describing their use.

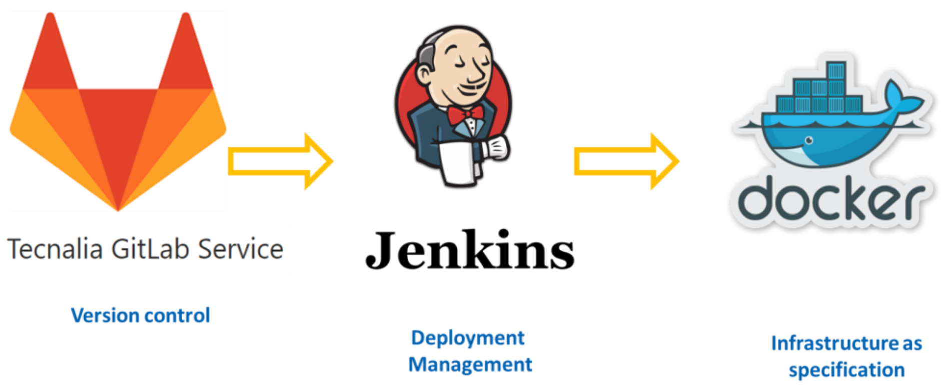


Figure 2. Main tools used in URBANITE DevOps infrastructure.

2.2 Version control and task management

2.2.1 Software repository

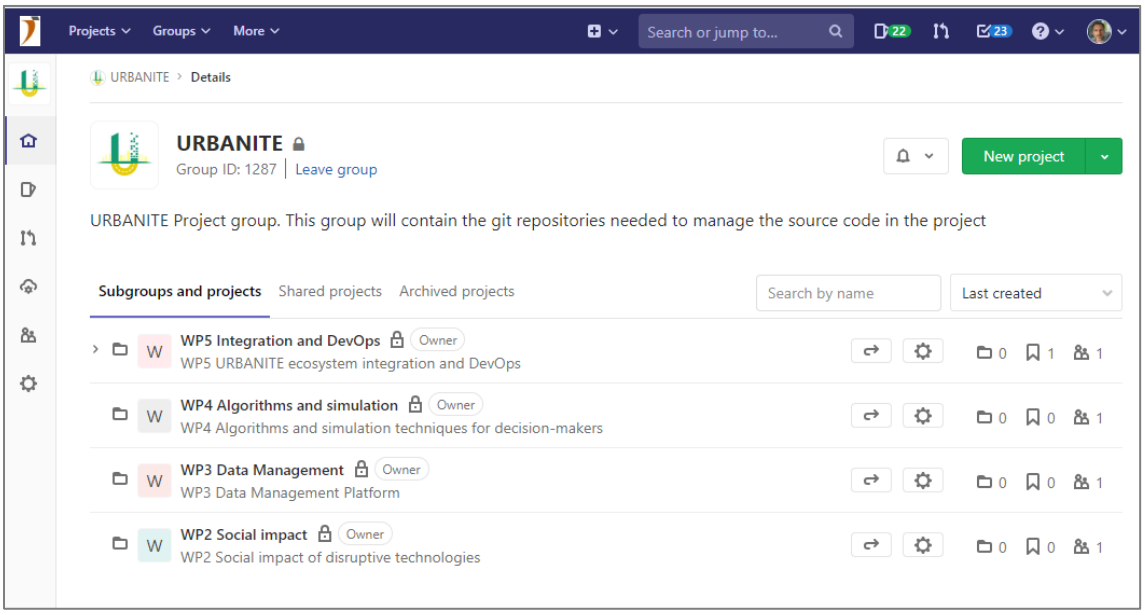
The technical work packages of URBANITE will use **GitLab** [3] to manage source code and for revision control. GitLab is a Web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git [4] and adds additional proprietary features.

The URBANITE GitLab[1] is offered by Tecnalia and will host both **Private** and **Public** repositories.

- The **private** repositories will be used to host the initial stages of the different components of the project until they are mature enough. Besides, the private repositories will also be used to store repositories required by the pilots to develop their pilot-oriented specific source code and resources.
- Then, those components to be released under open source license will be deployed in the **pubic** repositories, where they will be publicly available.

As established in the different individual and collaborative dissemination strategies, any proprietary implementations of the partners will remain in private repositories. Other URBANITE components will become public as defined in the Open Source model followed in the project and expressed in the exploitation strategy [14].

[1] https://git.code.tecnalia.com/urbanite



URBANITE
Group ID: 1287 | Leave group

URBANITE Project group. This group will contain the git repositories needed to manage the source code in the project

WP5 Integration and DevOps (Owner) — WP5 URBANITE ecosystem integration and DevOps	0	1	1
WP4 Algorithms and simulation (Owner) — WP4 Algorithms and simulation techniques for decision-makers	0	0	1
WP3 Data Management (Owner) — WP3 Data Management Platform	0	0	1
WP2 Social impact (Owner) — WP2 Social impact of disruptive technologies	0	0	1

Figure 3. Private GitLab repository of URBANITE, organized by Work Packages.

2.2.2 Development tracking

The URBANITE platform will be comprised by a set of software components that form the KRs. Development issues are a fundamental medium for collaborating on ideas and planning work, so it is envisioned that the project will use a tool for managing such information.

Among the many tools available to manage issues, a logical decision is to choose the GitLab issue tracker, which comes integrated with the source code management in use in the project.

GitLab issue tracker [15] is a tool for managing the development, for collaboratively developing ideas, solving problems and planning work. Issues are always associated with a specific project, but if there are multiple projects in a group, one can also view all the reported problems collectively at the group level.

Everyday use cases for issues include (i) Tracking tasks and work status; (ii) Accepting feature proposals, questions, support requests, or bug reports; (iii) Elaborating on new code implementations.

The issues management in GitLab contains a variety of content and metadata, enabling large flexibility in how they are used. Each issue can include, among others, the following **attributes**:

- Content
    - Title
    - Description and tasks
- People
    - Author
    - Assignee(s)
- State
    - State (open or closed)
    - Health status (on track, needs attention, or at risk)
    - Confidentiality
    - Tasks (completed vs. outstanding)
- Planning and tracking

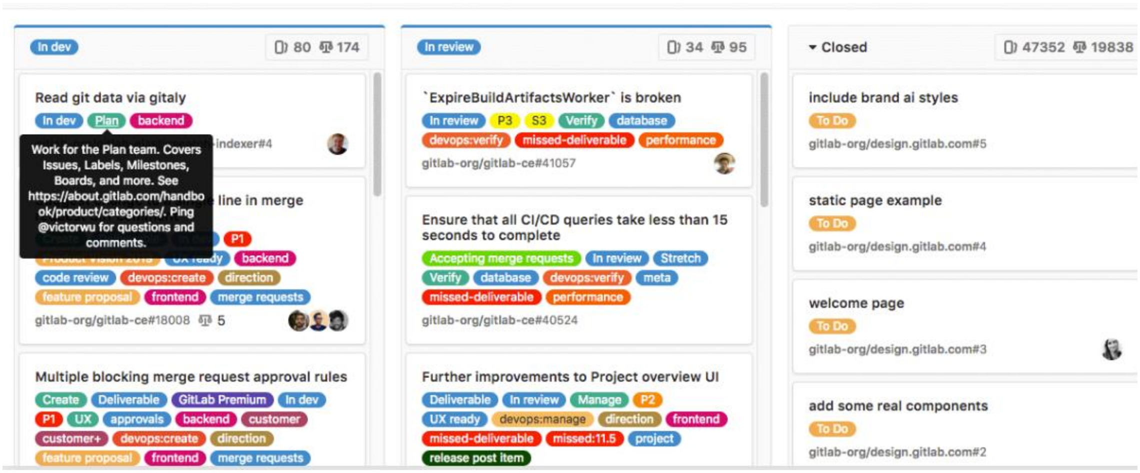
- Milestone
- Due date
- Weight
- Labels
- Linked issues

An Issue can be seen in Kanban boards with columns displaying issues based on their labels or their assignees. They offer the flexibility to manage issues using highly customizable workflows. You can reorder issues dragging it to another column. The entire board can also be filtered to only include issues from a particular milestone or an overarching label.

An issue can be linked to others, with one of these relations: relates to, blocks, or is blocked by. The issue has not a "**type**" properly said, but this can be implemented through the "**labels**" features, that can be defined and assigned to issues. Among the default labels, we can found the following: *Bug, Discussion, Documentation, Enhancement, Suggestion* or *Critical.*

Labels are part of issue boards, and permits:

- Categorize epics, issues, and merge requests using colors and descriptive titles.
- Dynamically filter and manage epics, issues, and merge requests.
- Search lists of issues, merge requests, and epics, as well as issue boards.


In dev 80 174

Read git data via gitaly
In dev Plan backend
indexer#4

Work for the Plan team. Covers Issues, Labels, Milestones, Boards, and more. See https://about.gitlab.com/handbook/product/categories/. Ping @victorwu for questions and comments.

line in merge
P1
backend
code review devops:create direction
feature proposal frontend merge requests
gitlab-org/gitlab-ce#18008 5

Multiple blocking merge request approval rules
Create Deliverable GitLab Premium In dev
P1 UX approvals backend customer
customer+ devops:create direction
feature proposal frontend merge requests

In review 34 95

\`ExpireBuildArtifactsWorker\` is broken
In review P3 S3 Verify database
devops:verify missed-deliverable performance
gitlab-org/gitlab-ce#41057

Ensure that all CI/CD queries take less than 15 seconds to complete
Accepting merge requests In review Stretch
Verify database devops:verify meta
missed-deliverable performance
gitlab-org/gitlab-ce#40524

Further improvements to Project overview UI
Deliverable In review Manage P2
UX ready devops:manage direction frontend
missed-deliverable missed:11.5 project
release post item

Closed 47352 19838

include brand ai styles
To Do
gitlab-org/design.gitlab.com#5

static page example
To Do
gitlab-org/design.gitlab.com#4

welcome page
To Do
gitlab-org/design.gitlab.com#3

add some real components
To Do
gitlab-org/design.gitlab.com#2

Figure 4. An example of the Kanban boards of issues with labels in GitLab.

2.2.2.1 Version release

URBANITE project release schedule will be as follows:

- A new major version release will be produced at each milestone as initially planned in the URBANITE Proposal DoW [2], at months M15, M27 and M33.
- In general, the minor version will be increased in each plan-build-test iteration of the integration strategy proposed (see section 3.1). Usually, this relates to a new feature or enhancement added to the component.

Versions are identified using a standard triplet of integers in the form: *Major.Minor.Interim* (e.g. 3.2.11). In between a Major or Minor version, partners can generate any number of interim releases.

In URBANITE, software versions will be defined by **labels**. Labels can be defined at project or group level. Major and Minor releases will be created for every project in the issue tracker tool. This will allow an explicit assignation of tasks scheduled for each specific version.

2.3 Deployment management

One of the main objectives of the DevOps philosophy is to enhance the flow between the development stage and the operation one, to decrease the production times. There are many approaches to reach such objective: Kaizen [16], a concept referring to business activities that continuously improve all functions and involve all employees from the CEO to the assembly line workers); Lean [17], whose translation of its manufacturing principles and practices to the software development domain with the support of a pro-lean subculture within the Agile community Lean software development is; or SixSigma [18], a set of techniques and tools for process improvement, that Jack Welch made central to his business strategy at General Electric in 1995. All those methods identify the systematic automation of repetitive tasks as one of the main ways of achieving the mentioned time reduction from development to production.

There exist multiple tools for automating tasks, like Bamboo [19], Travis [20] or CruiseControl.NET [21], which tie automated builds, tests, and releases together in a single workflow. Two of the most extended automation tools are Jenkins [8] and GitLab CI/CD [22], a tool built into GitLab for software development through the continuous methodologies: Continuous Integration (CI), Continuous Delivery (CD) and Continuous Deployment (CD)- so we include a brief comparison of both tools below. On their basis, they do pretty much the same: to allow you to execute shell/bat scripts, defined in a YAML file, on a server/Docker image. These scripts are written in a YAML[2] file in CI/CD, while in Jenkins the *jenkinsfile* uses Groovy[3] syntax.

Jenkins

Job chaining and orchestrating in Jenkins can be more straightforward, because of the UI, than in GitLab via YAML (calling curl commands). Starting with Jenkins 2.0, the pipeline capability - which has been available as a plugin before this version- is built into Jenkins itself. Jenkins pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code". The definition of a Jenkins pipeline is typically written into a text file (Jenkinsfile) which in turn is checked into a project's source control repository.

Jenkins is very configurable because of all the available plugins. The downside of this is that it can become a nightmare of plugins. Also, as plugin are made from third parties, many of them are no longer actively maintained and as a result, they may be incompatible with later versions of Jenkins or other plugins. Plugins can become expensive to maintain, secure, and upgrade.

The plugins available for Jenkins allow you to visualize all kinds of reporting, such as test results, coverage and other static analyzers. Another typical plugin, that provides a nice GUI to Jenkins is Blue Ocean[4].

Users need to host and setup Jenkins by themselves. Self-hosting provides a safe location to store key environment variables since it is the user in charge of the server and environment. The disadvantage is that this results in both a high initial setup time, as well as time sunk into maintenance over a project's duration.

[2] https://yaml.org
[3] http://www.groovy-lang.org/
[4] https://www.jenkins.io/projects/blueocean/

Jenkins technology is very mature (has been in development since 2004) and is one of the most popular tools of its kind, with a lot of documentation and resources available.

Being a Java application, it can be installed under any OS: Windows, Linux, and macOS. On the other hand, JNLP slaves also enrich the cross-platform build support for its agents.

Thanks to the credentials API plugin, Jenkins provides encryption of secrets

GitLab CI/CD

On the other side, GitLab CI/CD is naturally integrated in GitLab. The setup of GitLab CI for projects hosted on GitLab -since it uses the GitLab API for setting up hooks- can be done in one click. Its maintenance is easy as there are not plugins. It is part of the repository, and scaling the runners is simple. You can have branches with different yaml files for different release branches. You can create pipelines using *gitlab-ci.yml* files and manipulate them through a graphical interface.

GitLab CI is a visual management tool, all members of the working team -including the non-technical ones- can have a quick access to applications life cycle status. Therefore, it can be used as an interactive and operational dashboard for release management.

The tests of GitLab CI run parallel to each other and are distributed on different machines. Developers can add as many devices as they want or need, making GitLab CI highly scalable to the development team's needs.

It has no Windows support, but it's possible to use a Bitnami[5] stack (a packaging that facilitates getting open source software tools up and running on many platforms, including laptops, Kubernetes and all the significant clouds).

Combining GitLab CI and Jenkins, although it might sound redundant, is possible and quite powerful: while GitLab CI orchestrates (chains, runs, monitors...) pipelines and one can benefit its graphical interface integrated to GitLab, Jenkins runs the job and facilitates dialog with third-party tools. This setup can be convenient when dealing with complex applications with heterogeneous technologies, having each a different build environment, and still need to have a unified application life cycle management UI.

A summary of the comparison between Jenkins and CI/CD is shown in the following table:

Table 1. Comparison of tools: Jenkins vs GitLab CI.

	Jenkins	**GitLab CI/CD**
License	OSS (MIT license)	Community Edition: OSS (MIT) Enterprise Edition: Proprietary
OS	Windows, Linux, macOS (Java based)	Linux (Windows via Bitnami)
Hosting	Self-managed	Self-managed,

[5] https://bitnami.com/

		GitLab SaaS
GUI	Yes (via BlueOcean plugin)	Yes
Migrate from Jenkins / CI	-	Instructions provided
Features		
CI/CD	Yes	Yes
Container registry	-	Yes
API	Yes	Yes
Pipeline graphs	Yes	Yes
Scheduled triggering of pipelines	Yes	Yes
Environments and deployments	-	Yes
Group-level variables	-	Yes
Automatic Retry for Failed CI Jobs	Yes	Yes
Include external files in CI/CD pipeline definition	Yes	Yes

In URBANITE, it is envisioned to use **Jenkins**. The main reasons for this selection are, on the one hand, its open source licensing model, but the availability of professional support services when necessary, the extensibility offered with the plugin ecosystem, and lastly, the previous experience of some partners of URBANITE in its use.

The usage of an automation server such as Jenkins, provides a lot of advantages when sharing the information about the status of the continuous integration tasks, both for the developers and for the users.

2.3.1 Main functionality

The Jenkins server provides multiple functionalities, extendable by adding plugins (more than 2000 plugins available). In this section, some of these functionalities, which are relevant for URBANITE project, will be described.

We will focus only on some stages of the DevOps cycle, like registering the component, continuous integration, debugging, modifying the component and deleting it.

During the continuous integration stage, the automation server should support different development and testing cycles (see section 3.1 for more details). The needed functionalities for such a development strategy are:

- Creation of the automatization tasks to systematize DevOps cycle and accelerate the incorporation of the changes and modifications into the production environment.
- Grouping the tasks into different groups, to manage complex development.
- Review of the execution log, so that we have mechanisms to identify the causes of a failure in the automation task.

- Keep the results of past executions and check their status, to analyze possible failures.
- Notify the status of the executions to be able to launch further tasks automatically.
- Get the automation scripts from Git so that we can include the integration tasks in the configuration management.
- Delete projects.

Apart from these functionalities, Jenkins can manage the infrastructure itself where the different components are executed (for example, using maven or similar plugins) to perform the integration tests against them.

2.3.2 Integration points

With respect to the integration technologies, in URBANITE it is envisioned to use mainly the REST API provided by Jenkins. Jenkins provides machine-consumable remote access API [22] to its functionalities.

Remote access API is offered in a REST-like style. There is no single-entry point for all features, and instead, they are available under the ".../api/" URL where "..." portion is the data that it acts on. For example, if your Jenkins installation sits at *https://ci.jenkins.io*, visiting *https://ci.jenkins.io/api/* will show just the top-level API features available – primarily a listing of the configured jobs for this Jenkins instance.

When the Jenkins installation is secured, you can use HTTP BASIC authentication to authenticate remote API requests.

This API provides functionalities for:

- Recovering information from Jenkins
- Launching executions
- Creating / Copying Jobs

To launch a simple job without parameters, you merely need to perform an HTTP POST:

JENKINS_URL/job/JOBNAME/build.

A simple example of launching a job with string parameters is:

```
curl JENKINS_URL/job/JOB_NAME/buildWithParameters   \
--user USER:TOKEN   \
--data id=123 --data verbosity=high
```

Python API wrappers

JenkinsAPI[6] and Python-Jenkins[7] are object-oriented python wrappers for the Python REST API, aiming to provide a more conventionally pythonic way of controlling a Jenkins server. It provides a higher-level API containing several convenience functions. Services offered currently include:

- Query the test-results of a completed build
- Get objects representing the latest builds of a job
- Block until jobs are complete

[6] https://pypi.org/project/jenkinsapi/
[7] https://pypi.org/project/python-jenkins/

- Install artifacts to custom-specified directory structures
- Authentication support for Jenkins instances
- Ability to search for builds by subversion revision
- Ability to add/remove/query Jenkins agents

Java API wrappers

The jenkins-rest[8] library provides access to the Jenkins REST API programmatically. It is built using the jclouds[9] toolkit and can easily be extended to support more REST endpoints. In its current state, it is possible to submit a job, track its progress through the queue, monitor its execution until its completion, and obtain the build status. Services currently offered include:

- Endpoint definition (property or environment variable)
- Authentication (basic and API token via property or environment variable)
- Folder support
- Jobs API (build, config, create, delete, description, disable, enable, jobInfo, etc.)
- Plugin manager API (installNecessaryPlugins, list current plugins)
- Queue API (cancel, list queue items, query queue item)
- Systems API (systemInfo)

2.4 Infrastructure as specification

In complex development environments, to ensure the independence of the modules and avoid technical incompatibilities, it is expected that components are allocated in different virtual (or physical) machines.

This approach implies several tasks, for example:

- Requesting and managing new (virtual) machines.
- Installing/configuring the operating system in each machine.
- Configuring the access to those machines, including opening the corresponding ports (e.g. SSH access)
- Configuring the platform and the requirements of each of the software components manually.

The **containers** technology allows the definition of separate spaces in a single machine -that can be virtual or physical- optimizing the computational resources. The container provides this separation both at file system and communications level. At the same time, container-based technologies (such as Docker [6] or Warden [23]) allows the explicit provision of the configuration of the containers. Things like baseline operating system, packages included, initial content, etc. This allows the instantiation of the several containers with exactly the same initial characteristics.

Furthermore, some container technologies support the container's registry usage, where the developed containers can be uploaded so that other team members can download, use, and test them with a small set of instructions.

[8] https://github.com/cdancy/jenkins-rest
[9] https://jclouds.apache.org/

In URBANITE it is envisioned to use **Docker** as containerization technology. Some reasons for this election are:

- Open Source technology, with extensive and growing users' community.
- Professional support is provided if required
- It provides a public registry for the containers; alternatively, it allows you to create your own private one.

2.4.1 Main functionality

The Docker server provides a lot of functionalities, but, in this section, we will only refer to those relevant for URBANITE, focusing on the DevOps stages of continuous integration, publication, distribution and updating.

In URBANITE it is envisioned to use the following capacities from Docker:

- General
  - Definition of the platform requirements for the components.
  - Containers creation including both the component and the platform requirements for it.
  - Configuration of the containers during its instantiation.
  - Logs communication
  - Containers instantiation
  - Containers instances stopping and deleting
  - Persistency definition
- Communication
  - Containers registration into the registry
  - Project level registry creation
- Distribution
  - Download containers from the registry
- Updating
  - Download containers versions from the registry

2.4.2 Integration points

With respect to the integration technologies, it is envisioned that URBANITE will use mainly the **Maven** plugin for Docker [24]. This technology will allow us to obtain the actions registry (log) of what is happening in the different actions that support the DevOps cycle. These actions are *Build*, *Run*, *Stop*, *Pushing* into the registry, and *Log*.

This registry integrates perfectly with Jenkins and allows to analyze what happened during the execution of the different activities.

The proposed approach introduces some complexity in the integration strategy, given that it requires the knowledge of the three tools employed -Jenkins, Maven and Docker-. But this approach provides a configuration that is editable and adjustable to the needs of each project.

All the configuration files will be stored along with the project and them will be accessible and modifiable by the team working on it.

2.5 Development conventions and best practices

In URBANITE, there are planned three types of deliverables, as presented in the Quality Plan [25]. Among them, the software deliverables (type "other"), which are actually a piece of code corresponding to a Key Result, will be accompanied by a **short technical report** explaining the main functionalities, technical design, downloading information, installation manual and licensing schema of the software. The technical report will also have to follow the provided template for software deliverables.

This section introduces the naming conventions for the Software deliverables:

- Source files will follow the following naming convention:

  eu.URBANITEh2020.modulename.componentname.subcomponentname

- Endpoints naming convention is as follows:

  /URBANITE/[group]/[componentname]/

- Domain names will be:

  ***.dev.URBANITE-h2020.eu (for the development environment)

  ***.URBANITE-h2020.eu (for the production environment)

- Source code files heading shall follow the following format:

```
/*
* Copyright (c) 202x <<Company_name>>.
* All rights reserved. This program and the accompanying materials
* are made available under the terms of the
* <<licensing_schema_to_be_URBANITE>> which accompanies
* this distribution, and is available at
* <<link of the information of the selected licensing schema>>
*
* Contributors:
*
* <<Full Name of the contributor(s)>> <<(Organization Name(s))>>
*
* Initially developed in the context of URBANITE EU project
* www.urbanite-project.eu
*/
```

3 DevOps integration and validation strategy

3.1 Integration stage: Strategy and process

The integration stage will focus on compiling the code, managing the interactions among the different components, and performing the integration test reports. This stage also includes the availability of a common storage mechanism for the binaries created, as well as the assets required to deploy the applications (e.g. configuration files, infrastructure-as-code files, deployment scripts). The tools envisioned to support the integration stage in URBANITE are Jenkins and Maven.

The Integration stage in URBANITE will be in charge of several processes. The source code from the different components, generally developed by different partners, is collected (along with the integration tests), built, run and tested. Once a component has successfully passed them, the component will become a candidate to a release. Then, the software components have to pass a functional testing by a group of experts. When this functional testing is successful, it will become part of an official release. Once the official release has been created, it will be trespassed to the production environment where the user case can test it.

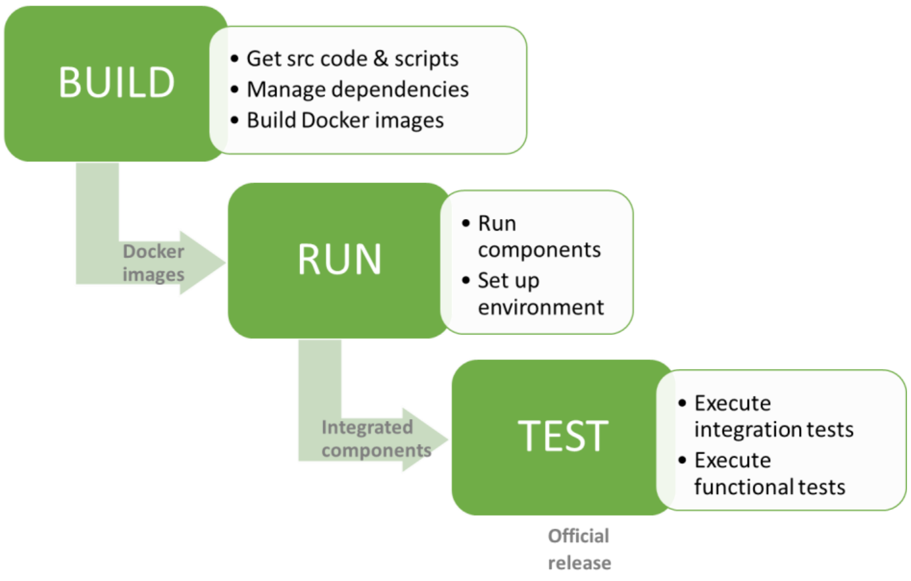


Figure 5. Proposed integration strategy in URBANITE

A more detailed view of the process and involved steps is shown in Figure 5, where the main activities of integration stage can be viewed:

- Phase 1: **Build** - During this phase the Jenkins tasks (functionality and data) for getting the source code from the software repository and building the Docker image will be created and performed. This phase may also include Jenkins tasks for getting specific pre-requisites needed to build the image, such as dependencies to specific libraries or needed endpoints.

  The output of these activities is the docker image of the specific component and the related integration tests, which will have been developed by the component developers' team. The management of the dependencies -needed to compile the components- will be in charge of the Maven client included in the Jenkins framework.

- Phase 2: **Run** - During this phase the Jenkins tasks for running the component and setting up the environment are created and performed.

- Phase 3: **Test** - During this phase, the Jenkins tasks for running the integration tests of the components will be created and performed. The tests will be defined following a layer's approach, from the simplest (without dependencies) to the most complex ones (with dependencies upon the previous tests):

  - 0.0 Testing data loading
  - 0.1 Testing of functions that need data to be executed
  - 0.2 Testing of functions that need 0.1 tests to be executed
  - Etc.

Once all the tests are passed successfully, the next process starts, that includes some additional steps to test the functionality of the integrated system:

- Tag all the components as a candidate to release.
- Manual functional testing (by a group of experts which is defined in the context of URBANITE)

When the component has successfully passed the functional testing, it will become an URBANITE official release, to be validated in the Use Cases.

3.2 Requirements validation: Strategy and process

This section introduces how the different functional requirements of the software components to be developed during URBANITE will be prioritized through the different software releases.

Several functional requirements for the different software components have been elicited in the context of the different technical WPs. The resulting list is covered in deliverable D5.1 [27]. The requirements have been gathered through various requirement elicitation techniques, being the most favored ones: document analysis, expert judgment and brainstorming. The consortium has taken as input the text in the DoA and has transformed it into a requirement format; Brainstorming sessions have been carried out in workshops and teleconferences, while expert judgment has been used to determine the feasibility of the requirements elicited.

After defining them, the prioritization of the different functional requirements with respect to the Milestones will be performed, based on the Use Cases' needs and technical reasons. The result of this process will be reflected in a table in the above mentioned deliverable, which will serve as a plan for the software development activities in each WP, establishing the functionalities that will be developed and validated in each of the development cycles. The table will be updated in subsequent versions of the deliverable D5.1, in every official release in URBANITE.

The prioritization of the requirements will be based on the experience of the partners and the functionalities identified as core. For this prioritization, the following key aspects will be considered:

- Baseline functionalities will be implemented in the initial versions. Complex functionalities depending on the baseline functionalities, will be implemented in the subsequent releases.
- Functionalities with strong dependencies with other tools (i.e. interfaces-related functionalities) will be implemented in early stages of the project, so that the integrated framework can be built up and each component can test the communications with other components.
- Functionalities that are core for the use cases have been prioritized for the initial releases. This initial prioritization has been done based on D6.1 [26], where the initial description of the use cases is included. The priorities will be updated in parallel to the next WP6 deliverables, which will detail more and eventually specify new UC requirements. The UC validation process will be defined in the context of WP6.

The prioritization of the functional requirements will lead to an incremental implementation, grouping them by the milestones at months M15, M27 and M33

The main properties included in this table are:

- **Req id**: This is the unique identification of the functional requirement, used in the work package document where has been defined during the elicitation process.
- **Component**: This is the name of the software component/subcomponent part of the specific key result.
- **Partner**: This is the acronym for the partner responsible for the development of the software component.
- **Release**: This term indicates the dates of the different official releases for the software deliverables.
- **WP**: This is the WP number where the software component will be developed.
- **No**: Number of the functional requirement (considering all the functional requirements form every key result and component)

4 User Interface integration

This section describes the strategy followed to provide a single GUI for the URBANITE system as a wrapper of the different components: what is the intended design, which technologies and tools are going to be used, and how it will be implemented. The information reported in this section represents an initial version of the strategy and technological approach for the realisation of the GUI of URBANITE system (URBANITE UI).

4.1 Design System

This section provides an initial version of the guidelines about the design system that the different components of URBANITE system should use. It is essential to underline that even if these guidelines are not mandatory for existing applications, the developers should try to implement them if and when possible.

A design system is a collection of reusable components with a standard style that can be assembled to build several applications. Moreover, a design system provides also best practices to build a high quality, consistent, efficient and scalable UI. Examples of common features that are usually provided within a design system are: a colour scheme called also a “theme”, a set of visual components (e.g. buttons, cards, accordions, dialogs), layout rules defined for instance using a grid system, icons, fonts and form elements.

The colour scheme is a common approach to defining a primary colour that is intended to be used frequently across all the applications; usually, the primary colour reflects the brand colour. An optional secondary colour it is usually defined to accent and distinguish elements. Moreover, it is also a best practice, used by most of the existing design systems, to define semantic colours that will be used to provide useful information to the users. The corresponding CSS (Cascading Style Sheets) classes are self-explanatory, and they are info, success, warning and danger. An example of a success message, that is usually styled with green colour, is provided when the user successfully submits something to a backend API.

The grid system is used to guarantee responsiveness to the application. The main components are containers, rows and columns used to align and display elements. A row is usually divided into 12 columns and the responsiveness is managed through the so-called breakpoints. A breakpoint is the range of predetermined screen sizes (in pixels) that have specific layout requirements. At a given breakpoint range, the layout adjusts to suit the screen size and orientation. Most of the existing design systems defines a set of breakpoints that correspond to specific CSS class and, therefore, to column dimension, and these are usually: small (sm), medium (md), large (lg) and extralarge (xl).

Creating a design system from scratch requires lot of effort, therefore, several open source solutions are available on the market. An example of such open source design systems is Eva Design System[10]. This design system, based on Atomic Design principles[11], provides several components[12] and a set of open source icons[13] by default. Moreover, it is a deeply customizable product since most of the elements are managed thanks to a visual theme definition. In this context, a theme[14] is a set of variables and connection between them that can be customized and used across several applications.

In the context of Urbanite project, taking advantage of the project logo colours, a custom Eva Design theme was built, using Roboto[15] as the suggested primary font. The chosen principal colour is green (hex colour #139973). The Figure 6 depicts the complete palette, including also semantic colours.

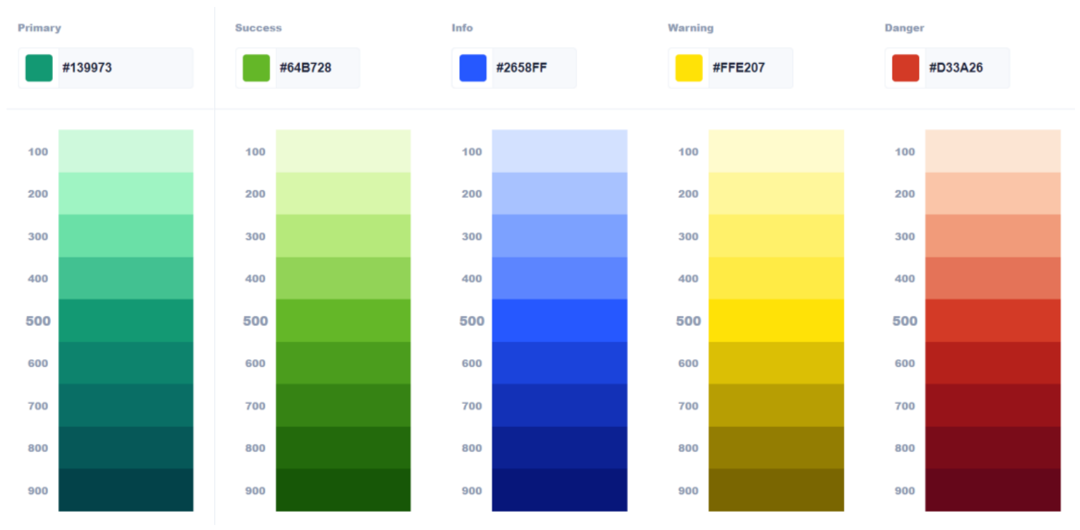


Figure 6: Urbanite Colour Palette

4.2 Suggested technologies

This section provides the list of the main suggested technologies that can be used to build the different applications composing URBANITE UI. Among the several alternatives, the suggested basic JavaScript framework is **Angular**[16]. Angular is written in TypeScript[17] and provides a framework to build efficient single-page applications. The basic core concepts are NgModules, Components and Services. NgModules are the basic brick for every Angular application, and each module provides the compilation context for the Component. A Component defines the view and the logic of a single screen element and may use Services. A Service provides functionalities not strictly related to the view, for instance, a Service might be used to share configuration data among the several Components. One of the most important NgModules is the Router, which provides functionalities to define the navigation path in the single-page application and can also be used in a hierarchical manner.

[10] https://eva.design/
[11] https://atomicdesign.bradfrost.com/
[12] https://eva.design/components
[13] https://akveo.github.io/eva-icons/#/
[14] https://akveo.github.io/nebular/docs/design-system/design-system-theme#eva-design-system-theme
[15] https://fonts.google.com/specimen/Roboto
[16] https://angular.io/
[17] https://www.typescriptlang.org/

An implementation of the above mentioned Eva Design System for Angular framework is **Nebular**[18]. This implementation is an Angular UI library that provides several components, four default visual themes and useful modules such as the Auth and Security modules that, among others, support Oauth2 protocol by default. Being the Angular implementation of the Eva Design System, it is possible to build custom themes[19] for Nebular simply defining a SASS[20] (Syntactically Awesome Style Sheets) map, taking advantage of the several CSS variables provided by the framework4.3.

Finally, **ngx-admin**[21] is an open source dashboard based on Angular, Nebular with Eva Design System. Besides the Nebular capabilities, ngx-admin includes some of the most useful angular libraries by default. For instance:

- Leaflet[22], with its Angular plugin ngx-leaflet[23], that is used to manage and interact with maps.
- Chart.js[24], with its Angular plugin angular2-chartjs[25], used to build interactive charts.

Moreover, it takes advantage of **Bootstrap**[26], one of the most popular front-end open source toolkits and includes, besides default Eva icons, also Fontawesome[27] icon free set and Angular Material[28] capabilities. Section 4.3 describes a customized version of ngx-admin intended to be used as a basic template for Urbanite Integrated UI and, therefore, for the new applications that are going to be built within the project. The following Table 2 summarizes the suggested technologies introduced in this section.

Table 2: Summary of suggested technologies

Name	Type	Rationale
Angular	Front-end JavaScript Framework	This library is one of the most used open source JavaScript Frameworks. It is written in Typescript and is used to build efficient single-page applications.
Nebular	Angular UI Library	This library provides several reusable components and features that ease the development of Angular based applications. It is based on Eva Design System and it is possible to customize the theme of an application defining a SASS map.
ngx-admin	Angular Dashboard	This dashboard is based on Angular and Nebular. It includes some of the

[18] https://akveo.github.io/nebular/
[19] https://akveo.github.io/nebular/docs/design-system/design-system-theme#eva-design-system-theme
[20] https://sass-lang.com/
[21] https://github.com/akveo/ngx-admin
[22] https://leafletjs.com/
[23] https://github.com/Asymmetrik/ngx-leaflet
[24] https://www.chartjs.org/
[25] https://www.npmjs.com/package/angular2-chartjs
[26] https://getbootstrap.com/
[27] https://fontawesome.com/icons?d=gallery
[28] https://material.angular.io/

		most used Angular libraries by default.
Leaflet	JavaScript library	This open source library is used to create and manage interactive maps.
ngx-leaflet	Angular library	This library provides components to integrate Leaflet into Angular projects
Chart.js	JavaScript Library	This open source library provides features for creating and managing charts
angular2-chartjs	Angular library	This library provides components to integrate Chart.js into Angular projects
Bootstrap	Front-end UI toolkit	This toolkit provides several front-end features such as a responsive grid system layout
Fontawesome	Icon set	This is an additional icon set, beside the default one provided by Nebular
Angular Material	Angular UI Library	This library provides Material Design components for Angular applications

4.3 UI Template and Integration strategies

This section provides the description of a UI template that is suggested to be used for the integrated UI and the new applications that will be developed. Moreover, this section also provides some initial guidelines about possible integration strategies.

Taking advantage of the default ngx-admin dashboard, a customized version is provided. The following two JavaScript packages are added to the default list:

- **ngx-translate**[29]: this package is used to manage internationalization inside Angular applications.
- **@ngx-config**[30]: this package is a useful utility since it allows us to load configuration properties (e.g. backend API base URL) at runtime. In the context of the URBANITE project, it helps in the "dockerization" process of Angular application.

The Urbanite logo is added to the navbar, as depicted in Figure 7.

Furthermore, the default material theme is customized to include the colour palette depicted into Figure 6, and the following is a portion of such theme[31] (urbanite-ui-template/src/app/@theme/styles/material/_material-urbanite.scss):

```
$nb-themes: nb-register-theme((
  font-family-primary: Roboto,
  font-family-secondary: Roboto,
```

[29] https://github.com/ngx-translate/core
[30] https://www.npmjs.com/package/@ngx-config/core
[31] https://git.code.tecnalia.com/urbanite/wp5-integration-and-devops/urbanite-ui-template/-/blob/master/src/app/@theme/styles/material/_material-urbanite.scss

```
  color-primary-100: #CEF9DC,
  color-primary-200: #9FF4C3,
  color-primary-300: #6AE0A7,
  color-primary-400: #42C191,
  color-primary-500: #139973,
  color-primary-600: #0D836D,
  color-primary-700: #096E65,
  color-primary-800: #065858,
  color-primary-900: #034249,
 ...
), material-urbanite, default);
```

The template itself provides some application integration strategies examples, indeed Figure 7 shows an example of the template's services menu and each item provides an example of such strategies.

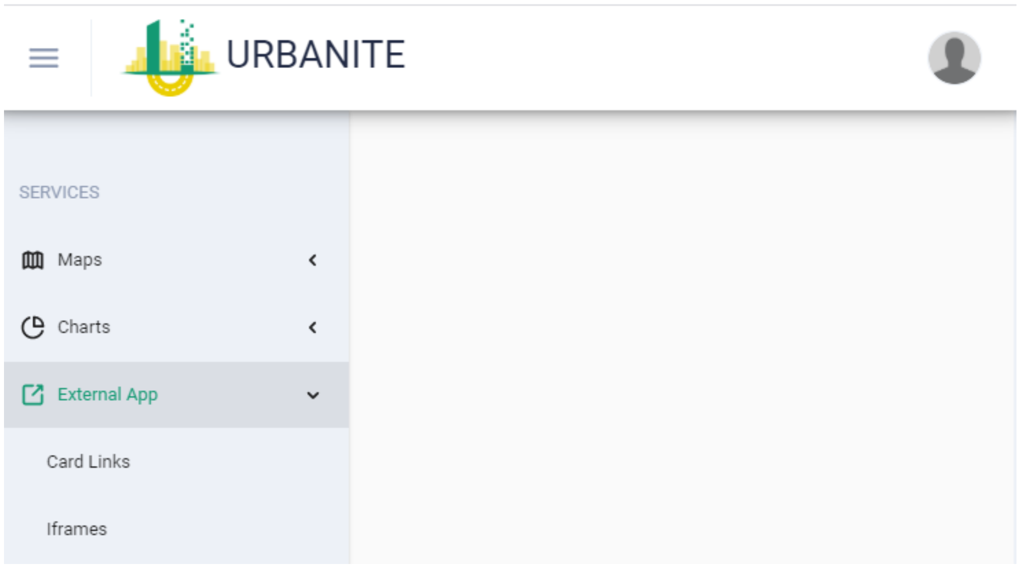
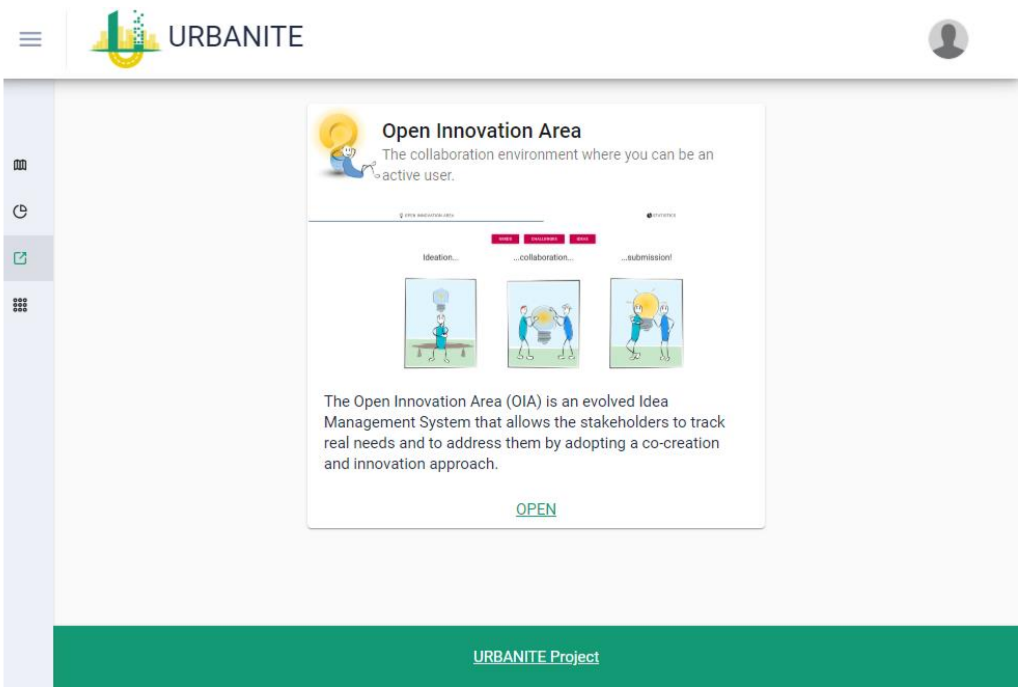


Figure 7: URBANITE-UI Template Menu

In the context of URBANITE UI the two different integration strategies are considered:

- **External component integration**: this type of integration strategy consists in adding the applications through iframes (that is, to embed in an HTML page another HTML document) or links provided using Nebular component, whose example is depicted in Figure 8 showing a Nebular card, that is the basic content container component, describing an external application with its specific link that can be opened through the OPEN button.



The Open Innovation Area (OIA) is an evolved Idea Management System that allows the stakeholders to track real needs and to address them by adopting a co-creation and innovation approach.

Figure 8: URBANITE UI Template, Card Link integration example

- **Template component integration**: this type of integration strategy consists of adding a new application as a new component of the template, taking advantage of the full features provided by the template itself.

To be compliant with the Template component integration strategy, a developer should clone or fork the git repository of the provided basic template, install its dependencies with Npm[32] (Node Package Manager) and work on the new feature following Angular style guide and the suggested best practices[33]. Then, once the newly developed components are ready, these are incorporated in the integrated URBANITE UI. Two examples of such integration strategy are the Maps and Charts services shown in the menu in Figure 7. The Maps service takes advantage of the already provided Leaflet library, and it merely gives an example of how to use such a library, as depicted in Figure 9.

[32] https://www.npmjs.com/
[33] https://angular.io/guide/styleguide

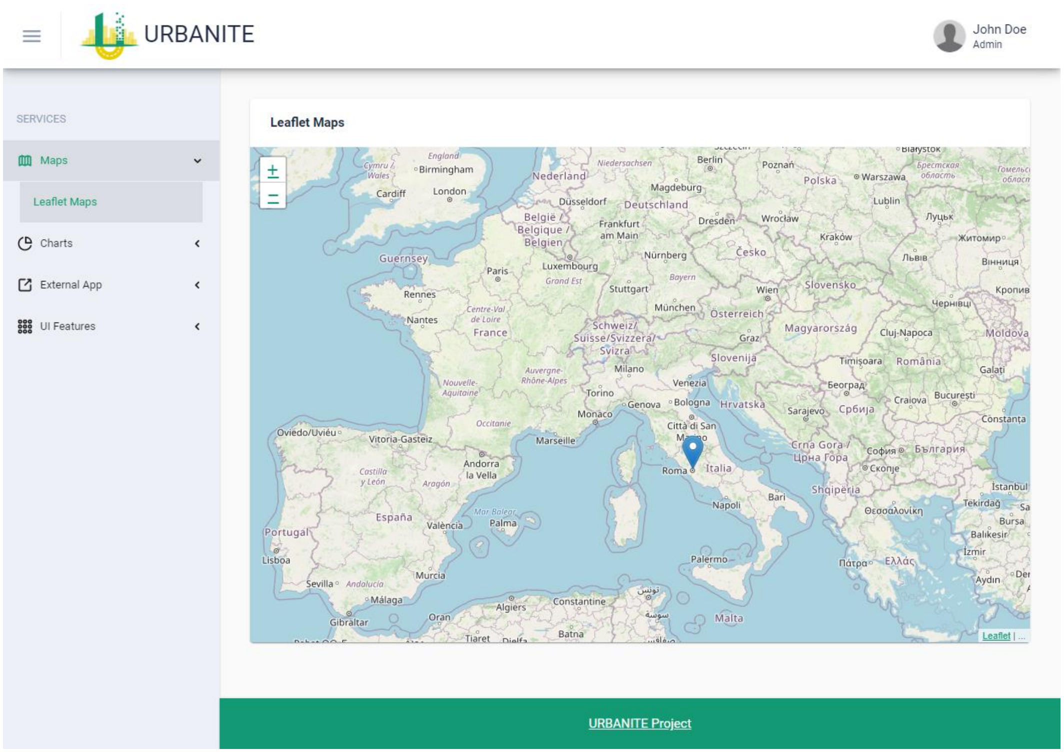


Figure 9: URBANITE-UI Template, Leaflet Map example

Furthermore, Charts service provides examples of how a developer can build one or more charts taking advantage of the template functionalities. The following Figure 10 shows charts built using Chart.js library.

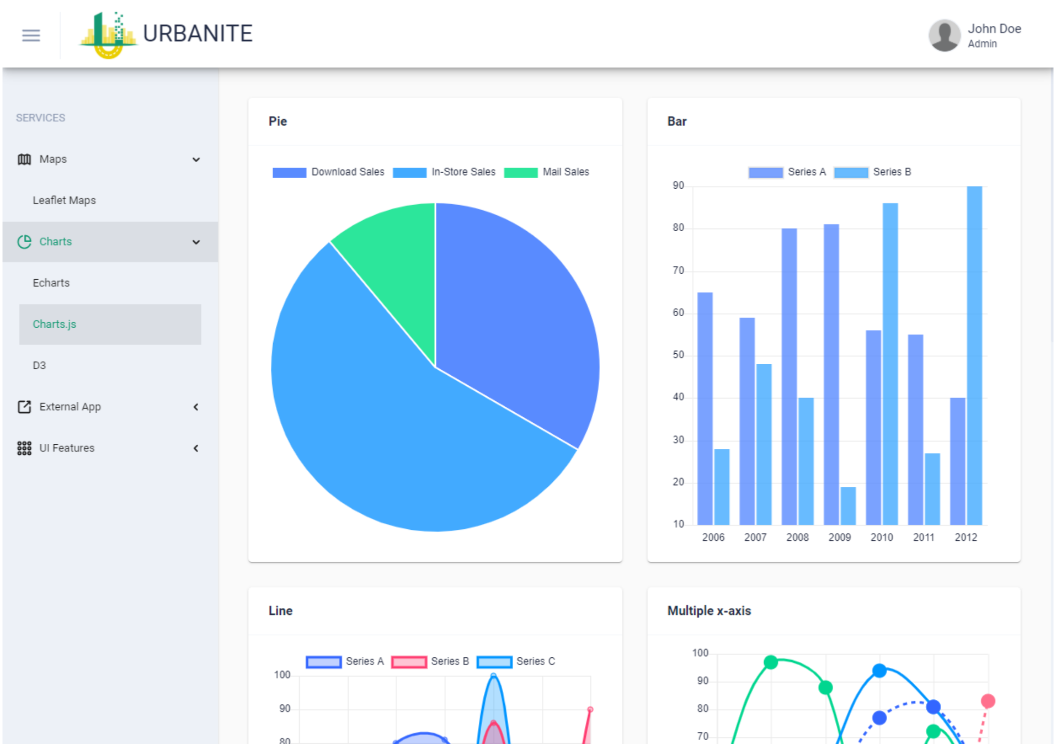


Figure 10: URBANITE-UI Template, Charts.js example

5 Conclusions

This deliverable presents the different tools, environments and strategies envisioned for the management of the development, integration and validation stages of the software components to be implemented during the life cycle of the project.

The DevOps infrastructure has been presented, based in the GitLab version control tool, and with separate environments for development, integration and pilots. Tecnalia is in charge of Task 5.3 - *Continuous Integration and DevOps approach* and will provide the described tools and setup and manage the main software repositories.

Integration and validation strategies have been defined. Jenkins is the tool selected to manage the deployment of the components, virtual machines, and Docker as the container technology are to be used to maintain the hardware independence in deployment.

The provided conventions and best practices will help developers to maintain coherence in naming source files, endpoints, domain names and documenting sources.

The deployments in the several pilots' environments will be used to validate that the URBANITE ecosystem fulfills the requirements elicited in the technical work packages.

The baseline presented here will facilitate partner cooperation in the technical work packages, and will help to deliver the project results -focusing on the software products- meeting the expected schedules.

6 References

[1] New Relic, "Navigating DevOps - What it is and why it matters to you and your business", New Relic, 2014.

[2] URBANITE Consortium, "URBANITE Annex 1 - Description of Action," 2019.

[3] GitLab, «GitLab,» [En línea]. Available: https://about.gitlab.com/. [Último acceso: July 2020].

[4] Git, "Git," [Online]. Available: https://git-scm.com/.

[5] Apache, "Apache Maven," [Online]. Available: maven.apache.org. [Accessed 09 05 2017].

[6] Docker, "Docker," [Online]. Available: www.docker.com. [Accessed 09 05 2017].

[7] xUnit, «xUnit,» [En línea]. Available: https://xunit.net/. [Último acceso: July 2020].

[8] Jenkins, "Jenkins," [Online]. Available: jenkins-ci.org. [Accessed 9 5 2017].

[9] Chef, "Chef," [Online]. Available: www.chef.io. [Accessed 9 5 2017].

[10] Puppet Labs, "Puppet," [Online]. Available: puppetlabs.com/puppet/puppet-open-source. [Accessed 2017 05 09].

[11] Logstash, "Logstash," [Online]. Available: www.logstash.net. [Accessed 09 05 2017].

[12] osTicket, "osTicket," [Online]. Available: osticket.com. [Accessed 09 05 2017].

[13] "Trac," [Online]. Available: trac.edgewall.org. [Accessed 09 05 2017].

[14] URBANITE Consortium, "D7.6 Market Innovation and Applicability Analysis," 2020.

[15] «GitLab issues,» [En línea]. Available: https://docs.gitlab.com/13.0/ee/user/project/issues/. [Último acceso: July 2020].

[16] Wikipedia, "Kaizen-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Kaizen. [Accessed 15 05 2017].

[17] Wikipedia, "Lean-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Lean_software_development. [Accessed 15 05 2017].

[18] Wikipedia, "SixSigma - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Six_Sigma. [Accessed 15 5 2017].

[19] Atlassian, "Bamboo," [Online]. Available: https://www.atlassian.com/software/bamboo. [Accessed 15 05 2017].

[20] Travis CI, "Travis CI," [Online]. Available: https://travis-ci.org/. [Accessed 15 05 2017].

[21] Chili project, "Cruise Control net," [Online]. Available: http://www.cruisecontrolnet.org/. [Accessed 15 5 2017].

[22] GitLab, «GitLab CI/CD,» [En línea]. Available: https://docs.gitlab.com/13.1/ee/ci/README.html.

[23] Jenkins, "Jenkins API," [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Remote+access+API. [Accessed 15 05 2017].

[24] Cloud Foundry, "Warden," [Online]. Available: https://docs.cloudfoundry.org/concepts/architecture/warden.html. [Accessed 15 05 2017].

[25] Github, «Docker, Maven plugin,» [En línea]. Available: https://github.com/fabric8io/docker-maven-plugin. [Último acceso: 15 05 2017].

[26] URBANITE consortium, “D1.1 Project Management and Quality plan,” 2020.

[27] URBANITE Consortium, D5.1 Detailed requirements specification.

[28] URBANITE consortium, “URBANITE use cases requirements and evaluation methodology,” 2017.

[29] Atlassian, “Jira Software,” Atlassian, [Online]. Available: https://www.atlassian.com/software/jira.

[30] Aniket Deshpande, “’DevOps’ an Extension of Agile Methodology – How It will Impact QA?,” Software Testing Help.

[31] GitHub, Inc., “GitHub,” GitHub, Inc., [Online]. Available: https://github.com/.

[32] Wikimedia, “GitHub Wiki,” [Online]. Available: https://en.wikipedia.org/wiki/GitHub.