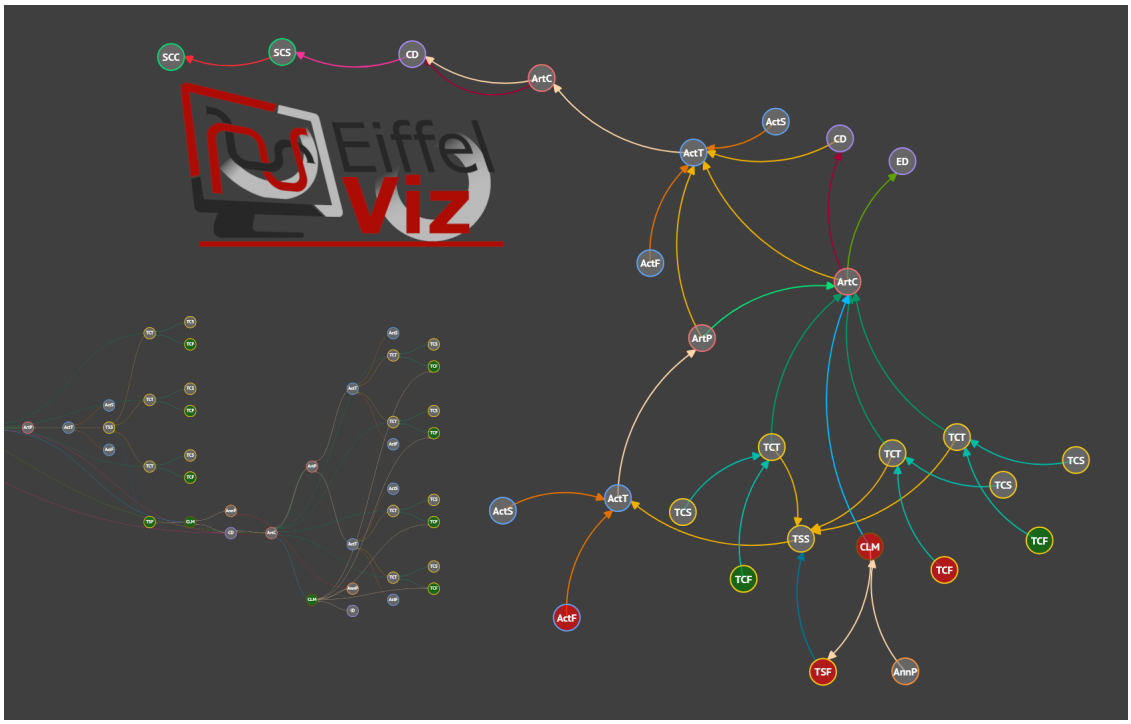
CHALMERS



Visualization of CI/CD Flows Using Eiffel Events

An interactive dashboard for viewing pipelines of Eiffel Events

Degree Project Report in Computer Science and Engineering

Douglas Lindqvist & Philip Osbeck

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Degree project report 2025

Visualization of CI/CD Flows Using Eiffel Events

An interactive dashboard for viewing pipelines of Eiffel Events

Douglas Lindqvist
Philip Osbeck

CHALMERS

Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Visualization of CI/CD Flows Using Eiffel Events
Douglas Lindqvist
Philip Osbeck

Project Supervisor at Nexer: Nikolai Dahlberg
Supervisor: Henrik Jansson Valter, Computer Science and Engineering
Examiner: Muhammad Mustafa Hassan, Computer Science and Engineering

Degree Project Report 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Example picture showing the front-page dashboard for the EiffelViz web application.

Typeset in LaTeX
Gothenburg, Sweden 2025

Visualization of CI/CD Flows Using Eiffel Events
Douglas Lindqvist,
Philip Osbeck
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The Eiffel protocol was developed by Ericsson to enable technology agnostic communication between continuous integration and continuous delivery/deployment (CI/CD) ecosystems. In the Eiffel protocol, every action taken in a CI/CD pipeline is an event. Currently, there is no open-source tool available to get a clear overview of events or view a pipeline flow that is intuitive and easy to understand. The goal was to create a web application that uses the data from Eiffel events and displays the sequence of events and their contents with a dashboard. The application was built using Next.js, Express.js, and MongoDB. It is possible to browse through repositories and branches, and select pipelines to view in their entirety. The visually accessible page showcasing the event pipeline helps the user understand how different events relate to each other and how one sequence led to another. The resulting web application has been validated through positive feedback by stakeholders at Nexer and has helped lay the foundation for how an event-visualizing web application could look and function. An application similar to the one created in this project can in the future help developers and stakeholders follow and analyze Eiffel event pipelines with ease.

Keywords: Eiffel protocol, Eiffel event, web application, Next.js, Express.js, CI/CD, visualization.

Acknowledgements

We want to thank Storm Rothoff for designing the EiffelViz-logo. We also thank our supervisor at Nexer — Nikolai "Tesla" Dahlberg.

Douglas Lindqvist & Philip Osbeck, Gothenburg, May 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
JSX	JavaScript XML
NPM	Node Package Manager
ODM	Object Data Modeling
REST	Representational State Transfer
UI	User Interface
UX	User Experience
VS Code	Visual Studio Code
SQL	Structured Query Language
SVG	Scalable Vector Graphics
XML	Extensible Markup Language

Nomenclature

Below is a list of nomenclature used throughout the report divided into their respective areas.

Git-based systems & CI/CD

Repository A storage of files and directories with metadata to track version history of said files and directories.

Branch A copy of a repository that allow modifications to be made in parallel with other branches of a repository.

Commit A reference to modifications made, each commit represents a different version in the branch and contain metadata about modified files. It is also the act of adding a commit to a branch and thus updating the latest version.

Push The act of publishing local commits or branches to a remote repository.

Rebase Rebasing saves local changes (commits) that are not made on the target branch into a temporary state. The current branch is then reset to the target so it is up to date after which the saved commits are applied one by one. The result is an updated branch with the local changes put after any new changes made in the target.

Merge Merge puts the history of two branches together since their time of divergence into the current branch. The result is a single merge commit that shows all changes.

Pull The act of fetching and merging or rebasing changes from the remote repository into the local repository. It makes the local copy up to date with the remote repository.

Pipeline A series of actions taken manually or automatically when deploying or building new software versions.

Web development

Library A collection of tools leveraged during development to implement additional functionality. Usually in the form of executable code such as functions and classes or objects.

Framework A web framework that eases development by creating a standardized way of building and deploying web applications. Compared to a library it locks the

developer into a workflow with default behaviors and promotes extensibility through structured mechanisms instead of modifying existing code.

Server-side Code that is executed by a server without using resources from the client device. A result is optionally sent to a client.

Client-side Code that is fetched, executed and optionally displayed by the client device.

Full-stack In reference to a framework that can be used to develop both client-side and server-side functionality.

Schema A template of an object-structure with optional additional requirements. For SQL databases this would be the table structure and in NoSQL such as MongoDB it is the document template.

Graph Theory

Acyclic A graph that does not contain any loops. If an edge is followed from a node there will not be a path back to it without crossing already visited edges.

Tree An undirected graph where any two nodes are connected by exactly one path.

Tangled Tree A directed acyclic graph. It is a graph with tree-like structure with multiple inheritance and with one or more nodes identified as root.

Contents

List of Figures

1

Introduction

1.1 Background

As software engineering projects increase in size and complexity, the need for thorough testing and efficiency in the deployment of these projects also grows. In turn, this increases the number of different systems that are involved in testing environments and CI/CD (continuous integration and continuous delivery) flows. A study by Nguyen-Duc [1] highlights that an increase in complexity increases the proneness to failure and decreases maintainability of software. The many different measured metrics of complexity in Nguygen-Duc's work imply that more meticulous and extensive efforts in testing are needed to identify defects so that they can be corrected.

In 2012, Ericsson identified the need for a protocol that allows CI/CD systems to communicate by creating the Eiffel Protocol, which enables technology-agnostic event-based communication among the actors of the ecosystem [2]. In 2016, the protocol had its first contribution to open source, and Ericsson, among other companies, has since then made contributions to the Eiffel Community.

Today, the Eiffel Protocol is used by multiple major companies in their testing environments. The fact that it is open source makes it possible for anyone to integrate it into their system. However, no open-source tool is available to have a clear overview of events or view a pipeline flow that is intuitive and easy to understand. This paper aims to help solve that problem by creating a prototype of a web application with a dashboard that visualizes Eiffel events.

1.2 Purpose

The purpose of this project is to make the visualization of CI/CD flows in systems using the Eiffel protocol more visually accessible. This should improve efficiency and transparency with software developers, middle managers and anyone that development may concern.

1.3 Objective and requirement

Laid out by the supervisor of the project at Nexer, the web application should use the data from Eiffel events, create a website with a dashboard that shows the

sequence of events and what they contain. To achieve the objective and its purpose, we will create a web-based application that can fetch Eiffel events and visualize the data in a variety of different ways relevant to the user. The events presented will be grouped by a selected repository and branch, and the focus of the application will be on Git-based systems. To help achieve this goal, objectives surrounding the stack and design of the web application have been outlined.

1.3.1 Stack

The application will consist of a front-end and a back-end. Data will be stored in a MongoDB database. The front-end and back-end will be developed using JavaScript frameworks, with the front-end using Next.js, and back-end Express.js to get data from the database. Both will run server-side using Node.js. Each component will have tests to verify its correctness. The application will be packaged using Docker to allow for easy deployment on any operating system.

1.3.2 Design

The design will be made in Figma, and then realized in Next.js. There will be a front-page where general information about a selected repository and branch will be available. This homepage aims to achieve the goal of giving an overview of a selected repository. There will be another page listing all available repositories. Clicking on a repository will redirect to a page with a list of commits and artifacts in the selected branch. Furthermore, selecting one of these events will lead to a page where the entire pipeline of that event can be viewed.

1.4 Limitations

Frameworks other than Next.js and Express.js will not be considered or discussed. No comparison is made between other database-management systems and MongoDB since it uses JSON-like documents and since Eiffel events are based on JSON objects, MongoDB is a suitable choice. The objective is to create a functional product, but where room for improvement and further development is an option and encouraged. Any features that are desired but not implemented may be discussed in the report. The application will support one specific version of Eiffel, MongoDB, Express.js, Node.js and Next.js. There are no plans for maintenance of the application to support later or other versions of the technologies previously mentioned. Basic authentication will be implemented to differentiate users, but more advanced checks like checksum verification and validation of Eiffel event data are outside the scope of this project. However, measures like those mentioned are recommended for future development. Furthermore, the creation of Eiffel events will be limited to a number of fixed pipeline flows where, in real use, the composition of these pipelines can vary drastically in structure.

2

Technical Background

This chapter provides technical context to key technologies used in the project for better understanding of the report.

2.1 Web Application

A web application is software that runs in a web browser and provides access to complex functionality without requiring installation. It can be accessed from all modern web browsers, and the development of a web application is fairly simple and cost-effective, since the same version works across all web browsers. A web application typically consists of a client-side and a server-side. The client-side script, meaning a script such as JavaScript, handles functionality like drop-downs and buttons. Interactions like these can trigger communication with the server-side, which handles processing of data and communication with the database [3].

2.2 Figma

Figma, specifically Figma Design, is a web application collaborative design tool where multiple people can create and test design in the browser. Focus is on UI and UX design, and it allows multiple users to work on the same project in real time. Users can design page layout and simulate how buttons and pages interact before any development of the product starts [4].

2.3 JavaScript

JavaScript is a programming language that allows for the implementation of complex functionality on web pages. It is most often used for client-side scripts on web pages, but has been developed to also be used on the server-side. It plays a central role in modern web development, every time something more than static content is displayed on web page JavaScript is likely involved in the process [5].

2.4 Front-end

Front-end refers to the part of software, page or web application that the user directly interacts with. The focus of front-end development is on the user experience. Technologies involved are ones that interact directly with the user, for example, influencing how a component or web page looks or defining the behavior when a button is pressed [6].

2.4.1 HTML & JSX

HTML is the code that structures the content of a web page. It is the most basic building block of the web. It defines what the building block is and how it should behave [7]. How HTML building blocks are styled—as well as their layout is typically defined by CSS (Cascading Style Sheets). JSX is a syntax extension to JavaScript, and allows one to write HTML in React [8].

2.4.2 CSS

CSS is used to style HTML elements. It can control various characteristics of elements from such things as colors of text from the layout and size of all HTML elements on a webpage [9].

2.4.3 React

React is a JavaScript library that enables building user interfaces out of components. These individual pieces of components can be combined together into entire pages. A search bar for instance can be made into a component, and then reused multiple times across the app. React components work like JavaScript functions—and CSS and JSX can be specified inside the component to describe what the UI should look like [10].

2.4.4 Typescript

TypeScript is a programming language that adds additional syntax to JavaScript. It allows developers to add types. When the data has a specific type, it makes it easier to catch errors when types do not match. Type checking is done when compiling the code—meaning before running the code rather than while running the code [11].

2.4.5 Next.js

Next.js is a React framework made for developing fast full-stack applications. This framework handles requirements such as routing, data fetching and caching. Next.js also offers server-side rendering—speeding up the rendering process and increases performance on the client-side. Another feature of Next.js is static generation, which pre-generates HTML pages and stores them as static files and serves them directly on user request instead of fetching the pages when requested [12].

2.4.6 Tailwind CSS

Tailwind is a CSS framework that lets developers use pre-defined utility classes to style HTML elements. The "utility first" concept that Tailwind uses eliminates the need for writing custom CSS—and utility classes are used directly in the HTML code to style the element. An example of a utility class is "text-center"—and is placed in the HTML code. This code replaces the need for custom CSS of the HTML element with the code CSS "text-align: center" which would achieve the same outcome [13].

2.4.7 D3

D3 (or D3.js) is a JavaScript library for visualizing data. It is very flexible and built as a low-level toolbox of modules that work together to let users create almost any type of visualization ranging from graphs, trees, maps, force simulations and more [14].

2.5 Back-end

The back-end of an application is sometimes called the server-side. When a user interacts with the front-end, the back-end handles the request made by the user and responds. It provides all infrastructure to make the application work. It handles things like processing requests, managing the database and its' communication—and other core operations needed for a fully functional application. The back-end is not directly accessible by the user [6].

2.5.1 REST API

A REST API is an API that follows the design principles of the REST architectural style. In essence it is a stateless client-server interface accessed through HTTP meaning a client's state is not saved between GET requests and each request is separate. Data should also be cached to further streamline interactions. [15]

2.5.2 Node.js

Node.js is a runtime environment that can run JavaScript code—outside of a browser. Node can be used to write JavaScript in server-side back-end development. One of the core use cases for Node.js are Web servers and REST APIs. Node runs as a single process, asynchronous non-blocking I/O model which means that the app does not wait for input/output operations to complete. Because it is asynchronous and single-threaded, it can handle many connections at once at minimal overhead—which is what makes it suitable for large scale applications. Among other modules, Node use tools like NPM to handle dependencies and packages [16].

2.5.3 Express.js

Express.js is a Node.js web application framework. It provides features specifically for building web and mobile applications with ease. Express simplifies route handling

and supports building REST APIs as well as provides a lightweight structure for efficiently developing the back-end [17].

2.5.4 MongoDB

MongoDB is a database that stores data in JSON-like documents. Its document oriented approach categorizes it as a NoSQL database, which is another approach to the traditional table-based relational database model [18].

2.5.5 Mongoose

Mongoose is an ODM library to MongoDB. ODM stands for Object Document Model, which is an interface that maps documents from databases like MongoDB to objects in code. It helps developers interact with the database for things like data modeling, schema enforcement and general data manipulation [19].

2.6 Git

Git is an open source distributed version control system. Version control systems like Git let the user control and track changes in code during development. It keeps a complete history of changes, and through multiple functionalities, lets users collaborate on projects safely at the same time [20].

2.7 CI/CD

CI/CD stands for continuous integration and continuous delivery/deployment. The overall goal with CI/CD systems is to automate development workflow and accelerate the life cycle of software development. It works by often and continuously integrating into the main branch of a shared repository, while automatically testing during each commit and merge. Continuous delivery builds upon continuous integration by automating the deployment process, ensuring code can be automatically be deployed into testing or production environments [21].

2.8 The Eiffel Protocol

The Eiffel protocol was created by Ericsson in 2012 for large-scale software integration in the company. It became open source in 2016, and the Eiffel community was formed to support the continued development of Eiffel protocol. The stated mission of the Eiffel community—"The Eiffel community's mission is to provide seamless transparency and interconnectedness throughout polyglot and multi-organizational continuous delivery pipelines" [22].

The protocol provides technology agnostic communication between CI/CD eco systems with otherwise different communication standards. In the Eiffel protocol, every action taken in a CI/CD pipeline is an event. There are many events of different

kinds that represent different actions. Every event references other previous events (if they exist) using links, showing why an event occurred and in what context. These events form an acyclic graph of the entire CI/CD pipeline. Eiffel is the protocol that defines the events and how they reference each other [23].

3

Method

This chapter will aim to describe the process of building the application and its constituent parts.

3.1 Design

The development of the web application started with the creation of the design in Figma Design. Multiple changes were made between the design phase and development phase, which will become evident in the following sections and chapters.

3.1.1 Homepage

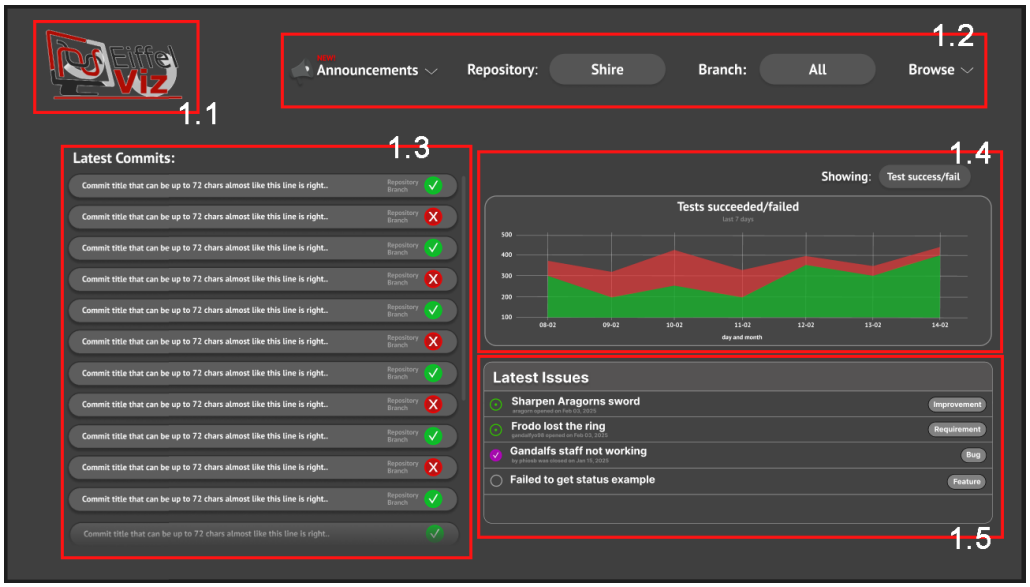


Figure 3.1: Overview of the homepage design in Figma with annotated sections: 1.1 Logo, 1.2 Navigation bar, 1.3 Latest Commits, 1.4 Statistics component, 1.5 Latest Issues.

1.1

The logo had been created as part of a project in another course at Chalmers. This logo was placed at the top left of the page on Figure 3.1 for no other reason than

standard homepage layout. According to a study made by Nielsen Norman Group, users are 89 % more likely to remember logos shown in the top left than the top right corner on web pages [24].

1.2

In the navigation bar, the user can select a repository. In the repository drop down, the user can search for the repository they want. The choice was made later on during front-end development to add a feature to be able to *favorite* a repository. This can be done on the repository page that will be presented in the 3.1.2-section. This was designed because users may not always know the name of the repository they want to view, and displaying a full list of repositories would lead to visual clutter and reduced usability. *Favoring* a repository puts it in a list where it can be accessed from the repository drop-down menu in the homepage. The functionality of branch drop-down is to pick a branch within the selected repository. The rest of the components on the homepage rely on the selected repository and branch to get data and show announcements, commits, stats and issues.

On the navigation bar, there is also a drop-down of the latest announcements in selected repository. An announcement is a type of Eiffel event. This design was aimed to fill out the navigation bar with functionalities relevant to the user.

The Browse drop-down to the right in the navigation bar will navigate to the other pages of the web application. A choice was made to not create the Issues-page and Pipelines-page as they were originally planned. Instead, the pipelines can be viewed from the Repositories page, and the Issues page was sidelined due to its limited relevance to the overall objective. Other tasks were prioritized to complete the web application within the given time frame, such as the page for viewing pipelines.

1.3

A component showing the latest commits of the selected repository and branch can be viewed on the left side of the homepage. This is scrollable and will have a limit of the max amount of commits that show. An icon to the right within a commit-box show whether the tests failed or succeeded for the commit. The commit-box also show the title of the commit with a maximum of 72 characters. Having a list of the latest commits in a repository gives a general overview of a repository's activity when a user is not actively looking for a specific commit on a specific date.

1.4

A component showing a statistic is shown in section 1.4 of Figure 3.1. There is also a drop-down to change what statistic to show. During the design phase, it was not clear what statistic was most relevant to the user. The one shown on the homepage in Figma is an example of a statistic that could be used. Showing statistics of a repository aims to, just like with the previous component, give a general overview of the repository.

1.5

In the bottom right of the homepage there is a Latest-issues component. Issues have different types, which is shown to the right of the issue name. There are three different icons describing whether an issue is open (green), closed (purple) or gray if the information can not be provided for any reason.

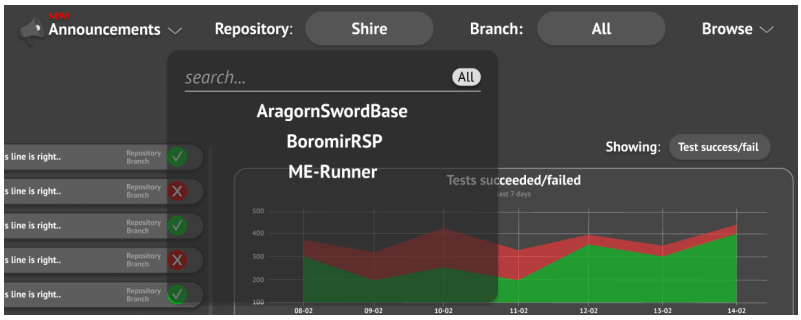


Figure 3.2: The repository selection component in the navigation bar with a drop-down.

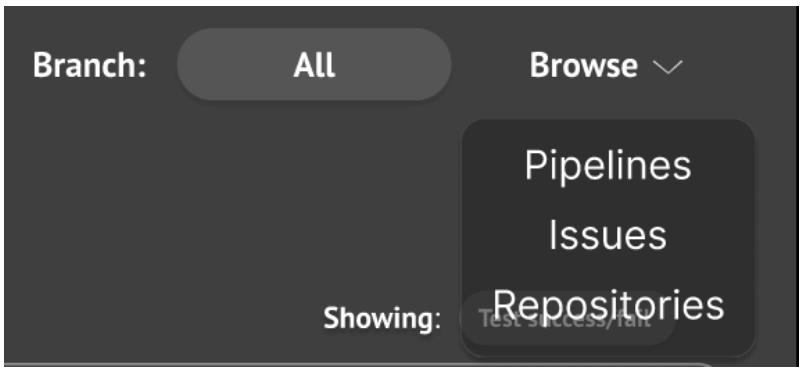


Figure 3.3: The *Browse* drop-down in the navigation bar.

3.1.2 Repositories

The Repositories page has a list of all repositories. The user can search for the repository in the search bar. The name of the repository is shown on the leftmost side of a row. The second column shows the date of the latest submit in the repository, and the user who is responsible for that submit is shown on the third column. The fourth column contains a link to the platform hosting the repository. To the right of the link, the user has the option to *favorite* the repository. Doing this, as previously mentioned will make the repository accessible from the drop-down in the navigation bar. It is possible to sort the repositories alphabetically by name or date of the last submit. The pagination has arrows to click to go to either the next or previous page. Every repository in the list can be clicked to create a drop-down.

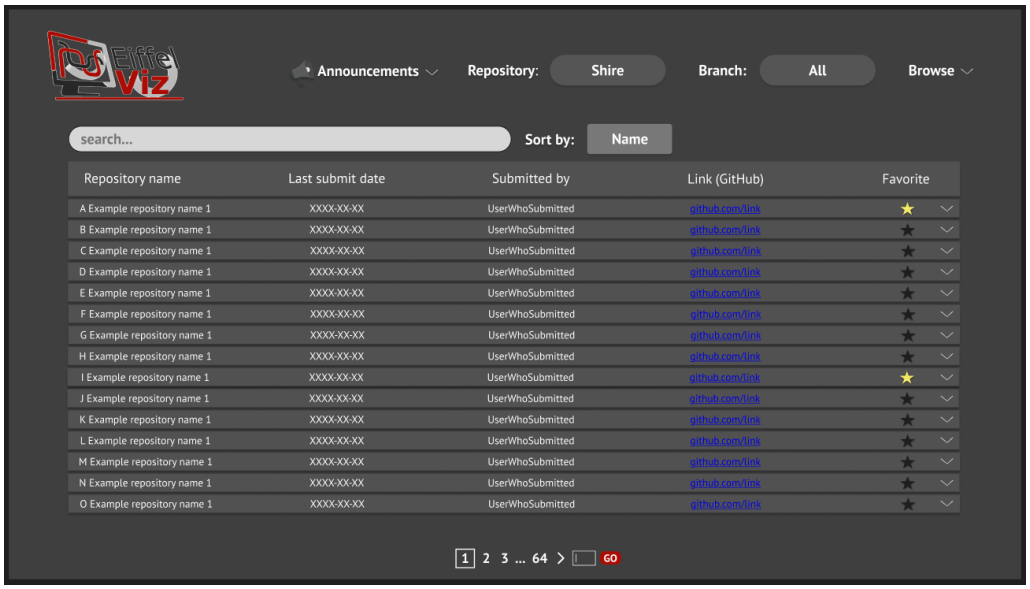


Figure 3.4: The Repositories page shows a search bar and a sort-by drop-down, along with a list of all repositories. At the bottom there is a pagination menu to go through the list of repositories.

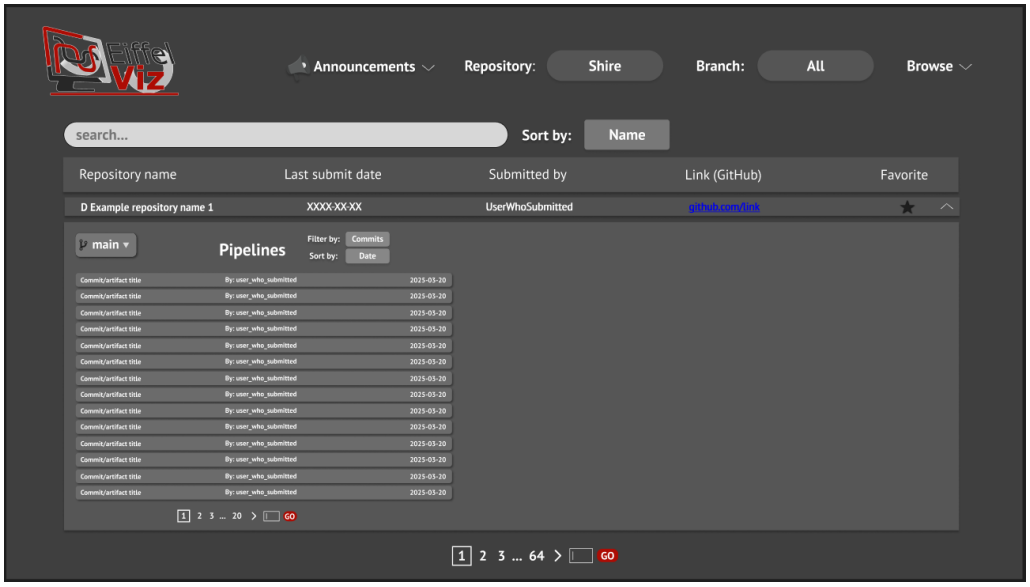


Figure 3.5: The pipelines in a repository and their selected branch.

When a repository is expanded, a box containing the pipelines of the repository show up. The branch can be selected at the top left of the expanded box in Figure 3.5. The list of pipelines can be sorted by date or name, and filtered by commits, artifacts, or both. The expanded pipelines page was never fully finished in Figma as feedback was needed whether to fill the entire repository drop-down with the pipelines or to include the issues in a repository on this expanded box. Clicking a commit or artifact will redirect the user to the page containing pipeline of the commit/artifact.

3.1.3 Pipelines

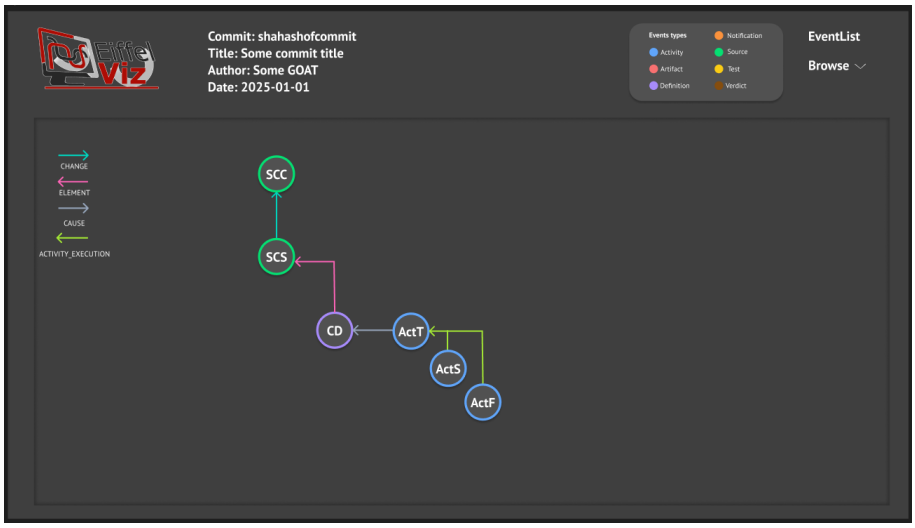


Figure 3.6: The pipeline page of an event showing events of different types and how they relate to one another.

In the designed event graph, there are less events than what is expected to be in a real pipeline. Information about the selected commit or artifact is shown at the top of the page, see Figure 3.6. To clarify which events were triggered and how they are related, a legend is displayed on the left side of the screen explaining the what the colors of the arrows represent. The events also have different types. These are explained in another legend at the top right of Figure 3.6. For further information on what the specific events within the type are, the abbreviations on the event nodes are elaborated on in another legend when pressing *EventList* at the top right of Figure 3.6.

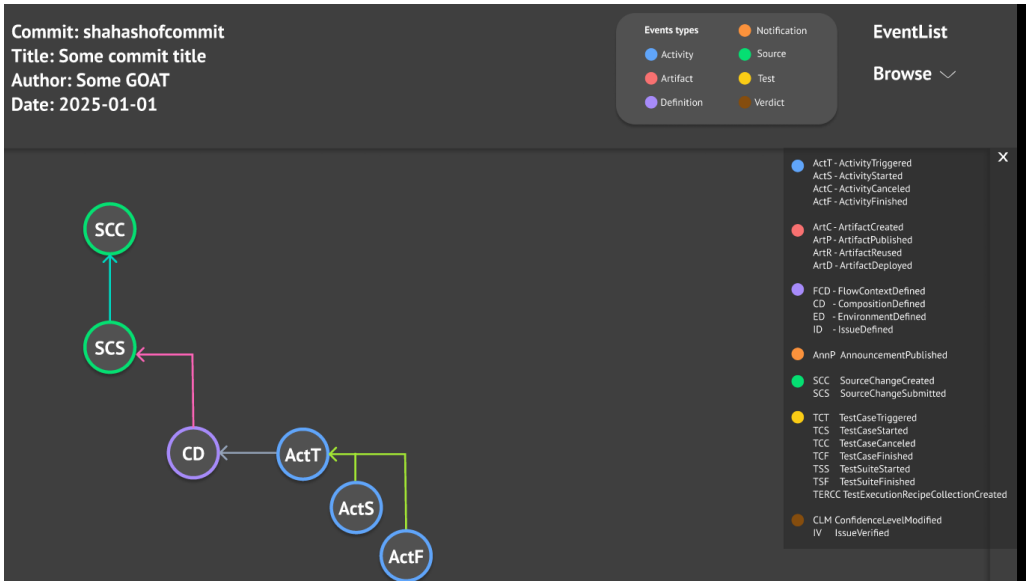


Figure 3.7: Zoomed in image of the pipeline page with the event legend visible on the right.

All of this information was introduced to the user to be able to follow the pipeline of events thoroughly.

3.2 Framework Selection

As outlined in section 1.3 the front-end uses Next.js and the back-end Express.js. It is possible to write both back-end and front-end in Next.js, but the separation is made to allow for deployment on separate servers without the need to download all packages for both. It also allows a separate back-end to be developed and connected with the front-end with minimal change, if any, to the front-end codebase.

Back-end

Express.js was chosen for its small footprint, popularity [25] and because the front-end also use JavaScript and Node.js. To communicate with the database the Mongoose library is used. Mongoose is an ODM that allows the enforcement of specific schemas with ease directly within the back-end application. The alternative is to setup schema validation rules in the database through tools such as MongoDB shell or Compass which would remove the overhead from Mongoose [26]. This would add to development time however and require additional error-handling in the application layer. It was determined that the ease of use outweighed the added overhead, the loss of any flexibility was not a factor as Eiffel schemas enforce a strict schema.

Front-end

The application requires interactivity but it must also be able to fetch large amounts of data without adding overhead for the client. Next.js solves both with server-side components that can fetch and cache data which is then passed on to client-side components that displays it and add interactivity. This component based approach together with the app and pages router [27] promotes a modular design that makes developing multiple different pages simultaneously easier. This is why the decision was made to use it for the front-end, along with its popularity and the performance benefits provided by static generation [28]. Static generation is when HTML pages are pre-generated and stored as static files, as explained in section 2.4.5.

3.3 Back-end

Since the focus of this application is data visualization, the back-end was kept simple to allow the majority of development time to be spent on implementing the front-end. The core functionality is to fetch Eiffel events and format them or aggregate data based on multiple events. It is also able to fetch a series of events that represent a complete pipeline. The back-end is not able to receive new events in its current state, instead a large amount of data was generated automatically to showcase the visualization.

3.3.1 Database Structure

A separate schema for each Eiffel event was constructed with validation implemented in Mongoose based on Eiffel's JSON-schemas [29]. The latest version as of date accessed was used for each event and no guarantee is made of future or backward compatibility. There is also a simple schema for user identification that stores a username, a hashed password and a list of favorite repository names and last selected repository. Basic authentication is made and the identification is used to showcase the functionality of favoring repositories for quick access to their statistics.

3.3.2 Planned routes

A initial plan of routes was made, see list below, and later iterated upon based on what the front-end needed. Some routes were removed completely as they were redundant or irrelevant to the front-end.

/announcements - Announcements for front-page.

/issues - List of issues.

/repositories - List of repositories.

/branch?repo - Branches from repository name.

/pipeline?id - Pipeline flow from ID.

/node?id - Single event by ID.

/tests - List of test cases.

3.3.3 Event Generation

Eiffel events were generated using a modified version of Eiffel's generator.py-script which is a Python-script that generates iterations of example pipelines created by Ericsson. It can be found in the reference data sets of the Eiffel Github repository [29]. It was modified to generate different repository names and branches as well as making sure to only link previous and future versions of pipelines that relate to the same repository.

3.3.4 Pipeline Construction

As outlined in section 2.8 the events in Eiffel are linked by a reference to previous events in the pipeline. Most pipelines, however, start with an event that describe a commit, a merge or artifact build that then leads into tests, new builds, announcements and more which do not have a link except to previous versions. These start points are what is listed on the front-end as they are the most descriptive and representative of an action taken by a developer. To construct a pipeline from these start points a recursive function was made to search for all events that have a link to the start point, iterate over any links and add any that has not been seen and repeating for every new event found. With visual aid of figure 3.8 the function would behave in the following order.

1. Commit.
2. Find Suite Start, ignore link to seen event.
3. Find Test1 Start, Test2 Start and Suite Finish, ignore links to seen event.
4. Find Issue Define link, ignore link.
5. Find Test1 Finish, ignore link.
6. Find Test2 Finish, ignore link.
7. Find Artifact Create, ignore link.
8. Find Artifact Publish, ignore link.

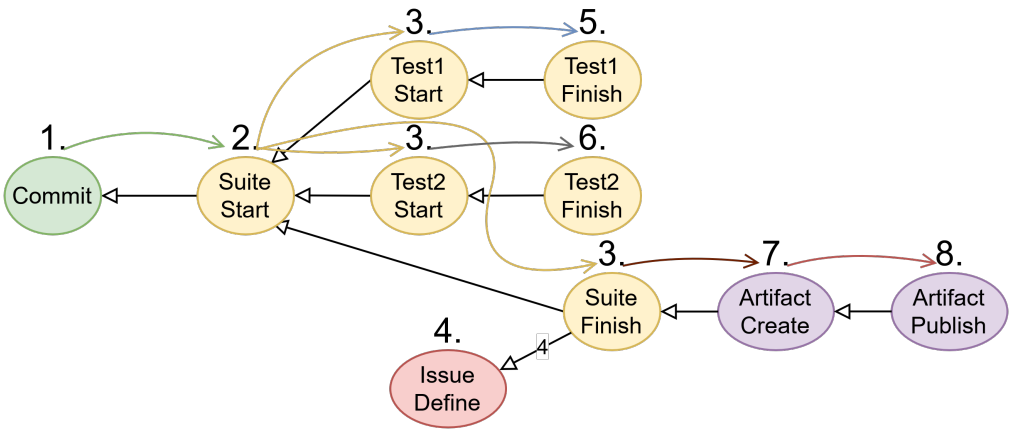


Figure 3.8: Pipeline construction example

A few exceptions were made to this function. If an environment definition event is found it will only be added to seen events but not processed further as multiple different pipelines might rely on the same environment. If an event has a link to a previous version it is instead added to an array of previous or future versions depending on the event was found. This is then used to quickly go forward or backward between pipeline versions in the same repository and branch in the front-end. The performance of the function was lastly analyzed as a load average of a few requests based on the pipeline length and how many events are stored in the database. These graphs are shown in 4.2.2.

3.4 Front-end

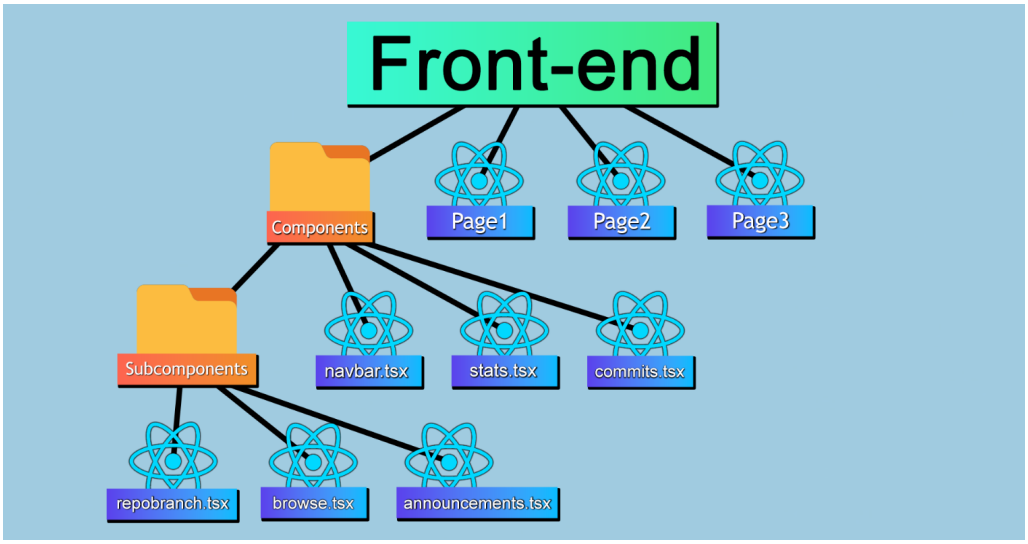


Figure 3.9: Simplified version of the directory layout in the front-end.

The front-end of the entire web application was developed by creating React components for each largely divided section. For instance, the navigation bar consists of three separate components that were created individually, and put in the same *navbar.tsx* file. That file can in turn be reused in as many pages as necessary or as desired.

3.4.1 Homepage

The homepage followed the same direction as it was designed in Figma, with the only change being the implementation of *favoring* a repository to make it visible in the list of repositories in the drop-down of repositories in the navigation bar. The browse-component only links to two pages, the Repositories-page and Homepage. The stat component was created using *Commits Per Day (Last 7 days)* as it was seen more fitting than the example used in the initial design.

The homepage was created using 4 components. A component for the navigation bar, a statistics component, and components for the latest issues and the latest commits.

3.4.2 Repositories

The Repositories-page was divided into multiple components, including the navigation bar. The *Search bar* and *Sort by*-button were made into one sub-component, as well as the expanded drop-drown of the list of pipelines. The list of repositories was made into a single component, and no design changes were made to the page from the state that was created in Figma. Some changes were made to the expanded drop-down however. One of the added changes were icons clearly showing whether

the event is a commit or an artifact. Clickable icons linking to the Git-platform hosting the commit/artifact were also added.

3.4.3 Pipeline Visualization

To build a visual graph of Eiffel events, the D3 library was used. In this graph, events represent nodes, and links are edges between these nodes. The events can be arranged loosely based on time but have no strict hierarchy. For instance, both an activity start and finish event point to the activity trigger event, and events can have multiple parent links.

Initially, a force-directed graph with time scaling was implemented. A legend was created and placed beside the graph with text in the same color as the links to explain what the colors represent. This differed from the initial design, but not significantly. The event list legend was discarded and instead the event type is shown in full when a node is clicked to reveal more information about that event. This information appears on the right or left side of the screen, depending on where the user clicks. Every node has a border that is color-coded to its type, and a legend for this is shown in the top-right corner as in the initial design. If a previous or future version is found, the option is shown to open them directly. For events that have a verdict or outcome such as tests or activities, the node background changes to reflect its status to allow for a fast visual cue of failed events, with green for success, black for inconclusive, and red for any other non-success result.

For pipelines that have a large number of events, the force graph made it difficult to follow the pipeline flow. To alleviate this problem, drag functionality was added to allow the user to move nodes and arrange them at will. This was not sufficient enough to solve the problem, so another visualization mode was implemented based on a tree structure. Since events can have multiple parents, a regular tree is not a viable solution. Instead, a tangled tree visualization was implemented. The tangled tree was made with D3's tree module by loosely arranging events by time as in the force-directed graph. A single link per event was used to force a hierarchy. Then a virtual node was used to connect all potential trees; this node was then simply hidden. The D3-generated links were ignored, and the original links were drawn to create the complete tangled tree. Lastly, an option was added to switch between this tangled tree and the force-directed graph visualization.

4

Results

4.1 Design of the web application

As stated in the Introduction in the section 1.3, goal was to create a web application that can use the data from Eiffel events with a dashboard that shows the sequence of events and what they contain. To achieve this, a design was introduced as an objective in section 1.3.2 that outlined that the application should contain four pages. One of these is the homepage that should give a general overview of a repository. All components in the front-end use test data fetched from the back-end.

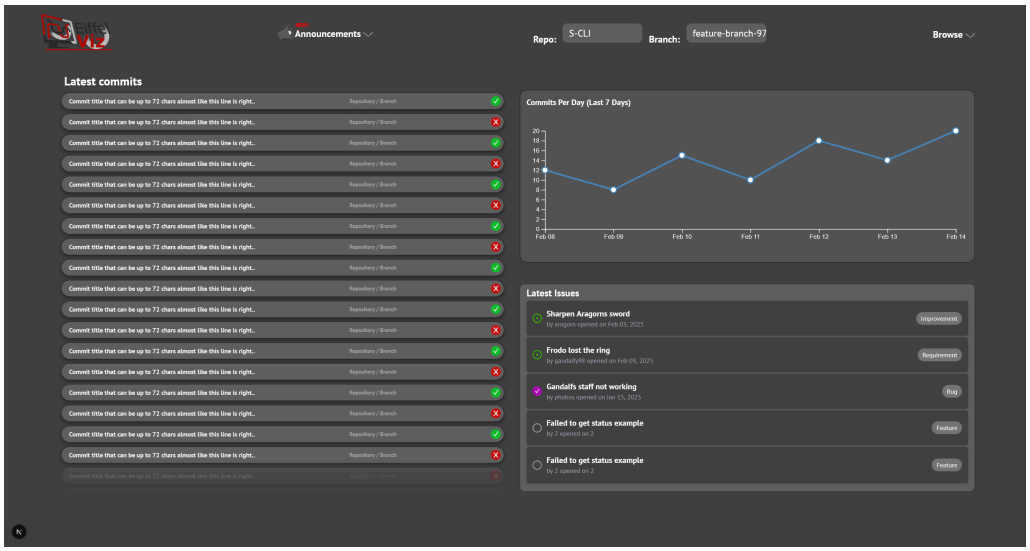


Figure 4.1: The homepage of the web application with mock data and test data.

The homepage of the web application includes the component *navbar.tsx* to select the repository and branch to get the overview of a repository. From this navigation bar it is also possible to redirect to the Repositories-page. The *announcements.tsx* subcomponent of the navigation bar fetches the five latest announcement-events from the repository chosen. There is a component of the latest commits in the repository and branch, a component that show the amount of commits in the repository and branch for the last 7 days, and a component of displaying the five latest issues opened in the repository and branch.

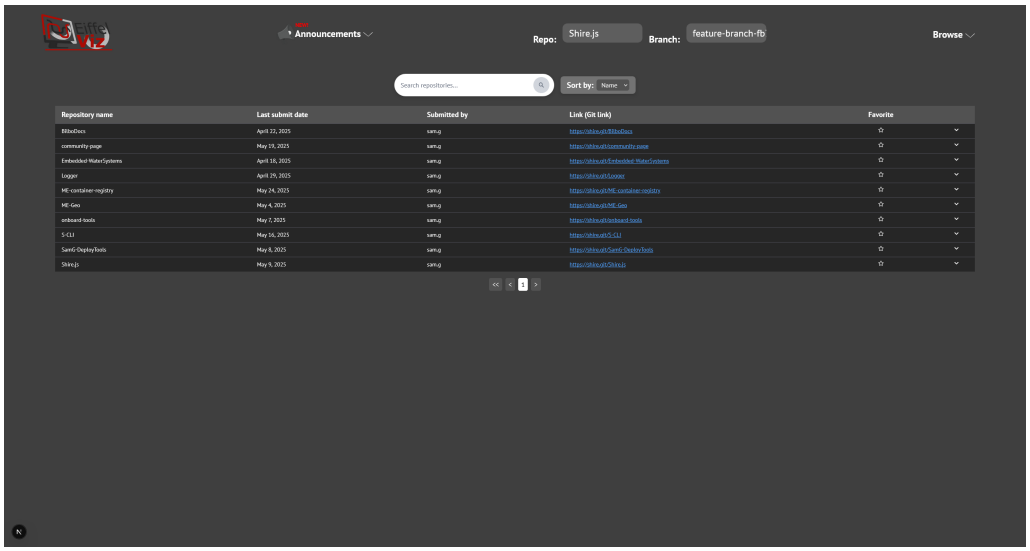


Figure 4.2: The Repositories-page containing a list of all repositories.

The Repositories-page on 4.2 show a list of repositories with the ability to search for repositories and sort either alphabetically by name or by date of last submit. There is also a function to flip through the pages in a pagination menu under the list of repositories. A row of a repository shows some columns of information, has a Git link, and at last the desired ability to *favorite* the repository to add it to a list of repositories that show up in navigation bar. Clicking on a repository does not link to another page as outlined in the desired design in section 1.3.2, but instead expands the repository as a drop-down.

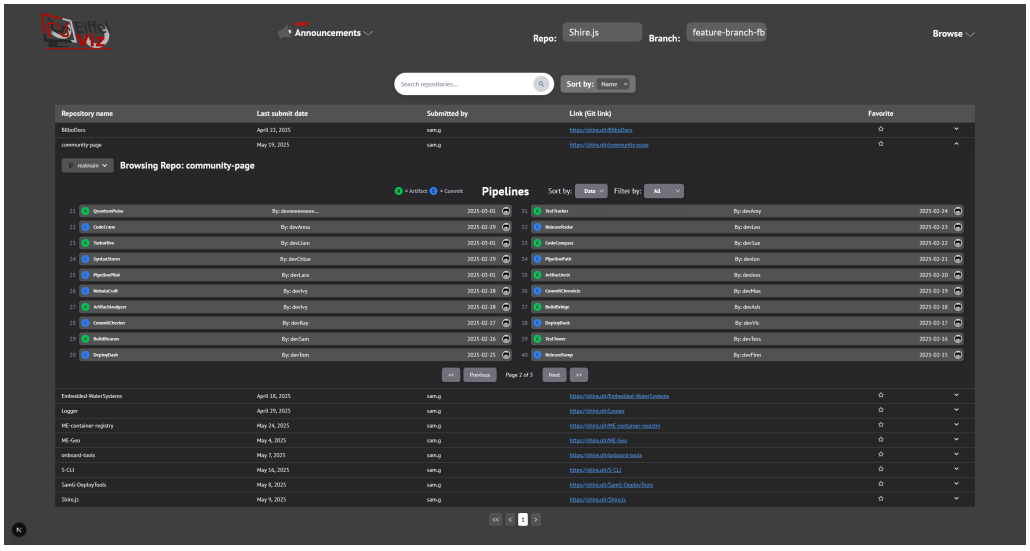


Figure 4.3: An expanded repository from the Repositories-page showing a list of pipelines.

The list of pipelines in Figure 4.3 shows commits and artifacts in the expanded repository. The branch can be selected in the drop down in the upper left hand

of the expanded repository. The pipelines can be sorted by date of last submit or alphabetically by name, with sorting by date being the default. There are icons showing whether each pipeline is an artifact or a commit, with a legend above the pipelines explaining what the icons represent above. The user can use the pagination to go the next page of pipelines in the repository.

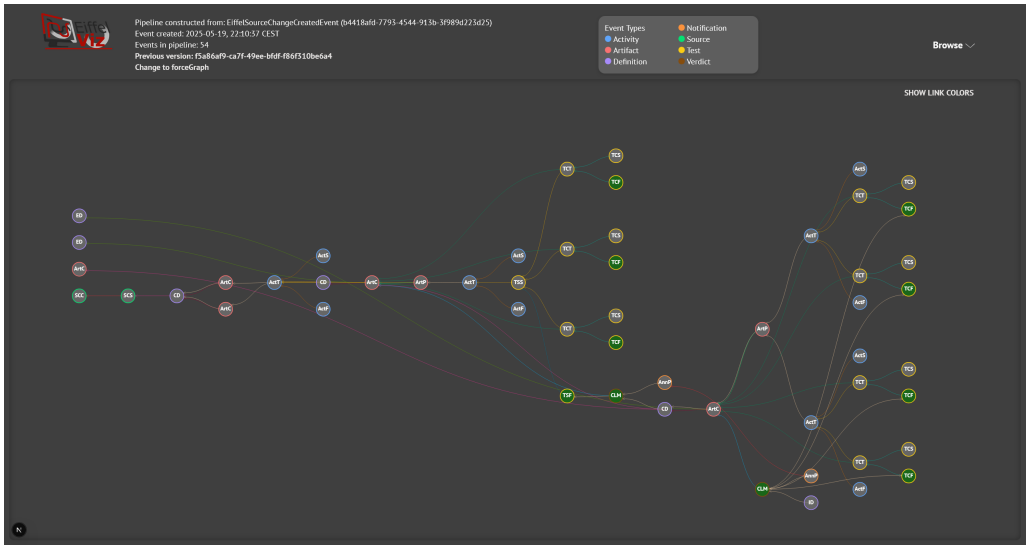


Figure 4.4: Pipeline tree showing multiple events and their connection.

To create a dashboard that shows the sequence of events and what they contain, there is a page capable of visualizing entire pipelines. Information about the pipeline is displayed at the top of the screen in 4.4. This shows which event the pipeline is constructed from, when it was created, how many events the pipeline contain, the previous version of the pipeline, and the ability to change the layout of the tree to better support different sizes of pipelines.

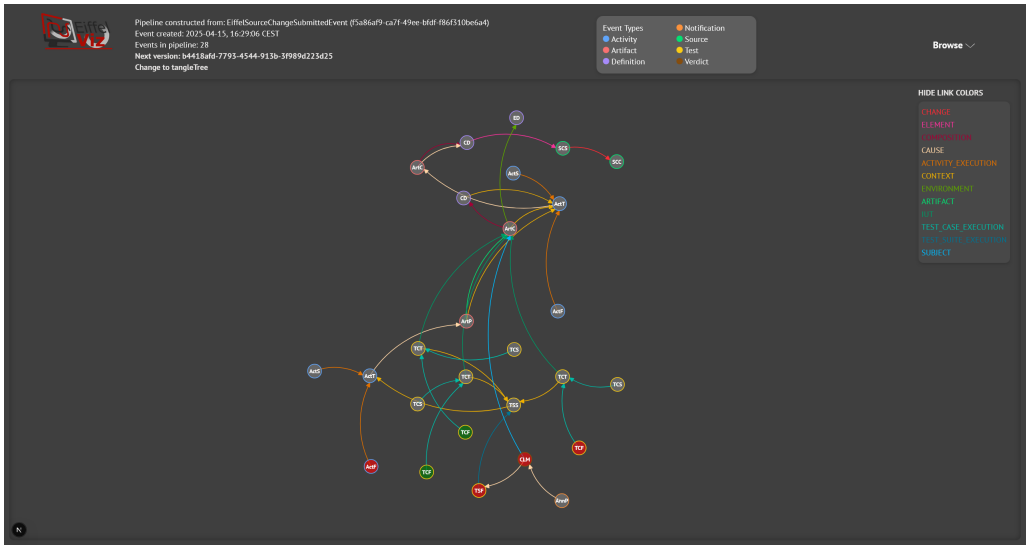


Figure 4.5: A pipeline utilizing an alternative layout of events.

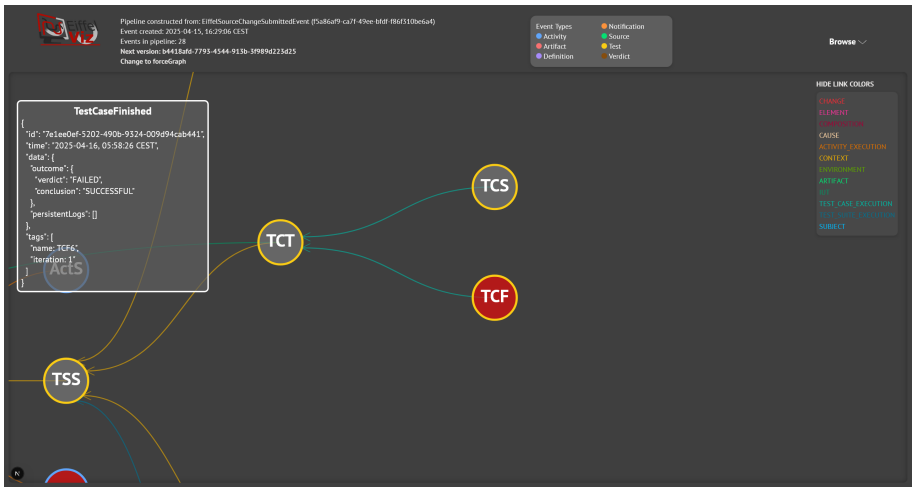


Figure 4.6: A zoomed in image of a pipeline showing the popup when an event is clicked.

The page allows the user to scroll freely and zoom in and out to view the pipeline in the desired way. A legend providing information on event types is shown at the top of the screen. Clicking on *SHOW LINK COLORS* as seen in Figure 4.4 makes the legend on what each link color represent, visible in Figure 4.6. Clicking on an event toggles a popup that show information about the event.

4.2 Back-end

The back-end consists of a few different routes to facilitate the visualization in the front-end. There are a some changes made to the initial plan discussed in 3.3.2. As outlined in the objective 1.3, the focus of this application is on the front-end and thus the back-end was designed to be as simple as possible.

4.2.1 Routes

The finished routes are described below. Except for the initial plan of routes mentioned in 3.3.2 there was no preparation or plan such as class diagrams and the routes were designed ad-hoc as the front-end required it. Any route that retrieves a list of events is paginated, and it is possible to send the query parameters to decide what page and how many to fetch. Query parameters are listed with their default values, if any, in Table 4.1.

Table 4.1: Available API routes and their query parameters or payload

Method / Path	Query / Payload	Description
GET /artifacts	**limit=25, page=1, repo=?, branch=?**	Fetch artifacts based on repository and branch name.
GET /pipeline/{id}	None	Get a complete pipeline-flow based on given {id} of event.
GET /repos	**limit=25, page=1**	Fetch a list of repositories.
GET /branches	**limit=25, page=1, repo=?**	Fetch a list of branches in a given repository.
GET /commits	**limit=25, page=1, repo=?, branch=?**	Fetch a list of commits for a given repository and branch.
GET /issues	**limit=25, page=1, repo=?**	Fetch a list of issues for a given repository.
GET /user/favorites	None	Fetch a list of favorite repositories for current user.
POST /user/favorites	{repo: ?, add: (true|false)}	Add or remove a favorite repository for current user.
POST /user/register	{username: ?, password: ?}	Create a new user.
POST /user/login	{username: ?, password: ?}	Login with username and password, returns a token.

4.2.2 Pipeline discovery

As outlined in section 3.3.4, pipelines are constructed using a recursive function. This function is the core that enables the front-end to visualize a pipeline-flow and is used by the */pipeline/{id}*-route. The function is further detailed in the pseudo-code in Section 4.1. The performance of the pipeline construction was tested in two ways. An average load time of 5 requests was calculated for each data point in both tests. The first shows the load time average as a function of how many events are in the completed pipeline and is shown in the plotted graph in Figure 4.7. The second test shows the average load time of a fixed length pipeline of 50 events based on how many other events are loaded into the database, this is displayed in Figure 4.8. In both tests the first SourceChangeCreated-event was used as a starting point for the construction.

Figure 4.7: Load average based on pipeline length

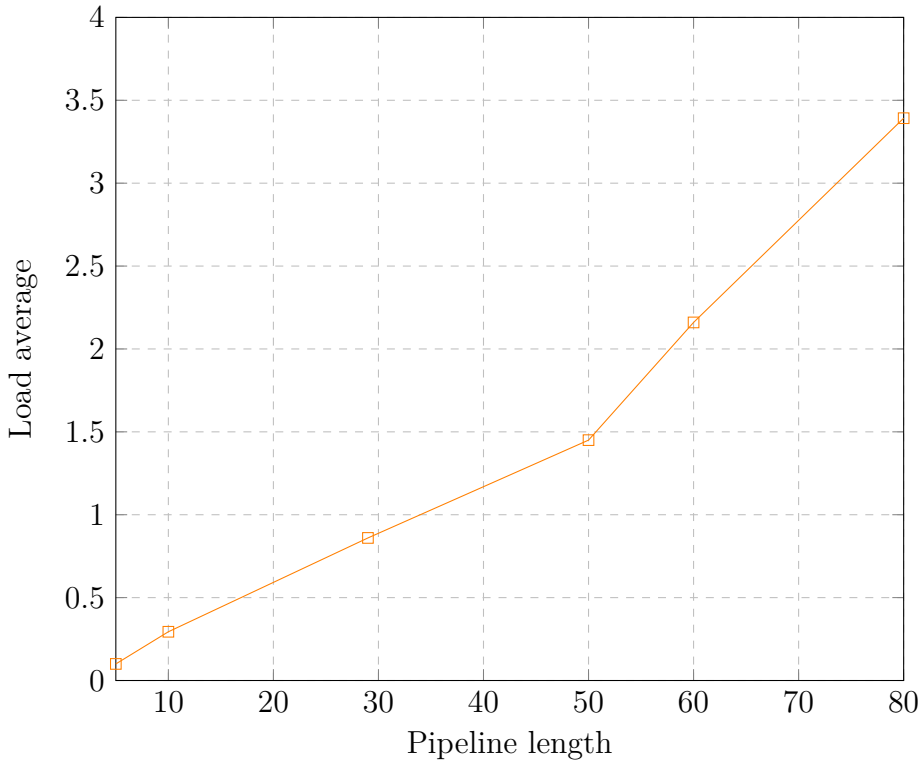
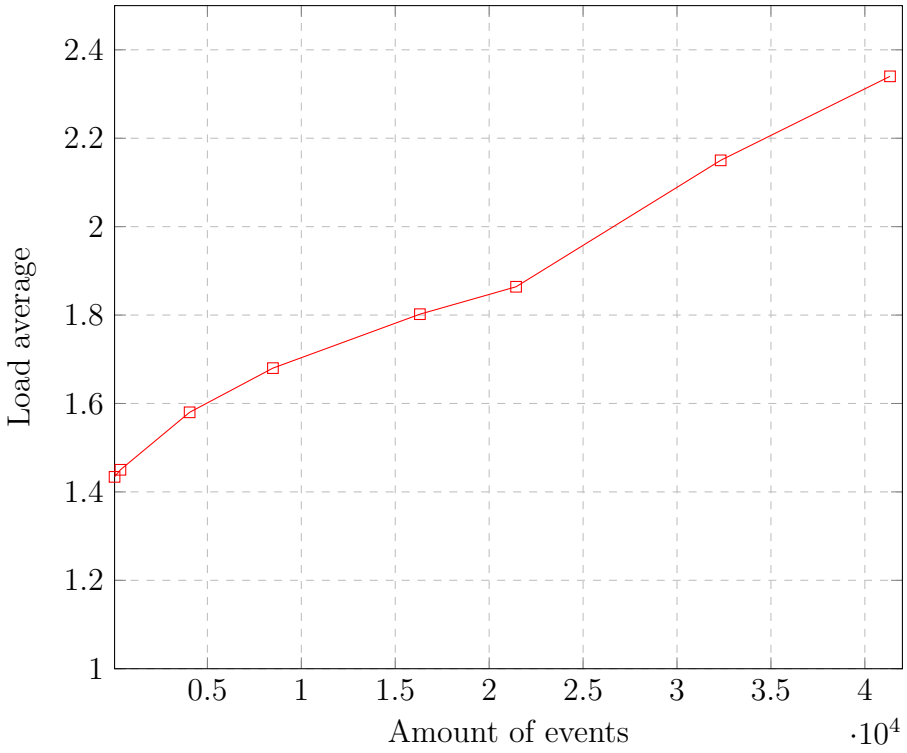


Figure 4.8: Load average based on amount of events stored in database



Listing 4.1: Pipeline recursive function pseudo-code

```
function TraverseEvents(currentEvent) {
  if (currentEvent in visitedEvents); return
  add currentEvent to visitedEvents
  add currentEvent to Pipeline

  // multiple pipelines can depend on the same environment.
  if currentEvent.type is "EiffelEnvironmentDefinedEvent"; return

  // find events in forward links
  foreach (link in currentEvent.links) {
    if (link.type is PREVIOUS_VERSION or BASE) {
        add link to previousVersions
        add link to visitedEvents
        continue
    }
    TraverseEvents(link.target)
  }

  // reverse links
  reverseEvents = newer events with currentEvent in a link.target
  foreach (reverseEvent in query) {
    if (find links in reverseEvent
      where link.type is PREVIOUS_VERSON
      or link.type is BASE
      and target is same as currentEvent) {
        add reverseEvent to futureVersions
        add reverseEvent to visitedEvents
      }
      TraverseEvents(reverseEvent)
  }
}
```

5

Discussion

5.1 Design, Objective & Requirements

A plan was created early to outline the design of the web application. The page visualizing Eiffel events played the most substantial role in accomplishing the primary goal of the project explained in Section 1.3. The rest of the application was mainly designed to increase usability. Objectives were set up in the rest of the design as well to increase usability and create an application that is both transparent and efficient, as per the purpose in section 1.2.

The homepage that was designed in Figma had some added features during the implementation of the design. There are also some flaws, and a few planned functionalities from the design were not implemented. A combination of ambitious design and lack of awareness regarding how much time was left to finish the project could be the reason for this. It is felt that when designing in Figma, it is easy to get carried away with functions, not realizing how much time different things take. A strategy to avoid this and achieve greater time awareness could be to design and develop each component after one another, rather than page by page, which is the way that it was done.

The decision to remove the page of issues planned in Figma was both time-related and for reasons related to whether it helped achieve the objective. A separate page with issues in a repository would not further the greater goal of visualizing sequences of events. An application with more functions that could be of interest to the user could be desirable, but when deciding to remove features because of time constraints, this was one of the first to be picked. The discussed option to have a component of issues in the expanded drop-down of a repository was shut down for the same reason.

5.2 Pipeline construction & visualization

The greatest challenge was constructing the pipeline flow and then visualizing it in an intuitive way. The web application makes it easy to follow a pipeline and quickly see if any tests have failed or what event triggered a build, for instance. There are improvements to be made, especially in how the additional information is displayed when clicking an event node. There was one version during develop-

ment that displayed all information when hovering over a node, ideally a condensed information box would show on hover in addition to the full information on click. Clarity of the information displayed could be improved with further formatting and color-coding. On the other hand, too much formatting could be distracting and lead to obfuscation.

Linking and constructing the pipeline through the recursive function outlined in Section 4.2.2 worked well even with a large amount of data and big pipelines. It is very plausible, however, that a database will continue to grow and accumulate data that is kept for the foreseeable future. In that case, optimizations will have to be made to keep the load times under acceptable values. Optimizations for both cases analyzed could possibly be made through changes to the database structure to allow for easier lookup of linked events instead of recursively searching all events in the database.

As for the objective of visualizing a sequence of Eiffel events and their contents, this is considered successfully achieved. This is supported by feedback from the project supervisor at Nexer, who agrees that the solution meets the requirements of the goal. The ability to navigate through repositories and their pipelines with a range of methods to filter and sort to locate the intended data has also sparked satisfaction with the overall design of the web application for its usability and clarity.

5.3 Further development

Further development should first focus on improving the back-end through database structure optimizations, and added functionality such as new event ingestion with processing and proper handling of user authentication with potential role-based access. If possible, it should also be modified to handle all available Eiffel versions. Ideally some form of object caching such as Redis should be implemented to more efficiently cache responses. Depending on the desired scope, a completely new back-end could be developed and implemented. As the front-end is decoupled from it, minimal changes are required to accept a new back-end.

Though the front-end is a good foundation for an event-visualizing application, there is room for further improvements. One area is visual clarity of the pipeline graphs that can become cluttered and busy in long pipelines with many events. A solution to this could be a dynamic filter that allows the user to filter out specific event types and hide or heavily fade out others. For the front-page there might be different types of statistics that users find relevant. More components that aggregate data in different ways could be implemented, and then the dashboard could be made mutable by the user, allowing them to personalize the data shown.

5.4 Ethics and sustainability

The application needs to take ethical aspects into account regarding security since there is a risk of Eiffel events containing sensitive data. There is basic authentication

implemented, but this is predominantly made to showcase functionality in the front-end, further development is required to allow fine-tuning of access and to potentially hide or remove sensitive data.

The ecological and environmental impact of the application can be viewed as based on two main factors. First is the energy consumption, network usage, and hardware requirements of the data center used when deploying the application [30]. Second is the development time required to make the application in the first place, this requires hardware and electricity for developers.

The choice was made to use popular and efficient frameworks for both the back-end and front-end. The popularity of the framework ensures it will be maintained for a long time and removes the need to rework the application in another framework should the current one be abandoned. The popularity also helps with development time, as it is well documented and thus easy to use.

From a social aspect, the application aims to help both developers familiar with Eiffel and others within an organization. It provides a way to grasp the status of the software being developed and deployed through automated pipelines. These events are difficult to understand or view in the first place for users without direct access and knowledge of the systems that generate the events. In that way the application helps to bridge the gap in understanding by allowing more to view, understand, and participate in the discussion of the CI/CD flows they use.

6

Conclusion

The project was introduced with one clear requirement from Nexer and the supervisor to create a web application with a dashboard that shows the sequence of events that use data from Eiffel events. Supporting objectives were also created to complement the application. These objectives were outlined as a design and stack, with the purpose of making the website user friendly and efficient to use.

As planned, a website with a homepage giving a general overview of a selected repository and branch. The user can redirect from the homepage to a page listing all repositories. Instead of a separate page showing all pipelines in repository as originally planned, they are available in a drop-down menu when clicking a repository. Finally, when clicking on a pipeline, the user gets redirected to a third page that visualizes the entire pipeline tree of Eiffel events along with information about the events.

By building the front-end with Next.js and back-end with Express.js both parts use similar technologies for a streamlined development process. The separation allows for deployment on separate servers as well as making future development easier. The resulting application is also efficient by leveraging technologies specifically designed for the web.

Multiple functionalities were not implemented due to time constraints, like the option to choose between numerous statistics, for instance. This could have been avoided by developing and designing component by component instead of page by page, which would have given better insight in how long different things took and how much time that was left before the project needed to be finished. Further development on the project on the project could improve on the back-end, including database structure optimizations, new event ingestion with processing, and proper handling of user authentication with potential role-based access. A solution to the cluttering that could occur and be an issue with particularly long pipelines could be a dynamic filter that allows the user to filter out specific event types.

The current state of the web application, its complete design, together with the pipeline tree visualization has been validated through positive feedback from the Nexer and the supervisor. The web application has established a foundation for the future development of an Eiffel event-visualization tool.

References

[1] A. Nguyen-Duc, "The impact of software complexity on cost and quality," *International Journal of Software Engineering & Applications*, vol. 8, no. 2, 17-31, March 2017. doi: https://doi.org/10.48550/arXiv.1712.00675

[2] Eiffel Community, "Eiffel Community Website," [Online]. Available: https://eiffel-community.github.io/ Accessed on: 2025-05-12

[3] Amazon Web Services, "What is a Web Application?" [Online]. Available: https://aws.amazon.com/what-is/web-application/ Accessed on: 2025-05-12

[4] Figma, "What is Figma?" [Online]. Available: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma Accessed on: 2025-05-12

[5] Mozilla, "What is JavaScript?" [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/What_is_JavaScript Accessed on: 2025-05-12

[6] Amazon Web Services, "The difference between frontend and backend," [Online]. Available: https://aws.amazon.com/compare/the-difference-between-frontend-and-backend/ Accessed on: 2025-05-12

[7] Mozilla, "HTML: HyperText Markup Language," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML Accessed on: 2025-05-12

[8] React, "Introducing JSX," React Documentation (Legacy). [Online]. Available: https://legacy.reactjs.org/docs/introducing-jsx.html Accessed on: 2025-05-12

[9] Mozilla, "What is CSS?" [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/What_is_CSS Accessed on: 2025-05-12

[10] React, "React – Official Documentation," [Online]. Available: https://react.dev/ Accessed on: 2025-05-12

[11] Refsnes Data, "TypeScript Introduction," [Online]. Available: https://www.w3schools.com/typescript/typescript_intro.php Accessed on: 2025-05-12

[12] Sanity, "Next.js – Glossary," Aug. 2024. [Online]. Available: https://www.sanity.io/glossary/next-js Accessed on: 2025-05-12

[13] Sanchhaya Education, "Introduction to Tailwind CSS," Oct. 2024. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-tailwind-css/ Accessed on: 2025-05-12

[14] Mike Bostock and Observable, "What is D3?" [Online]. Available: https://d3js.org/what-is-d3 Accessed on: 2025-05-12

[15] Red Hat, "What is a REST API?" May 2020. [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api Accessed on: 2025-05-13

[16] Miquido, "Why use Node.js?" Mar. 2023. [Online]. Available: https://www.miquido.com/blog/why-use-node-js/ Accessed on: 2025-05-12

[17] OpenJS Foundation, "Express - Node.js web application framework," [Online]. Available: https://expressjs.com/ Accessed on: 2025-05-12

[18] Oracle, "MongoDB Overview," Okt. 2024. [Online]. Available: https://www.oracle.com/se/database/mongodb/ Accessed on: 2025-05-12

[19] J. Hall, "Getting Started With MongoDB & Mongoose," Aug. 2024. [Online] Available: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/ Accessed on: 2025-05-12

[20] GitLab, "What is Git version control?" [Online]. Available: https://about.gitlab.com/topics/version-control/what-is-git-version-control/ Accessed on: 2025-05-12

[21] Red Hat, "What is CI/CD?" Dec. 2023. [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-ci-cd Accessed on: 2025-05-12

[22] Eiffel Community, "Mission and Vision." [Online]. Available: https://eiffel-community.github.io/mission-and-vision.html Accessed on: 2025-05-12

[23] Eiffel Community, "What is Eiffel and why should I care?", YouTube, Okt. 2016. [Online]. Available: https://www.youtube.com/watch?v=FNF6b4Yv7oQ Accessed on: 2025-05-12

[24] Nielsen Norman Group, "Logo Placement and Brand Recall," [Online]. Available: https://www.nngroup.com/articles/logo-placement-brand-recall/

[25] I. Buljic, E. Kadusic, T. Cvijanovic, N. Hadzajlic and N. Zivic, "Comparative Performance Analysis of Leading Backend Frameworks for Developers," *2025 24th International Symposium INFOTEH-JAHORINA (INFOTEH)*, East Sarajevo, Bosnia and Herzegovina, Mar. 2025, pp. 1-5, doi: 10.1109/INFOTEH64129.2025.10959250.

[26] A. Kukic, S Vlaeva, "MongoDB & Mongoose: Compatibility and Comparison," *MongoDB Developer*, Nov. 2021. [Online]. Avail-

able: https://www.mongodb.com/developer/languages/javascript/mongoose-versus-nodejs-driver/ Accessed on: 2025-05-12

[27] Vercel, "What is Next.js?", [Online]. Available: https://nextjs.org/docs Accessed on: 2025-05-12

[28] S. Pati and Y. Zaki, "Evaluating the Efficacy of Next.js: A Comparative Analysis with React.js on Performance, SEO, and Global Network Equity", *New York University Abu Dhabi*, United Arab Emirates, Jan. 2025. doi: https://doi.org/10.48550/arXiv.2502.15707

[29] Eiffel Community, Eiffel, Github. Available: https://github.com/eiffel-community/eiffel Accessed on: 2025-05-12

[30] E. Fatima and S. Ehsan, "Data Centers Sustainability: Approaches to Green Data Centers," *2023 International Conference on Communication Technologies (ComTech)*, Rawalpindi, Pakistan, 2023, pp. 105-110, doi: 10.1109/ComTech57708.2023.10165494.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS