
**Bachelor Thesis**

Submitted on

# Silhouette

Developer Avatar Visualization from Git Contribution Data

Mehmet Fatih Tekin

*Abstract*

Understanding and communicating software development activity remains a challenge in collaborative environments. In the context of software engineering, developers generate large volumes of contribution data ,but traditional dashboards offer limited personalization, engagement, and insights, making it difficult for developers to interpret even their own contributions beyond raw metrics. In this thesis, we present an avatar-based visualization system that maps Git based contribution data, such as commits, added lines, the number of programming languages used, and the number of repositories the developer contributed to, into RPG style avatars. Each visual trait of the avatar corresponds to a specific contribution metric, offering an intuitive and engaging representation of individual developer profiles. This approach enables more interpretable and personalized insight into developer activity.
**Keywords**: Git Activity Analysis; Gamification; Developer Profiling; Web Visualization

**Advisor:**
Michele Lanza
**Co-advisor:**
Stefano Campanella, Marco Raglianti

Approved by the advisor on Date:

# Contents

# 1 Introduction

## 1.1 Motivation

Version control systems, particularly Git, have long been integral to software development practices. They facilitate collaborative workflows, ensure traceability, and generate a continuous stream of structured metadata about how developers contribute to software projects. Platforms such as GitHub extend this functionality by providing social and collaborative features, enabling large-scale analyses of developer activity across projects [15, 18].

This metadata, ranging from commit frequency, the number of programming languages used, and the range of repositories a developer contributes to, can offer insights into individual contribution patterns, technical breadth, and participation levels [17]. However, such behavioral signals are typically visualized through basic heatmaps or static dashboards. While useful for quantitative analysis, these representations often lack personalization or intuitive meaning, limiting their utility for analyzing individual developers or identifying them.

This gap presents an opportunity to explore more engaging and interpretable visualization methods. In particular, personalized visual summaries of developer contributions can foster a stronger connection between developers and their own activity data. In this project, we investigate the use of RPG-style avatars to visually encode Git-based contribution metrics, enabling developers to explore their profiles in a more expressive and accessible way.

## 1.2 Problem Statement

Although platforms like GitHub provide extensive developer activity data, the way this information is typically presented often fails to support personalization or reflect individual identity [15].

This limitation can hinder self-reflection and reduce the motivational potential of development data. Developers may find it difficult to recognize their progress, understand their behavioral patterns, or communicate their contributions within a team. Particularly in collaborative settings, the absence of personalized, interpretable visual representations can obscure the individual and collective value of software contributions.

Recent research in gamification and software visualization highlights the importance of using intuitive, engaging metaphors to support behavioral awareness and intrinsic motivation [12, 17]. However, such mechanisms must be carefully designed; poorly implemented gamification can encourage superficial engagement, where developers focus on earning points rather than meaningful contributions [17]. In light of this, there is a need for visualization approaches that translate raw developer metrics into more expressive, human-centered representations.

This project addresses this gap by introducing a system that maps Git-based contribution data to RPG-style avatars[1] . The goal is to enable personalized and interpretable representations of developer activity, promoting self-awareness and engagement without relying on competitive scoring or extrinsic incentives.

## 1.3 Objectives and Contributions

The objective of this project is to develop a personalized and engaging way to represent developer activity by leveraging data available in Git repositories. Rather than relying on conventional charts or numeric summaries, this work proposes a visual approach that uses RPG-style avatars to encode developer contribution metrics. The goal is to support self-reflection, improve the interpretability of contribution data, and provide a novel visual representation of developer profiles.

To achieve this, we implemented a web-based system that extracts Git activity using PyDriller and computes key metrics, the total number of commits, lines of code added and deleted, language diversity, and repository involvement. These metrics are then mapped to specific avatar components using color-coded levels such as bronze, silver, gold, and so on, to visually convey the developer's activity. The detailed mapping logic, including the role of thresholds and the assignment of visual traits, is presented in Section 4.4.

The structure of the avatar remains fixed, ensuring visual consistency and comparability across developers. The frontend, built with Vue.js and PixiJS, renders these avatars and includes tooltips that explain each visual trait. The prototype provides interpretable visualization that balances clarity with personalization.

The key contributions of this project are as follows:

- A modular backend pipeline for extracting and processing contribution data from Git repositories.
- A frontend visualization interface with interactive explanations for each avatar component.
- A metric-to-trait mapping model for representing Git-based developer contribution metrics through static avatars.

[1] https://en.wikipedia.org/wiki/Role-playing_game

- A working prototype that demonstrates the feasibility and interpretability of avatar-based visualizations in the context of developer contributions on GitHub.

## 1.4 Report Structure

The remainder of this report is organized as follows:

- **Section 2** background on gamification in developer tools, the use of avatars as visual metaphors, and the rationale behind key design choices.
- **Section 3** defines the functional and non-functional requirements of the system.
- **Section 4** describes the system architecture and explains how Git-based developer metrics are extracted and mapped to visual avatar traits.
- **Section 5** presents concrete usage scenarios that illustrate how the system can be applied in practical contexts.
- **Section 6** evaluates the system in terms of performance, scalability, and interpretability of its visualizations.
- **Section 7** concludes the report and discusses directions for future work.

# 2 Background

## 2.1 Gamification in Developer Tools

Gamification refers to the use of game design elements—such as points, badges, leaderboards, or levels—in non-game contexts to improve user engagement and motivation [14]. It has gained traction across many domains, including education, fitness, and productivity applications. In educational settings, gamified systems often reward students with stars, badges, or progress indicators, which have been shown to increase motivation, engagement, and academic performance [13].

Fitness applications often include competitive features, allowing users to compare progress with friends and earn virtual medals. Language learning platforms like Duolingo employ similar strategies, using streaks, experience points (XP), and progress milestones to maintain engagement.[2] These techniques provide continuous feedback and a sense of progression that can help sustain long-term user participation.

Developer-focused tools have also begun integrating gamification elements. Platforms such as Stack Overflow award users with badges and reputation points based on the quality and frequency of their contributions.[3] Learning environments like Codecademy (Figure 1) and freeCodeCamp turn coding into a structured game, providing challenges, milestones, and visual rewards to encourage progression and persistence.

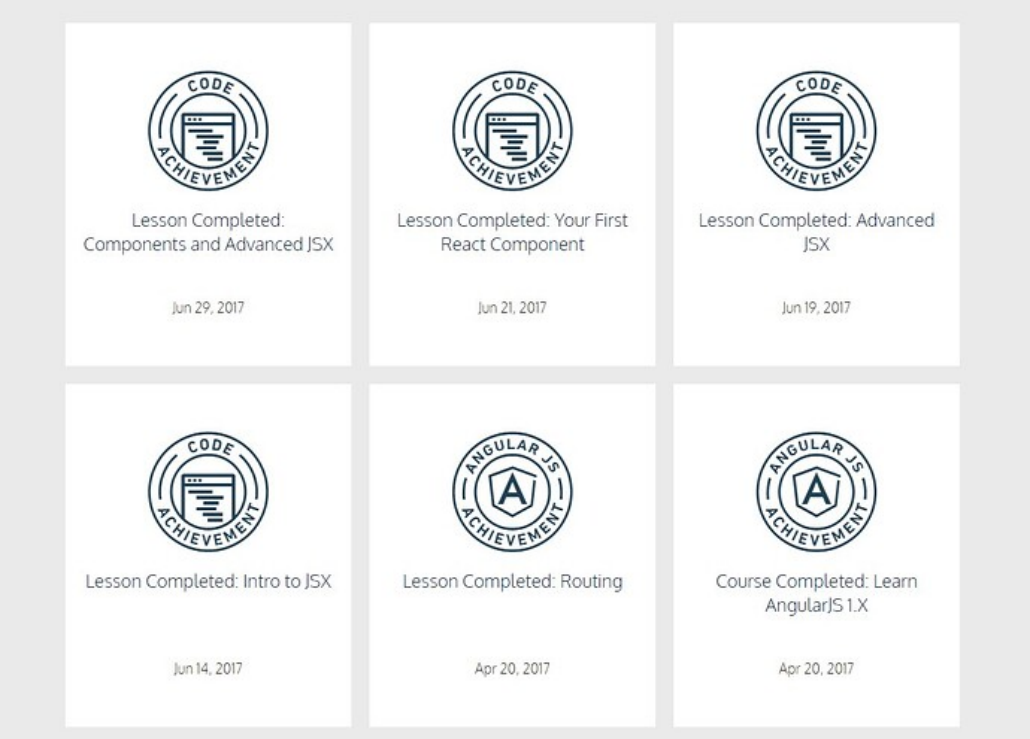


**Figure 1.** Gamification elements in Codecademy: badges, checkpoints, and progress tracking.

While not all visual elements constitute gamification—such as GitHub's contribution graph, which is primarily a form of data visualization—many tools adopt game-inspired mechanisms to influence developer behavior. These integrations reflect a broader trend toward making technical environments more engaging, reflective, and more personalized.

## 2.2 Avatars as Visual Metaphors

Avatars are a central element in gamified systems and user interfaces, providing users with a visual and often personalized representation of themselves within the system [11]. In this project, we use RPG-style avatars to visually encode software developer behavior using real data extracted from Git repositories. Rather than being procedurally generated, these avatars are composed of static graphical components whose color varies according to specific developer metrics.

The specific structure and visual mapping of avatar components in this project is described in Section 4.4.

[2] `https://en.wikipedia.org/wiki/Duolingo`
[3] `https://en.wikipedia.org/wiki/Stack_Overflow`

## 2.3 Developer Activity Visualization

Developer activity can be analyzed using various metrics, including the number of commits, programming languages used, lines of code added or deleted, and the number of repositories contributed to. These data points, available through version control systems such as Git, offer valuable insights into how developers engage with projects. However, interpreting this information can be challenging—particularly when presented in raw or overly technical formats that lack context or personalization.

Most modern developer platforms present activity through standard visual summaries. GitHub, for instance, uses a contribution heatmap—a grid of shaded squares indicating daily commit activity [15]. Other tools, such as GitStats and GitHub Insights, employ bar charts, timelines, and pie charts to provide high-level overviews of contributions over time [18, 16]. While effective for identifying overall activity levels, these approaches primarily emphasize quantity and frequency. They often fail to capture nuances related to a developer's habits, technical breadth, or coding style.

This lack of personalization can make it difficult for developers to recognize meaningful patterns in their own work. Standard dashboards do not convey how developers behave or how their work evolves across repositories and technologies. As a result, the visualizations may feel abstract or disconnected from the developers themselves.

Our project adopts a different perspective by using avatars as personalized visual representations of developer behavior. Rather than displaying contributions as numbers or charts, the system maps Git-derived metrics—such as language diversity, code churn, and repository involvement—to specific avatar components. For example, the number of added lines might influence the color of the pants, while language diversity affects the helmet.

# 3 Requirements

This section outlines the functional goals of the system. It describes the core operations the platform must support, followed by non-functional requirements related to performance, usability, and scalability.

## 3.1 Functional Requirements

The system must support several core functionalities to enable the collection, processing, and visualization of developer contribution data. First, it should be capable of automatically mining activity data from Git repositories, including commit history, lines of code added or deleted, file types, and metadata such as author names and emails. To ensure accurate attribution, the system must normalize and merge developer identities across repositories by comparing email addresses and name variants.

Once the data is collected, the system must compute key contribution metrics, including the total number of commits, code churn (lines added and deleted), language diversity based on file types, and the number of repositories a developer has contributed to. These metrics must then be mapped to specific avatar traits using a level-based visual encoding scheme that reflects a developer's activity level.

Based on the computed levels, the system must generate a static avatar for each developer. This avatar consists of fixed parts—helmet, armor, pants, boots, and weapon—each styled according to one of the computed metrics. These avatars must be rendered through a web-based interface that supports interaction, including sorting, filtering, and exploring individual developer profiles. Additionally, users should be able to customize the visualization by adjusting the thresholds that determine avatar styling, allowing for a more interpretable and personalized experience.

## 3.2 Non-Functional Requirements

The system must satisfy several non-functional requirements to ensure a reliable and user-friendly experience. In terms of performance, avatar rendering on the frontend must remain responsive under typical usage conditions. Since all developer metrics are preprocessed on the backend, the interface should load quickly without requiring intensive client-side computation. Backend operations such as filtering, sorting, and pagination must be efficiently handled to maintain lightweight and fast API responses.

Scalability is essential to accommodate a growing number of developers and repositories. The backend should handle increasing volumes of commit data by aggregating results efficiently and using pagination where necessary. Preprocessing data before frontend rendering helps maintain consistent performance even as data volume grows.

Responsiveness is another priority. The user interface is primarily optimized for desktop environments, where users can fully interact with avatars, tooltips, and modals in a rich layout. While basic layout adjustments exist for smaller screens, ensuring a reliable mobile experience is considered secondary and may require further refinement in future versions.

Usability is achieved by providing an intuitive interface that allows users to explore developer avatars, filter by language or repository, and adjust visualization thresholds without needing technical expertise. Interactive tooltips and modals offer real-time feedback and additional context, making the system accessible to a broad audience.

Maintainability is supported through a modular architecture. The backend is organized into services, controllers, and data transfer objects (DTOs), making it easy to extend or modify. The frontend uses reusable Vue.js components, ensuring a consistent and manageable structure for future development.

Finally, robustness and error handling are essential to maintain system stability. The platform should handle incomplete or inconsistent data gracefully. For example, if a developer has no commits or language data, the system must still render a default avatar without breaking functionality. Fallback mechanisms and error catching are included to ensure that the system performs reliably with real-world Git data.

# 4 System Design

## 4.1 Architecture Overview

The system follows a web-based client-server architecture composed of a backend service, a frontend interface, and a document-oriented database. All components run locally during development and are designed to work together as a self-contained application. While initially intended for small-scale deployment, the architecture allows for future public release or scaling if needed.

The data collection phase is handled by a standalone Python script using PyDriller. This miner is executed once and extracts structured data from local Git repositories, including commit metadata, developer information, and inferred language usage. The results are stored in MongoDB, which serves as the central data store.

The backend is implemented using Spring Boot. It exposes Representational State Transfer (REST) Application Programming Interfaces (APIs), that allow the frontend to request developer data, filtered and aggregated as needed. The backend handles sorting, searching, pagination, and metric summarization. Only the backend communicates with the database — the frontend does not query MongoDB directly.

The frontend is developed using Vue 3 and Vite. Vue Router is used to support multiple views such as individual developer pages, the full developer gallery, and per-project timelines (see Section 4.3 *Frontend: Visualization and Avatar Assembly* for details). All visualizations are rendered with PixiJS, and the interface styling combines Element Plus components with custom layouts and effects built using plain CSS. Avatars are constructed dynamically based on developer metrics returned from the backend, and the interface includes features such as tooltips and modals for interaction.

This architecture (See 2 supports a clean separation of concerns: The miner handles data ingestion, the backend serves and processes the data, and the frontend is responsible for rendering and user interaction.

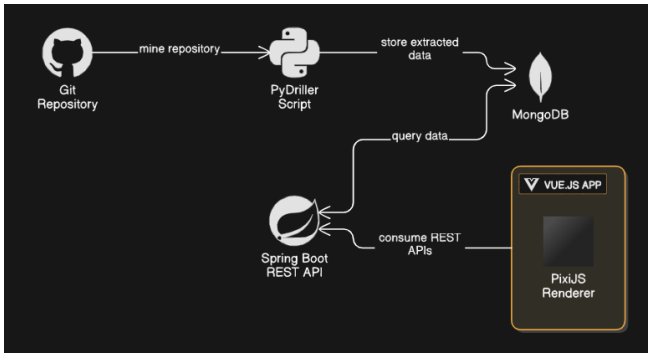


**Figure 2.** General architecture of the system.

## 4.2 Backend: Data Extraction and Processing

The backend of the system is responsible for retrieving, organizing, and serving developer data from a pre-populated MongoDB database. It plays a central role in preparing the information needed for visualizing avatars and powering the frontend's filters, metrics, and tooltips.

The data collection process is handled externally by a Python script using PyDriller. This miner is executed once before running the backend and reads a list of Git repository URLs from a CSV file. It traverses each repository's commit history and extracts detailed metadata including commit hashes, messages, timestamps, and line changes. Developers are uniquely identified using a combination of name and email, and duplication is avoided through internal checks. Each commit is also linked to a developer and a repository via identifiers. In addition, the miner infers the programming languages used by examining file extensions. All data is saved into three MongoDB collections: `developers`, `commits`, and `repositories`.

The backend is implemented using Spring Boot. It exposes RESTful APIs which allow the frontend to request both individual developer data and paginated lists of developers. The main API endpoint for retrieving developer overviews is `/api/developers/contributions`, which supports pagination, searching, filtering, and sorting. This endpoint powers the avatar gallery page and uses MongoDB's aggregation pipeline to efficiently summarize activity for each developer.

This aggregation process involves filtering developers based on user input (e.g., project, language, or name), grouping their commits by repository, and calculating total commit counts and lines added or deleted. The results are assembled into `DeveloperContributionsDTO` objects and returned with paging. This use of aggregation helps keep the system scalable and performant, even when many developers are present.

Other endpoints enrich the system's flexibility and interactivity. The endpoint `/api/developers/{id}` returns detailed information about a specific developer, including their name, email, username, total lines of code added and deleted, list of programming languages, and a mapping of their commit contributions per repository. This data powers the detailed developer view in the frontend, where avatar components and tooltips are customized based on the developer's individual metrics.

In addition to summary statistics, the backend also provides detailed commit-level data via the endpoint `/api/developers/{id}/commits`. This endpoint returns a list of individual commits made by the developer, each including metadata such as the commit hash, message, date, number of lines added and deleted, and the name and URL of the associated repository. Each commit object also includes a dynamically generated URL linking to the original commit location (e.g., on GitHub), enabling users to directly inspect specific contributions.

To support this functionality efficiently, the backend retrieves all commits associated with the given developer and performs a batched lookup to resolve related repository details. The results are mapped to `DeveloperCommitDetails-DTO` objects and sorted in reverse chronological order before being returned to the client.

This commit-level API allows the frontend to render a sortable and filterable list of commits, enabling users to explore contributions based on date or impact. It enhances traceability and supports a more detailed understanding of developer behavior, complementing the avatar-based visualization.

The `/api/projects` endpoint provides the list of all known repositories in the system. It is used to populate project filter options in the avatar gallery view and to list available projects in the project overview page. The `/api/projects/{id}/developers` endpoint returns the list of developers who contributed to a specific repository, along with a timeline of their commit activity. This enables the frontend to display how a developer's avatar evolves over time within a specific project, supporting time-based evaluation and comparative insights. The system also includes endpoints for retrieving all known programming languages, which helps filter developers based on language diversity.

To map developer metrics to avatar levels, a two-stage data-driven approach is used. For continuous metrics such as commit count, deleted lines or added lines, log-normalization followed by quantile-based binning is applied using Pandas' `qcut` function. For categorical metrics like the number of repositories or programming languages, percentile-based ranks are used. These computed levels determine the visual attributes of avatar parts, such as armor and helmet colors.

In the avatar gallery view, users can adjust these thresholds manually through the interface. The system uses default thresholds calculated during data mining, but allows for overriding them in the frontend to experiment with visual interpretations of the same data.

Sorting, searching and filtering behavior varies depending on the view. In the avatar gallery, all sorting (by name, commits, or repositories) and filtering (by project, language, or search keyword) is handled by the backend. In contrast, the project list view applies these operations locally in the frontend using Vue.js, as the dataset is relatively small and stable.

Overall, the backend serves as the central bridge between mined Git data and the interactive visual representation seen in the frontend. By preprocessing metrics, exposing flexible APIs, and offloading computation from the client, it ensures the system remains efficient, extensible, and ready for future scaling.

## 4.3 Frontend: Visualization and Avatar Assembly

The frontend of the system is implemented using Vue 3 and Vite, providing a responsive and interactive user interface for visualizing developer activity. Vue Router is used to manage navigation across multiple views, including the avatar gallery, developer profile pages, project timelines, and more. The application is modular in design, with reusable components such as the header and project-specific avatar canvas, although most rendering and logic is handled directly within view components.

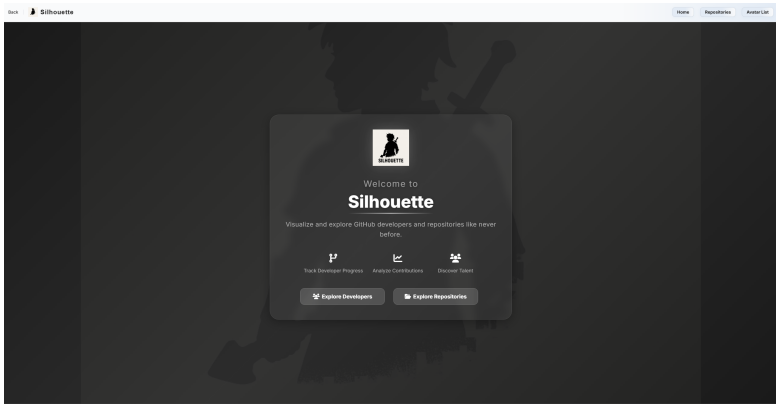


**Figure 3.** Welcome screen of the system.

Avatars are rendered using PixiJS, a high-performance 2D rendering engine based on WebGL. Initially, multiple strategies were explored for avatar generation, including the use of open-source assets and AI-generated pixel avatars using ComfyUI. However, inconsistencies across generated parts led to the decision to create a unified avatar by generating a full-body pixel art figure using an AI model and then manually segmenting it into individual parts: helmet, armor, pants, boots, weapon, and face. These parts are stored as static images and layered using hard-coded positioning to create the final avatar visual. The system also supports rendering multiple avatars simultaneously.

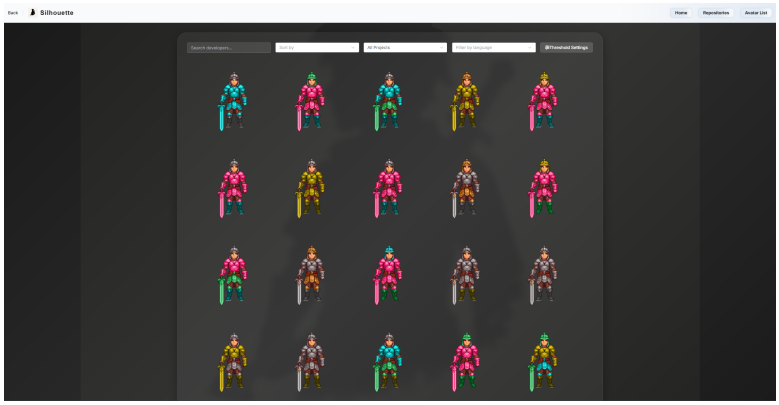


**Figure 4.** Avatar gallery view with searching, sorting, filtering, and adjustable thresholds.

The mapping from developer metrics to avatar visuals is handled on the client side. The system assigns each metric (e.g., number of commits, lines added, number of languages) to a specific avatar part. A utility function computes a visual level (e.g., bronze, silver, gold) for each metric based on pre-defined or user-defined thresholds. These thresholds are stored in the browser's local storage and can be customized via the user interface, allowing for flexible and interactive visual interpretation of developer traits.

Users can interact with the system through several mechanisms. Hovering over each part of an avatar reveals a tooltip that explains what metric it represents, what level it maps to, and a short narrative interpretation (e.g., "Smart mind for learning 5 languages"). In the avatar gallery, users can sort developers by name, number of commits, or number of repositories, and filter them by project or programming language. Thresholds that define metric levels can also be manually adjusted through the settings panel. In the developer detail view, users can also click on the displayed numbers for total commits, repositories, or programming languages to open dedicated modal views. The commits modal presents a detailed list of all commits made by the selected developer, including the commit message, date, repository name, and a breakdown of lines added and deleted. Users can sort this list either by date or by number of lines added. Additionally, each commit entry includes a clickable link that opens the commit on the original Git platform (e.g., GitHub), providing transparency and traceability down to the level of individual contributions.

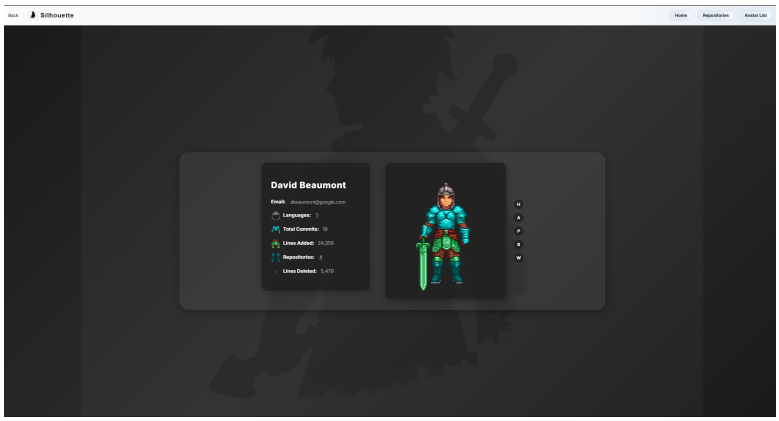


**Figure 5.** Developer detail view showing avatar and activity metrics.

The application is structured around view-based routing. Each core page of the system is defined as a Vue view: `AvatarGallery.vue` (Figure 4) for the main all-developers view, `DeveloperAvatarView.vue` (Figure 5) for detailed individual developer profiles, `ProjectView.vue` (Figure 7) for visualizing developer activity over time in a specific repository, and `ProjectsListView.vue` (Figure 6) for browsing available projects. The `ProjectAvatarCanvas.vue` component is used to render avatars in the project context, while `Header.vue` is used globally to provide consistent navigation.

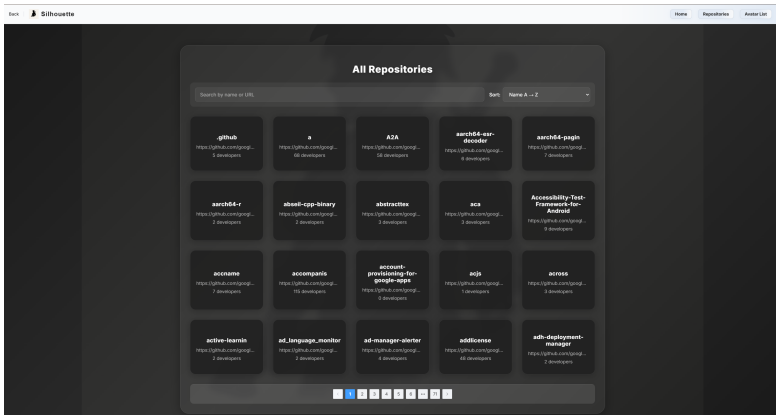


**Figure 6.** Repository list with alphabetical and developer-count sorting.

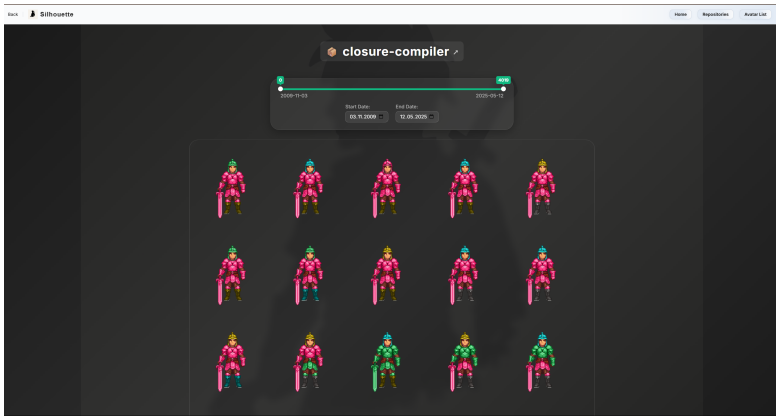


**Figure 7.** Project view showing developer avatars within a selected time interval.

By combining client-side rendering, user-driven customization, and dynamic visual storytelling, the frontend offers an engaging and meaningful way to explore developer activity. It transforms static Git data into visual representations that are intuitive, informative, and enjoyable to interact with.

## 4.4 Mapping Metrics to Visual Traits

Each part of the avatar is linked to a specific developer metric, chosen based on intuitive and thematic reasoning. The goal was to create a mapping that feels natural and visually represents different aspects of developer behavior.

- **Helmet – Programming Languages:** A developer who uses many languages is likely to be versatile and skilled. Since the helmet protects the head, it represents knowledge and adaptability.
- **Armor – Total Commits:** The armor covers the chest, which we associate with long-term effort. A high number of commits shows steady contribution, so we use armor color to reflect this.
- **Pants – Lines Added:** Writing new code is like moving forward or traveling. Pants, which cover the legs, are used to show this effort and progress.
- **Boots – Repositories Contributed To:** Developers who work across many repositories move between projects. Boots are a natural match to show this kind of movement.
- **Weapon – Lines Deleted:** Deleting code often means cleaning up or refactoring. It is not always visible work, but it is important. A stronger weapon reflects that this developer is doing the "dirty work" of maintaining the codebase.

Each of these metrics is converted into a color level ranging from bronze to emerald. To ensure fair and meaningful differentiation, we applied different normalization strategies based on the nature of each metric.

For high-variance numerical metrics such as total commits or lines of code, we observed that raw values were extremely skewed—some developers had tens or hundreds of times more contributions than others. To address this, we applied a logarithmic transformation to compress the range, followed by quantile-based binning using `qcut` to distribute values more evenly across five visual levels.

In contrast, categorical count metrics like the number of repositories or programming languages used showed more compact and clustered distributions. For these, we found that percentile-based ranking provided clearer visual separation and better reflected relative standing without over-amplifying outliers.

We initially experimented with fixed threshold intervals and linear binning, but these methods often led to unbalanced level assignments—either over-representing lower tiers or grouping too many developers into the same visual category. The final method was chosen based on its ability to handle outliers gracefully and produce more visually informative and fair avatars.

These levels are then used to select the appropriate sprite variant for each avatar part. This mapping strategy enables avatars to act as immediate visual summaries of a developer's habits and contributions, making it easier to compare contributors across projects and reinforcing both gamification and analytical goals.

## 4.5 Technologies Used

This project integrates a diverse set of technologies across data mining, backend services, frontend development, and asset generation. Each tool was selected to balance performance, flexibility, and ease of development.

**Data Mining.** To extract commit-level developer activity from Git repositories, we used PyDriller [19], a Python framework tailored for mining version control systems. It allowed us to programmatically access commit metadata, including author details, timestamps, commit messages, and file changes. For post-processing, we used Pandas [6] and NumPy [5] to calculate derived metrics such as total lines added or deleted, and to assign visual levels through logarithmic transformations and quantile-based binning. These tools enabled a clean and scalable preprocessing step before importing data into the database.

**Backend.** The backend was built using Java with the Spring Boot framework [8]. It is responsible for exposing RESTful API endpoints that serve developer data to the frontend, and handles functionality such as filtering, sorting, pagination, and metric aggregation. Data is stored in MongoDB [4], a document-oriented NoSQL database. The schema is divided into three main collections—developers, commits, and repositories—to reflect relationships and allow efficient querying. MongoDB's aggregation pipeline was used to perform statistical computations directly within the database for performance optimization.

**Frontend.** The user interface is implemented using Vue 3 [10], a reactive framework for building modern web applications. It is paired with Vite [9], a next-generation frontend build tool that ensures fast development workflows. All avatar rendering is handled by PixiJS [7], a WebGL-based 2D rendering library that allows for efficient canvas-based visual composition of sprite-based avatars. To provide a consistent and modular UI, we integrated Element Plus [3], which offers a suite of responsive components such as dropdowns, modal windows, and form inputs.

**Avatar Asset Generation.** The visual components of each avatar (e.g., helmet, armor, weapon) were generated using a custom workflow involving AI-based pixel art generation. We used ComfyUI [2], a node-based visual interface

for generative models, to create a consistent full-body RPG-style avatar. These base avatars were then manually segmented into parts using Adobe Photoshop [1] and saved as individual static sprite images. This approach ensured visual consistency across avatar elements while allowing flexibility in how they are dynamically assembled in the browser.

By combining modern web frameworks, efficient data tools, and generative design techniques, the system delivers a seamless experience from backend data processing to frontend visual storytelling.

# 5 Usage Scenario

## 5.1 User Interaction Flow

A typical user journey begins on the home page of the system (Figure 3), where they are invited to explore either developers or repositories. In this scenario, the user is interested in investigating the contributions of a developer named **Sergei Lebedev**.

The user proceeds to the **Avatar Gallery** view (Figure 8), which displays a grid of procedurally generated developer avatars. Using the search bar, the user types "Sergei" and locates the developer of interest. Sergei's avatar is styled with colored components that reflect his activity level across several metrics. The gallery also offers interactive controls for refining the view. Users can search developers by name, email, or username using the search bar. Filters are available to narrow the list by programming language or specific project. Additionally, sorting options allow users to reorder the avatars based on name, total commit count, or number of repositories. These operations are handled on the backend to ensure responsiveness even with larger datasets.

The user may also choose to customize how metrics are visually interpreted by adjusting threshold levels through the settings modal. This feature allows for real-time changes to avatar colors based on user-defined criteria. For example, increasing the commit threshold required for gold armor causes some avatars to downgrade to silver or bronze, helping users fine-tune comparisons based on their own expectations or evaluation needs.

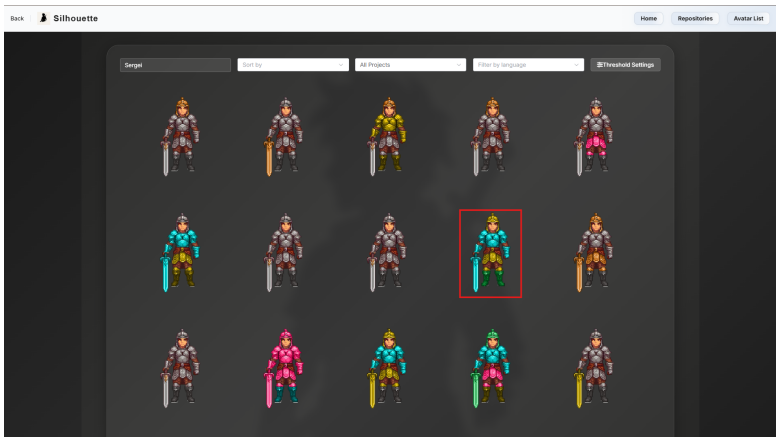


**Figure 8.** Avatar gallery view with search functionality, showing Sergei Lebedev.

Clicking on the avatar opens the **Developer Detail View** (Figure 9), where the user sees Sergei's profile information, total number of commits, lines of code added and deleted, number of repositories, and languages used. Hovering over letter circle components reveals contextual tooltips that explain the reasoning behind each visual element.

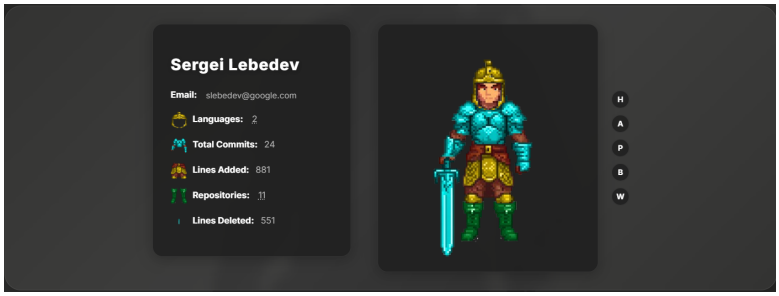


**Figure 9.** Detailed view of Sergei Lebedev's avatar, metrics, and associated repositories.

To gain deeper insight into Sergei's project-specific activity, the user inspects the list of repositories he contributed to (Figure 10). After noting the repository named `flax`, the user navigates to the Projects List View, uses the search bar to locate `flax`, and then selects it to access the project details.

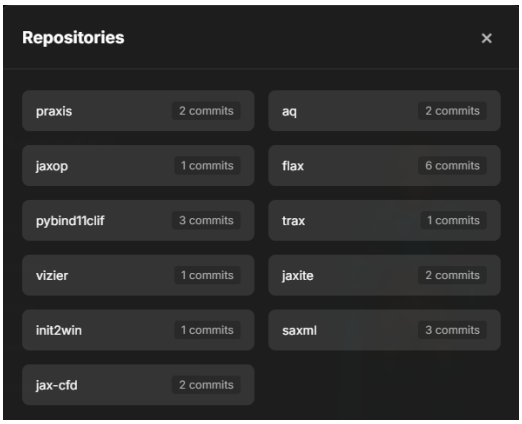


**Figure 10.** List of repositories contributed to by Sergei Lebedev.

In addition to repository-level data, the user can explore Sergei's associated programming languages and individual commit history. By clicking on the language count in the developer detail view, a modal opens showing all languages inferred from Sergei's contributions (Figure 11). This offers a quick and interpretable summary of the developer's technological breadth.

Similarly, clicking on the commit count opens an interactive commit modal (Figure 12), where the user can browse Sergei's individual commits. Each commit card displays the message, repository name, date, and the number of lines added and deleted. The list can be sorted either by commit date or by the number of lines added. Additionally, each entry includes a clickable link that directs the user to the corresponding commit page on GitHub, enabling seamless inspection of code-level activity. (At this stage, external linking is supported only for GitHub-hosted repositories.)

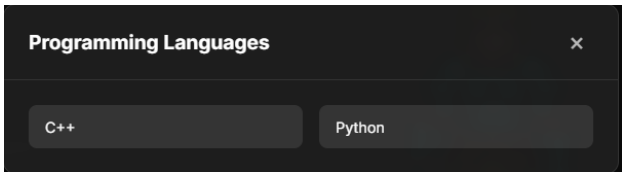


**Figure 11.** Programming languages associated with Sergei Lebedev's contributions.

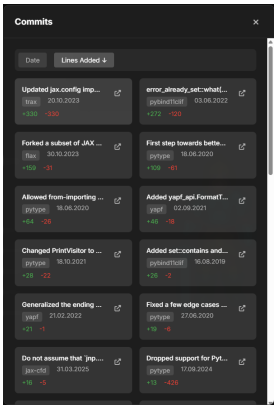


**Figure 12.** Commit-level view of Sergei Lebedev's activity with sorting and external links.

After exploring Sergei's overall activity through commits and language diversity, the user returns to repository-level exploration to examine his role within specific projects more closely.

This leads to the **Project View** (Figure 13), where the system displays avatars of all developers who contributed to the `flax` repository. The user can interact with a timeline slider to filter contributions within a specific time window. As the slider moves, avatars are updated to reflect only the activity within the selected interval. This allows the user to observe how Sergei's visual representation evolves over time and compare his contributions to those of other team members.

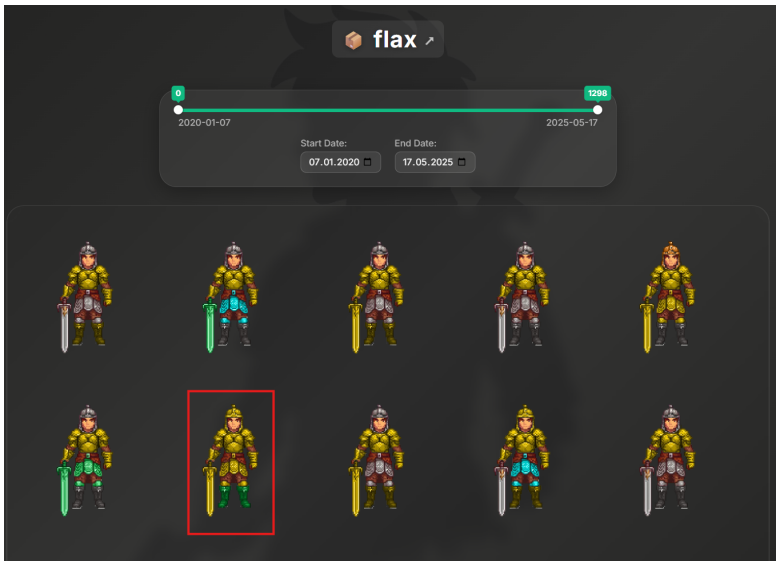


**Figure 13.** Project view for the `flax` repository showing Sergei's contributions over time.

This flow demonstrates how users can search, analyze, and navigate through developer data visually using a combination of interactivity, filtering, and storytelling—without needing to inspect raw statistics or logs.

## 5.2 Avatar Customization Example

One of the key interactive features of the system is the ability to customize the thresholds that map developer metrics to avatar visual styles. This functionality allows users to reinterpret the same developer data from different perspectives, highlighting specific traits or behaviors depending on the thresholds selected.

By default, each avatar part—helmet, armor, pants, boots, and weapon—is assigned a visual level based on precomputed thresholds. These thresholds correspond to metrics such as number of commits, repositories, lines of code added or deleted, and programming languages. For example, a developer who adds many lines of code might receive a more advanced pant color, while someone who works across many repositories is visualized with higher-level color boots.

To demonstrate the impact of customization, we modify the thresholds for the number of repositories and lines added to be more selective. With these new settings (Figure 14), Sergei's avatar is recalculated and updated accordingly. Notably, the color of his pants rises to a higher level due to the looser threshold for lines added, while his boots shift to a lower tier because the new repository threshold is stricter.

This example Figure 15 illustrates how adjustable thresholds empower users to control how developer traits are emphasized in the visualization. It supports comparative exploration and provides flexibility to tailor avatar interpretation to different team norms, scales, or evaluation goals.

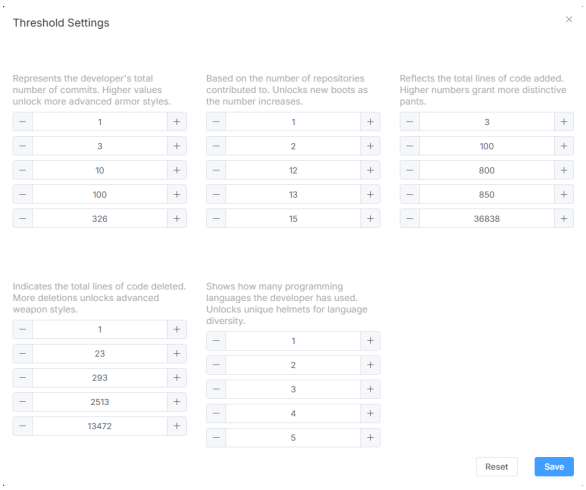


**Figure 14.** Customized threshold settings.

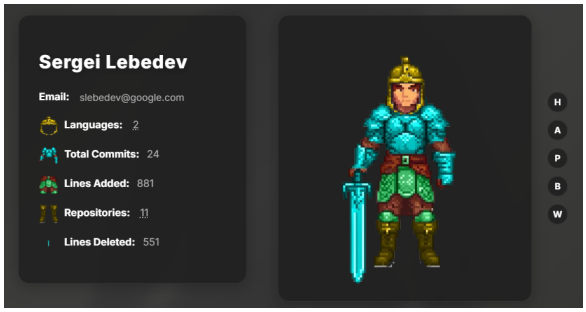


**Figure 15.** Sergei Lebedev's avatar appearance after threshold adjustment.

## 5.3 Developer Comparison Use Case

While metrics such as commit count or code churn provide quantitative insights, the avatar-based visualizations in the system allow for more intuitive and context-rich comparisons between developers — especially when combined with real project data.

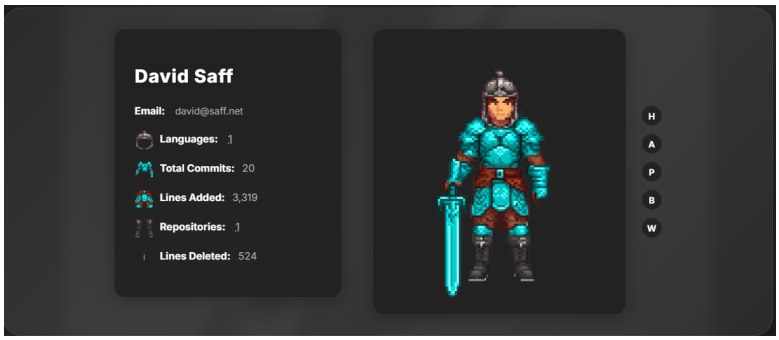


**Figure 16.** David Saff: Low commit count, but efficient high-impact contributions.

In one example, we examine **David Saff** (Figure 16), a developer with only 20 commits and a single repository. At first glance, his avatar may not appear extraordinary. However, closer inspection of his Git history reveals that a large portion of his added lines — over 3,000 — came from test file development and license inclusion. In fact, just four commits account for most of his contributions, including a single commit with 1,438 added and 275 deleted lines. Despite the low commit count, his focused and high-impact contributions paint a picture of an efficient, specialized role — one that might be overlooked in traditional dashboards.

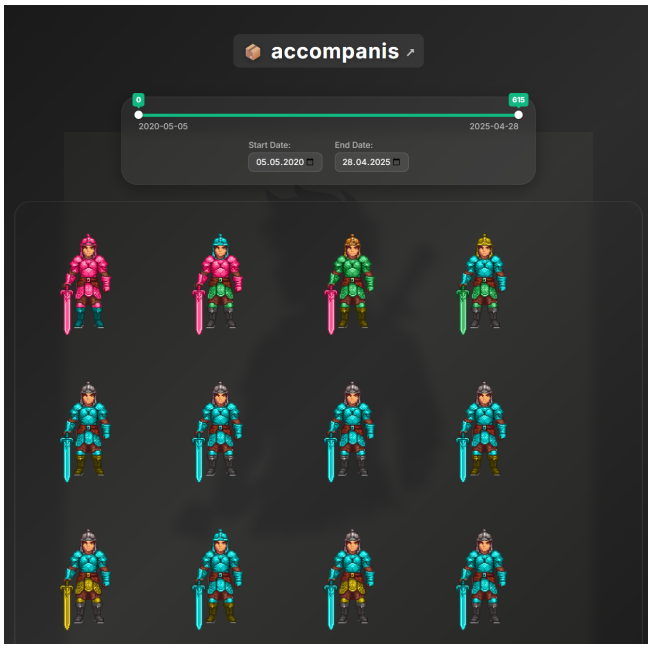


**Figure 17.** Disproportionate effort visible in the `accompanis` repository.

In contrast, a visit to the `accompanis` repository tells a different story (Figure 17). Among the developers, one avatar stands out dramatically — adorned with advanced-tier armor, boots, and weapon. This developer made **886 commits**, added over **100,000 lines**, and deleted more than **55,000**, dwarfing their peers. While other contributors show up in the data, their activity pales in comparison. This lone figure's dominance is clearly reflected in their avatar, showcasing how the system can visually surface disproportionate effort and code ownership within collaborative repositories.

Together, these cases illustrate how avatar comparisons go beyond raw numbers. They help highlight different types of contributors: the quiet specialist, the team's backbone, or the occasional cleaner — all through a format that's immediately accessible and visually expressive.

# 6 Evaluation

## 6.1 Evaluation Metrics

The effectiveness of our system can be evaluated across three key dimensions: representational accuracy, system performance, and interpretability. Representational accuracy refers to how well the avatars visually encode developer metrics. Performance measures how efficiently the system handles data processing and rendering. Interpretability focuses on whether users can extract meaningful insights by visually comparing avatars.

## 6.2 Avatar Representation Accuracy

Our avatars aim to reflect developer contributions through the visual encoding of five metrics: total commits, lines added, lines deleted, number of repositories, and language diversity. These are mapped respectively to the armor, pants, weapon, boots, and helmet.

To assess how accurately these mappings communicate developer activity, we analyze concrete use cases from real data. One such example involves the **Google Fonts** repository, a design-centric project with substantial code churn. Figure 18 shows avatars for its top contributors. Although many developers share similar commit counts, their avatars highlight significant differences in deleted lines and number of repositories. The high line churn — over 900,000 lines added and deleted across the project — is consistent with large asset updates (e.g., font files) rather than algorithmic development. This case demonstrates how avatars can reflect the character of a repository, alerting users that high activity does not always imply complex contributions.

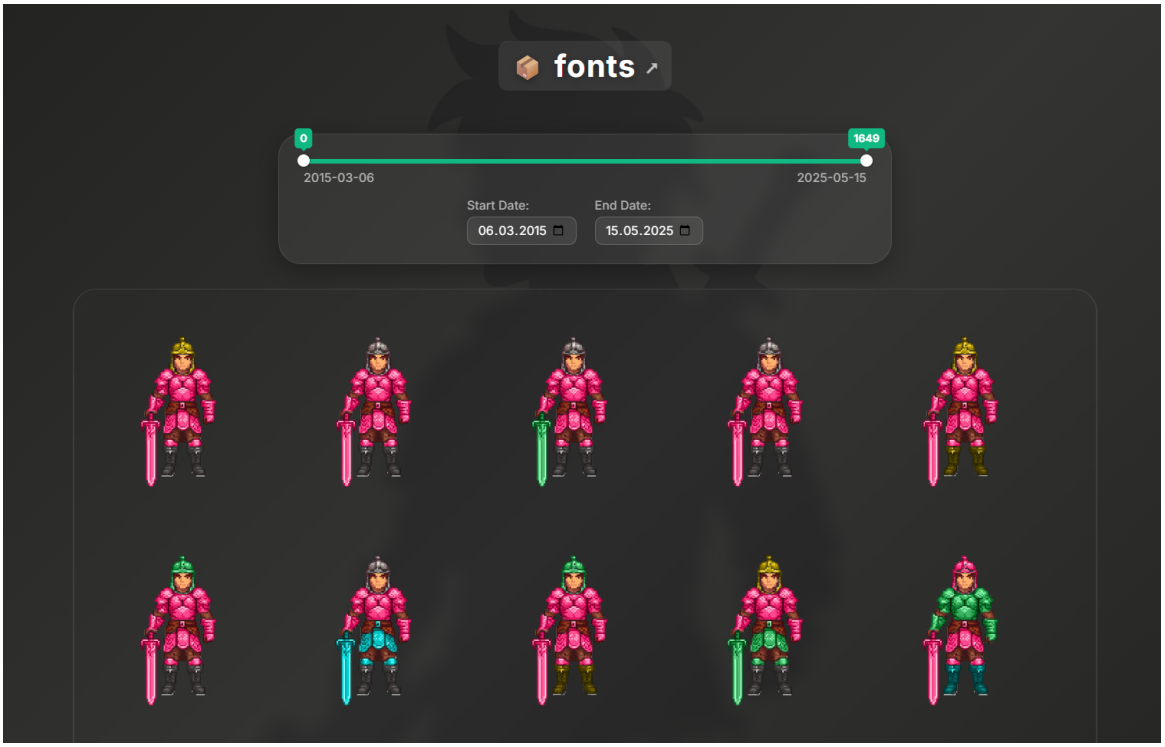


**Figure 18.** Contributor avatars in the Google Fonts repository, where high code churn is driven by large file updates.

Another example is **Yilei Yang**, shown in Figure 19. Despite contributing to **31 repositories**, his total commits and lines added are relatively low. His avatar reflects this with a basic helmet (low language diversity), vibrant boots (indicating broad repository presence), and moderate armor and pants. This contrast illustrates how the system highlights different contribution patterns—e.g., breadth versus depth—without requiring users to inspect raw numbers.

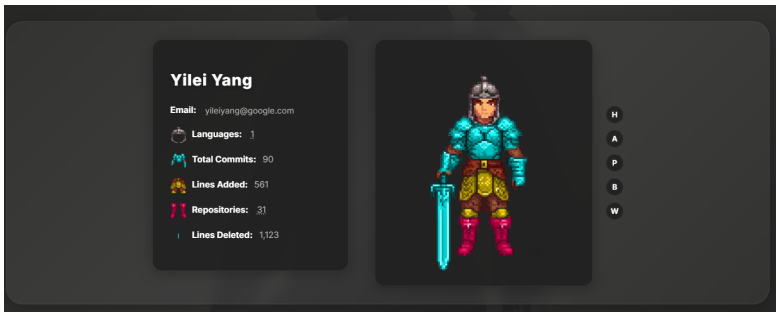


**Figure 19.** Avatar of Yilei Yang: many repositories, few commits, and minimal code contribution.

These examples show that the visual traits successfully encode a variety of developer behaviors. Users can infer work style, impact, and participation type at a glance, even without examining underlying metrics.

## 6.3 System Performance

The system is designed for efficiency across both backend and frontend. MongoDB aggregation pipelines allow the backend to compute commit statistics in bulk, minimizing the load on the application server and reducing query complexity. REST APIs expose filtered, sorted, and paginated developer data without transmitting unnecessary information.

On the frontend, PixiJS enables GPU-accelerated rendering, allowing multiple avatars to be drawn simultaneously with smooth visual performance. All level calculations and asset selection are handled client-side using preprocessed developer data, keeping the interface responsive.

Although formal performance testing has not been conducted, the system demonstrates fast response times and fluid interaction under typical use with hundreds of developers. The architecture supports scalability, and further optimizations could be introduced if deployed at a larger scale.

# 7 Conclusion and Future Work

## 7.1 Future Improvements

Although the system fulfills its primary goals, there are several areas where it can be enhanced and extended:

- **Expanded Avatar Vocabulary:** Currently, each avatar uses five components to encode developer traits. Introducing additional parts—such as shields for issue resolution, banners for code reviews, or capes for mentorship—would allow for richer representations of diverse developer roles and contributions.
- **Real-Time and Periodic Updates:** The current miner runs offline, capturing a snapshot of contributions. In future iterations, the system could support scheduled or real-time Git mining to reflect ongoing development and evolution of avatars as contributions are made.
- **Advanced Metric Layers:** Beyond commit size and code churn, the system could incorporate semantic features such as commit message analysis, pull request frequency, test coverage, or co-authorship networks. These dimensions would enhance avatar interpretability and depth.
- **Visual Dynamism and Animation:** Adding subtle animations or transitions—such as shimmering armor for high activity or sword sparkles for impactful deletions—could make avatars more lively and visually responsive to user interaction.
- **User Accounts and Personalization:** Adding login support would allow users to save threshold configurations, follow specific developers, mark favorites, and track their own avatar evolution over time.
- **Public Deployment and Scale Testing:** The current system has been tested locally with a subset of GitHub repositories. Deploying it as a public platform and collecting performance benchmarks under heavier loads would inform further scalability improvements.

These future directions highlight the flexibility and potential of the avatar-based representation paradigm. By continuing to blend gamification, software analytics, and interactive storytelling, the project paves the way for tools that are not only functional, but also motivating and visually engaging for developers, educators, and managers alike.

## 7.2 Conclusion

This project presented a novel system for transforming static Git contribution data into visually expressive, RPG-style avatars that reflect developer behavior, experience, and coding habits. By analyzing key metrics such as total commits, lines of code added or deleted, programming languages used, and repository diversity, the system builds avatars where each component—helmet, armor, boots, pants, weapon—symbolizes a distinct developer trait. This approach introduces an intuitive and engaging alternative to traditional charts or dashboards.

We designed and implemented an end-to-end architecture encompassing offline data mining with PyDriller, scalable metric aggregation on a Spring Boot backend, and highly interactive avatar rendering with Vue.js and PixiJS on the frontend. Users can explore developers individually or in group contexts, with filtering and sorting capabilities, threshold customization, and time-based visualizations. The system brings to life an otherwise abstract dataset, offering a gamified interface where developers are represented not only by numbers but by stylized avatars whose appearance tells a data-driven story.

The value of this approach was demonstrated through usage scenarios and case studies. We showed how the system can highlight lone contributors, identify superficial commit patterns, contrast contributors within the same project, and expose anomalies in developer behavior. By combining storytelling, interactivity, and analytics—including detailed, navigable commit views that enhance transparency—the system helps users build a richer and more trustworthy understanding of software contribution dynamics.

# References


[1] Adobe photoshop. https://www.adobe.com/products/photoshop.html.

[2] Comfyui - a node-based ui for generative models. https://github.com/comfyanonymous/ComfyUI.

[3] Element plus ui framework. https://element-plus.org.

[4] Mongodb nosql database. https://www.mongodb.com.

[5] Numpy. https://numpy.org.

[6] pandas: Python data analysis library. https://pandas.pydata.org.

[7] Pixijs: The html5 creation engine. https://pixijs.com.

[8] Spring boot. https://spring.io/projects/spring-boot.

[9] Vite - next generation frontend tooling. https://vitejs.dev.

[10] Vue.js - the progressive javascript framework. https://vuejs.org.

[11] M. V. Birk and R. L. Mandryk. Fostering intrinsic motivation through avatar identification in digital games. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2982–2995. ACM, 2016.

[12] T. Dal Sasso, A. Mocci, M. Lanza, and E. Mastrodicasa. How to gamify software engineering. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 261–271. IEEE, 2017.

[13] EduTechWiki. Gamification in education. 2020. Accessed from https://edutechwiki.unige.ch/en/Gamification_in_education.

[14] J. Hamari, J. Koivisto, and H. Sarsa. Does gamification work?–a literature review of empirical studies on gamification. *Proceedings of the 47th Hawaii International Conference on System Sciences*, pages 3025–3034, 2014.

[15] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. *Empirical Software Engineering*, 21:2035–2071, 2016.

[16] L. Merino, R. Robbes, and M. Lanza. Towards actionable visualization for software developers. *Journal of Software: Evolution and Process*, 29(1), 2017.

[17] L. Moldon, S. C. Wagner, E. Guzman, and A. Joorabchi. How gamification affects software developers: Cautionary evidence from a natural experiment on github. *arXiv preprint arXiv:2006.02371*, 2021.

[18] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 2017.

[19] D. Spadini, M. Aniche, and A. Bacchelli. Pydriller: Python framework for mining software repositories. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 502–506. IEEE, 2018.