Reactify: An Intelligent Platform for Agile Requirements Engineering Based on REACT and REACT-M

Katlen V. dos Santos da Silva
Departamento de Ciências Exatas e Tecnológicas (DCET), Universidade Federal do Amapá (UNIFAP)
Macapá, Amapá, Brasil
katlenvanessa15@gmail.com

Luma G. Andrade da Silva
Departamento de Ciências Exatas e Tecnológicas (DCET), Universidade Federal do Amapá (UNIFAP)
Macapá, Amapá, Brasil
lumagabriela3331@gmail.com

Julio Cezar Costa Furtado
Departamento de Ciências Exatas e Tecnológicas (DCET), Universidade Federal do Amapá (UNIFAP)
Macapá, Amapá, Brasil
furtado@unifap.br

Abstract

Requirements quality remains an open problem in agile development, where user stories are often incomplete, ambiguous, or weakly aligned with business goals. Existing tools fall short: backlog platforms scatter artifacts, and standalone artificial intelligence (AI) prototypes stay disconnected from any established methodology. The REACT and REACT-M methodologies organize requirements development and maintenance into disciplined ceremonies, but their adoption is hindered by scattered artifacts, manual interview transcription, and the absence of continuous quality control. This paper presents Reactify, a web platform that operationalizes REACT and REACT-M in a single environment and embeds AI across the elicitation, refinement, and modeling ceremonies while keeping inspection under human control. It combines artifact generation from interview transcripts, a conversational assistant backed by retrieval-augmented generation, and an INVEST-based quality gate reviewed manually before any story enters a sprint. The tool has so far been used internally to verify feasibility, and a controlled empirical study remains as future work. **Demonstration video:** https://zenodo.org/records/20289495.

Keywords

Requirements Engineering, Agile Methodologies, REACT, Artificial Intelligence, RAG, Artifact Generation

1 Introduction

Requirements engineering is among the most consequential activities in software development, with direct effects on the quality, cost, and schedule of projects. In agile settings, user stories have become the dominant artifact for requirements, as documented by a systematic mapping of 186 studies [1] and by reports of adoption in complex domains [6]. Despite this uptake, requirements quality remains an open problem: studies routinely report stories that are incomplete, ambiguous, and weakly aligned with business goals [15].

The REACT methodology, introduced by Santos [12], addresses this problem by organizing requirements development into a sequence of ceremonies, namely Inception, Story Discovery, Refining, Modeling, and Inspection, and was later extended by Silva [13] to the maintenance phase as REACT-M, with change management and goal-driven prioritization. We adopt REACT and REACT-M because they are agile requirements methodologies of Brazilian origin, defined and evaluated through empirical studies, and because their explicit ceremonies and roles make the process naturally amenable to tool support, which the absence of a dedicated environment has so far left unrealized. Consolidating the method into a single environment is therefore both a practical need for teams already using it and the specific gap that this paper addresses. The method defines three human roles: the Customer, who sponsors the product and validates vision and priority; the Domain Expert, who conducts the elicitation ceremonies; and the Team, which performs modeling and execution. In practice, REACT faces three recurring obstacles: artifacts scattered across heterogeneous tools, the manual effort of transcribing and analyzing interviews, and the absence of continuous quality control over the resulting requirements.

Recent work has explored language models to mitigate some of these obstacles, with systematic guidelines for classification, generation, and ambiguity-detection tasks [14], automated assistance for improving requirements completeness [9], and conversational agents to enhance user story quality [16]. Relatedly, retrieval-augmented generation [7] has been applied to related problems, such as the recovery of traceability links between requirements and code [5]. These proposals tend to remain standalone prototypes, disconnected from any established agile methodology and lacking native integration across elicitation, refinement, modeling, and inspection.

This paper presents Reactify, a web platform that operationalizes REACT [12] and REACT-M [13] in a unified environment and treats artificial intelligence as a first-class part of the team's workflow. The tool combines three forms of support: automatic artifact generation from interview transcripts, a conversational assistant backed by retrieval-augmented generation, and a manual INVEST audit recorded as a hard gate before any story enters a sprint. The architecture separates a reactive front end, a back end for business rules, and microservices dedicated to natural language processing, with relational storage to ensure hierarchical traceability across artifacts.

We make four contributions. First, a complete operationalization of REACT and REACT-M in a single platform, removing external tools from the methodological flow. Second, an integration of language models with the project content, enabling contextualized artifact generation that avoids fabricated content. Third, an internal auditing mechanism, performed manually by the team, that records requirements conformity through verifiable INVEST criteria and acts as a hard gate before any story enters a sprint. Fourth, a direct link between textual requirements and visual prototypes through an integrated prototyping area, which keeps the two representations consistent from elicitation onward.

2 Related Work

Work related to Reactify spans three areas: the assessment of user story quality, the application of language models to requirements engineering, and the use of retrieval-augmented generation for traceability.

Regarding user story quality, the systematic mapping by Amna and Poels [1] identifies ambiguity and low quality as recurring obstacles and points to gaps in mature solutions. Lucassen et al. [8] propose the Quality User Story framework, a set of thirteen criteria, together with a companion tool that applies natural language processing to detect defects in user stories automatically, although the check runs outside any methodological flow. Xu et al. [15] propose a quantitative assessment method grounded in the INVEST heuristic, with criteria amenable to automation through natural language processing. Kannan et al. [6] report the practical application of the same heuristic in agile development and show quality gains when the criteria are revisited at every iteration. These works establish INVEST as a reference but treat auditing as an external activity, disconnected from the elicitation flow.

For language models in requirements engineering, Vogelsang and Fischbach [14] offer systematic guidelines for classification, generation, and ambiguity-detection tasks. Luitel et al. [9] investigate automated assistance for improving requirements completeness. Zhang et al. [16] evaluate language-model agents for user story quality across six industrial teams and report positive acceptance. These proposals remain standalone prototypes, with no anchoring in an established agile methodology and no hierarchical traceability.

In the retrieval-augmented generation strand, Hey et al. [5] recover traceability links between requirements and code with results that outperform approaches based on textual similarity alone. Their work focuses on post hoc traceability and does not address the earlier stages of the cycle.

Reactify integrates these three areas within a single environment aligned with REACT and REACT-M. INVEST auditing is integrated into the workflow as an explicit human-controlled step, and the AI features operate over the project content across every other ceremony. Traceability emerges from elicitation itself, rather than being recovered after the fact.

Tables 1 and 2 situate Reactify against representative tools currently used to support agile requirements work. In both tables, each row states a capability relevant to methodology-aligned requirements work, and each tool is marked Y when it provides the capability natively, P when support is partial or indirect, and N when it is absent. Industrial platforms such as Jira [3] with Rovo [2], Azure DevOps [10] with Copilot4DevOps [11], and boards such as Trello [4] offer mature backlog management and, in the first two cases, generative features for drafting stories, expanding acceptance criteria, and breaking down epics. Among academic proposals with running prototypes, Luitel et al. [9] target completeness improvement of individual requirement statements, Zhang et al. [16] apply language-model agents to enhance user story quality, and Hey et al. [5] recover traceability links after the fact via retrieval-augmented generation. None is anchored in a specific agile method for requirements development, none generates artifacts from interview transcripts, and quality auditing, when present, is executed by the model rather than recorded as an explicit human decision tied to a hard gate before a sprint. Reactify is designed precisely around this gap.

Table 1: Comparison of Reactify with commercial tools. Y: capability supported natively; P: partially or indirectly supported; N: absent.

Capability	Reactify	Jira +Rovo	ADO +C4DO	Trello
Anchored in a RE method	Y	N	N	N
Generation from interviews	Y	N	N	N
Contextual RAG assistant	Y	P	P	N
Hierarchical traceability	Y	P	P	N
Human INVEST hard gate	Y	N	N	N
Story-linked prototyping	Y	N	N	N
Multi-provider LLM support	Y	N	N	N

Table 2: Comparison of Reactify with academic proposals. Y: capability supported; N: absent.

Capability	Reactify	Luitel [9]	Zhang [16]	Hey [5]
Anchored in a RE method	Y	N	N	N
Generation from interviews	Y	N	N	N
Contextual RAG assistant	Y	N	N	N
Hierarchical traceability	Y	N	N	N
Human INVEST hard gate	Y	N	N	N
Story-linked prototyping	Y	N	N	N
Multi-provider LLM support	Y	N	N	N

3 Tool Architecture

Reactify's architecture is modular and hybrid, separating the user interface, the business rules core, and the dedicated AI services. The term hybrid denotes two deliberate combinations: server-side rendering paired with client-side reactivity in the interface, and commercial language models paired with locally hosted ones in the AI layer. This layered design isolates computationally intensive components from the critical interaction path and lets each layer scale on demand. Figure 1 shows the overall organization.

User interface. The presentation layer is built on React together with the Tailwind CSS utility-first framework, allowing declarative component composition and native responsiveness. Communication with the back end uses Inertia.js, a rendering adapter that combines initial server-side mounting with client-side reactivity.

Business rules and persistence. The application core is managed by the Laravel PHP framework, which handles authentication, authorization, validation of business rules, and orchestration of AI activities. The relational store, configured on PostgreSQL, holds the entities needed for methodological traceability under REACT and REACT-M, and extends the same engine with vector indexing through the pgvector extension. Keeping artifacts, embeddings, and relationships in one transactional cycle avoids running a separate vector database.

Asynchronous processing and real-time notifications. AI operations often take tens of seconds to minutes, so the architecture routes them to a database-backed queue executed by dedicated workers. At the end of each stage, back-end events propagate to the client through an embedded WebSocket server, so the interface reflects progress on transcriptions, artifact generation, and assistant responses without page reloads.

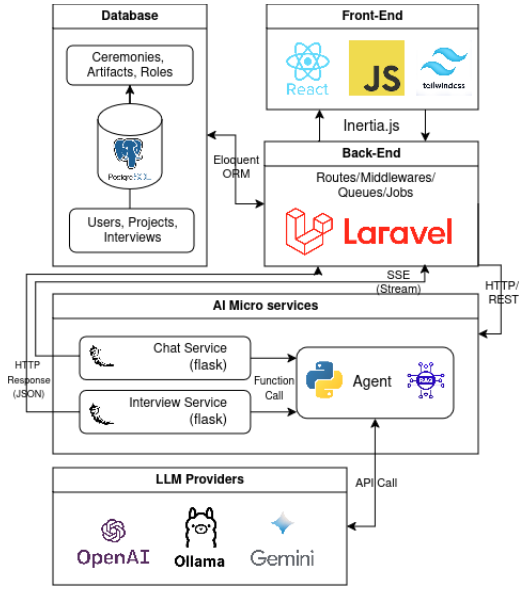

Figure 1: Overall architecture of Reactify, showing the separation between the reactive front end, the unified back end, the AI microservices, and the external providers of language models.

AI microservices. Natural language processing flows live in two Python services built with Flask. One handles the conversational assistant, built atop the LangChain agent-orchestration framework that coordinates calls to language models and auxiliary tools. The other handles the interview pipeline, with three configurable stages: raw content extraction, transcription by a speech recognition model, and post-processing by a local reasoning language model that normalizes and segments the text.

Language models and retrieval-augmented generation. The platform draws on a multi-provider infrastructure. For artifact generation and for the Facilitator agent, both orchestrated with LangChain, it can route requests to commercial providers, Google (Gemini) and Anthropic (Claude), or to locally hosted models served through Ollama, such as Qwen3 and gpt-oss, interchangeable by project configuration. The internal verification reported here used a single provider, Google Gemini 2.5 Flash. Interview transcription runs on Whisper, a local faster-whisper deployment by default, with the OpenAI transcription API as an alternative, and the cleanup and segmentation of the transcript use a local Qwen3 model served through Ollama. Context vectorization for the retrieval-augmented generation pipeline [7] uses the OpenAI text-embedding-3-small model, stored alongside the textual artifacts in PostgreSQL through pgvector. This setup supports trade-offs between cost, latency, and quality for each task and avoids exclusive dependence on a single vendor.

4 Features

Reactify organizes the team's work around the five REACT ceremonies, Inception, Story Discovery, Refining, Modeling, and Inspection, together with the execution stages of Product Backlog and Sprint. Each ceremony has a dedicated screen with its own artifacts, and the navigation flow follows the progression of the method. Hosting these modules within a single environment, on a shared relational persistence layer, enables end-to-end hierarchical traceability from business goals down to the stories running on the Kanban board (Figure 2F).

What sets the tool apart is the cross-cutting presence of AI support: nearly every artifact screen offers a generation action, and every module can summon a contextual conversational assistant, while the INVEST audit is deliberately reserved for the Team. The rest of this section describes first the AI mechanisms that operate across the ceremonies and then the artifacts specific to each one.

Contextual artifact generation. Personas, business goals, journeys, user stories, business rules, usage scenarios, epics, classes of the overall model, system interfaces, and the product canvas itself can be produced by language models. Generation is conditional: each artifact type has specific preconditions. Story generation, for example, requires a transcribed interview, personas with goals, journeys with steps, and business goals on file; when any input is missing, the interface lists what is needed instead of producing thin content. Generated items appear as suggestions marked with a visual flag, and the user reviews them individually, approving, editing, or discarding each one. Regeneration is incremental: on a new trigger, the model receives the already approved set as context and produces only complements, avoiding duplication.

Contextual conversational assistant. The Facilitator, shown in Figure 2D, is always reachable through a floating icon on the side of the screen. Two properties set it apart from a generic chat. First, the assistant knows which screen the user is on: the system prompt is built at runtime by concatenating the active ceremony name, the artifact in focus, the role of the requesting user, and a short methodological description of the activity retrieved from the REACT knowledge base. Second, the assistant has three tools scoped to the project: querying the data of any registered artifact, querying the interviews, and searching a REACT knowledge base via retrieval-augmented generation. The retrieval pipeline splits the source material recursively, preferring paragraph and line boundaries, and vectorizes each segment with a commercial embedding model stored alongside the textual artifacts in PostgreSQL through pgvector. At query time, the segments most similar to the question are passed to the language model as grounding context. The assistant operates strictly in an advisory capacity: its three tools read project data without modifying it, it cannot create or approve artifacts, and it cannot advance a story through the workflow. Any content it suggests reaches the project only after a team member reviews and confirms it, so an incorrect or low-confidence response is caught at the same manual checkpoint that governs generated artifacts and never bypasses the human-controlled INVEST gate. A typical reply combines a diagnosis of the current state with a recommended next step, as shown in Figure 2D, where the Facilitator flags personas without linked stories, stories without business rules, and items with pending INVEST.

Provider flexibility. The generation layer supports multiple language model providers, interchangeable by configuration, which avoids exclusive dependence on a single vendor and supports trade-offs among cost, latency, and quality depending on the task.

Inception. The ceremony opens with the interview module, which accepts PDF, MP3, and MP4 files and runs a three-stage pipeline of raw extraction, speech recognition, and language-model cleanup; the transcript is reviewed manually and then indexed in the project's vector store, becoming the primary input for later generation. On the Product Canvas the team records the high-level business view, problems, solutions, personas, constraints, and product scope, and then fills in personas, business goals, and journeys. Every story created later must link back to a persona and a goal, so hierarchical traceability begins in this ceremony.

Story Discovery. The team registers user and system stories, distinguished by identifier prefixes and tied to their parent artifacts; when a story is broken down, its children receive composite identifiers that preserve the hierarchy, and epics derive from parent stories. Assisted generation is offered here only when its preconditions are met, otherwise the interface lists what must be completed first.

Refining. For each prioritized story the team records business rules and usage scenarios in the Given-When-Then format, and creates epics when a single story exceeds the capacity of a sprint.

Modeling. The architectural design is captured through cards of responsibilities and collaborations. Visual prototyping happens within the platform through an embedded drawing library: a side panel lists the project's stories and links a textual requirement directly to a newly created storyboard, so consistency between the two representations no longer needs to be maintained manually.

Inspection. Auditing stories against the INVEST criteria [6, 15] is kept out of AI scope by design: this is the only stage where AI does not touch the artifact. The Team marks each of the six criteria as Yes, No, or Not Applicable in a dedicated modal, and the Inspection screen in Figure 2G consolidates the status for every story. Assessing criteria such as independence, value, and negotiability depends on human judgment about business context and team state, and handing that decision to a model would weaken the gatekeeping role of inspection. The practical consequence is a hard gate: a story can enter a sprint only after all criteria are marked Yes or Not Applicable, a check performed on the server before any state change. Beyond inspection, other decisions are kept human by design: prioritization on the matrix, role assignment, and the approval of every generated suggestion, so the platform proposes artifacts but never commits them autonomously.

5 Usage Example

This section illustrates the tool in practice through a complete elicitation, refinement, and prioritization scenario aligned with REACT-M. The scenario follows a streaming platform project for public-domain films from initial registration to the first executable sprint. It traverses the five ceremonies of the method and exercises the three human roles introduced in Section 1: Customer, Domain Expert, and Team. The conversational assistant takes the non-human role of Facilitator. Figure 2 shows the main screens involved in the flow described below.

The scenario begins with the creation of a project named "ClassicFlix", a streaming platform dedicated to public-domain films. The platform then renders a work panel with per-ceremony progress and an alert that the Inception artifacts are still empty. The Domain Expert invites the remaining participants and assigns each a role, and permissions for every ceremony follow automatically.

The first relevant action is the upload of an audio interview with the Customer, conducted by the Domain Expert. The system runs the three-stage pipeline, presents the transcript for manual correction, and, once confirmed, indexes it in the project's vector store. From this point on, the interview content informs the rest of the work.

With the interview available, the Domain Expert conducts the Inception ceremony. The Product Canvas in Figure 2E captures the vision, the problem, and the constraints, including, for instance, the restriction to works with expired copyright. Next, on the Business Goals screen, the Domain Expert triggers assisted generation and receives suggestions such as "reach ten thousand subscribers within one year" and "comply with copyright law and public-domain content regulations", each tagged with a visual marker indicating its AI origin. The Domain Expert reviews and edits each suggestion, then moves to the Personas and Journeys screens, where the procedure repeats. Regeneration is incremental: on the second run for Personas, the model receives the already approved personas as context and produces complementary profiles rather than redrafting them, preventing duplication and giving a steadily growing, non-redundant set. Figure 2B shows a persona's detail view, with profile, expectations, goals, and the user stories linked to it. Persona and journey generation runs asynchronously, and the interface updates in real time without page reloads.

The Story Discovery ceremony is shown in Figure 2A. On this screen, the Domain Expert sees the user stories and system stories already registered, with identifier prefixes such as US and SS that preserve traceability. AI generation is available only when preconditions are met: a transcribed interview, personas with goals, journeys with steps, and business goals on file; if any is missing, the interface lists what must be completed first. In Figure 2A, story US8 is broken into child story US8.1, whose composite identifier preserves the link to the parent requirement.

Detailing happens in the Refining ceremony, jointly conducted by the Domain Expert and the Team. For each prioritized story, the team records business rules, usage scenarios in the Given-When-Then format, and epics when a single story exceeds the capacity of a sprint. Modeling is the Team's responsibility: the architectural design is captured through cards of responsibilities and collaborations, where entities such as Film, Catalog, and Video Player are modeled with their responsibilities and dependencies. The Team also builds visual storyboards with the embedded drawing library: a side panel lists the project's open stories and links a new storyboard to a chosen story; later edits to the requirement text surface next to the linked sketch, so the two representations stay together and cannot drift apart unnoticed.

INVEST inspection is performed manually by the Team for each story, on the screen shown in Figure 2G. The team marks Yes, No, or Not Applicable for each of the six criteria, Independent, Negotiable, Valuable, Estimable, Small, and Testable, and adds a brief justification when the verdict is No. In the scenario, story US4 (cross-browser video playback) is marked No on Independent, since it presupposes the user-account story US1, and on Small, since it cannot fit a single sprint; the platform refuses to add US4 to the next sprint and routes it back to Refining, where it is later split. A

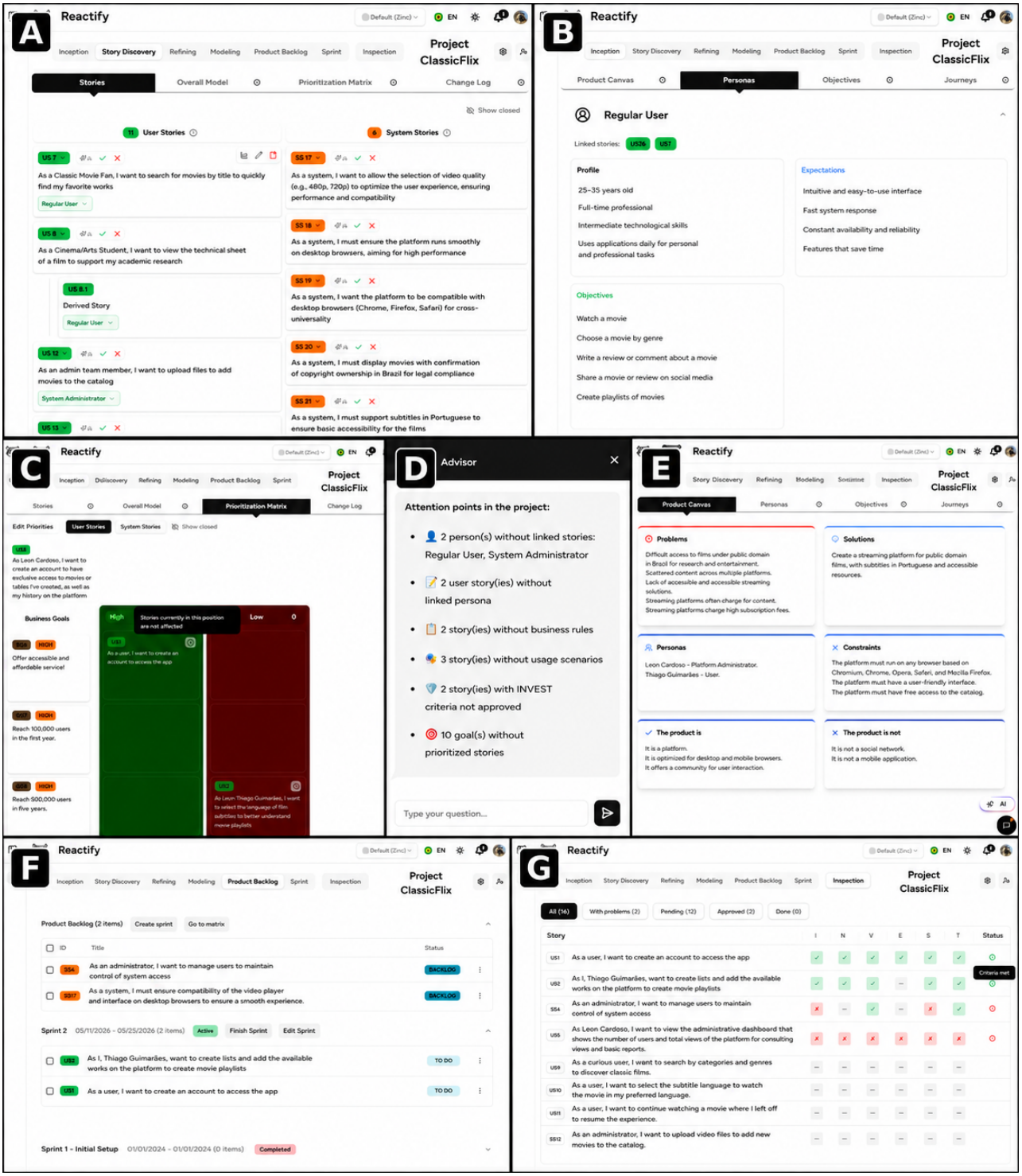

Figure 2: Structural view of Reactify: (A) Story Discovery tab listing user and system stories; (B) Inception screen detailing a persona profile; (C) prioritization matrix organizing scope across business goals; (D) Facilitator assistant suggesting an audit; (E) Product Canvas structuring the business view; (F) Product Backlog panel with sprints; (G) Inspection screen monitoring approval against quality criteria.

story becomes eligible for a sprint only once all criteria are marked Yes or Not Applicable, a check the server enforces on every attempt to move a story into a sprint, regardless of how the request is made. This is the only stage in the workflow that deliberately excludes AI from the decision.

At any point, any participant can summon the Facilitator (Figure 2D). Asked for a general check, it combines artifact data with methodological knowledge from the vector store and produces a report of attention points. In the illustrated example, the assistant flags two personas without linked stories, two stories without business rules, three stories without usage scenarios, two stories with pending or rejected INVEST, and ten goals without prioritized stories, with a suggested next step for each category.

The flow concludes at the prioritization matrix (Figure 2C), where the Customer and the Domain Expert work together: the Customer drags a story card to the cell where a business goal meets a priority level, for example pairing "reach ten thousand subscribers within one year" with High priority. The action binds the story to the business goal and releases it to the product backlog, shown in Figure 2F. From this panel, the Team creates two-week sprints, pulls prioritized stories, and tracks progress on a Kanban board (In Progress, Testing, Done). When a sprint ends, a modal lets the Team return incomplete stories to the backlog or roll them into the next sprint, closing the REACT-M cycle.

6 Conclusion

This paper has presented Reactify, a web platform that operationalizes REACT and REACT-M within a single environment with native AI support. The tool addresses three recurring obstacles in the practical adoption of REACT: artifacts scattered across heterogeneous systems, the manual effort of transcription, and the absence of continuous quality control. The three forms of support introduced in the paper, assisted artifact generation, a conversational assistant backed by retrieval-augmented generation, and a manual INVEST audit recorded as a hard gate to sprints, ease the team's workload without compromising methodological rigor.

The proposal has known limitations. So far the tool has been used only internally, to verify the feasibility of the approach; no controlled empirical evaluation has been carried out yet, and such a study with academic and industrial teams is the most important threat to the strength of the claims made in this paper. Pending that study, quality decisions remain anchored by the human-controlled INVEST gate and the mandatory review of every generated artifact, which keep the team, rather than the model, responsible for what enters a sprint. The generation features depend on external language model providers, which introduces sensitivity to vendor pricing, latency, and policy changes, partially mitigated by the multi-provider design. The methodological coverage is restricted to REACT and REACT-M and does not yet address other agile or hybrid practices. Finally, the quality of generated artifacts is bounded by the quality of the input interviews, since incomplete or contradictory stakeholder material will propagate into the suggestions reviewed by the team; the manual transcript review step and the preconditions that refuse to generate thin content partially contain this effect.

As future work, we plan an empirical study with academic and industrial teams to measure the impact of Reactify on team productivity and on the quality of elicited requirements. We also plan to broaden methodological coverage to other agile practices and to add automated detection of textual ambiguity and inter-artifact inconsistency.

License. Reactify is released under the GNU Affero General Public License v3.0 (AGPL-3.0), with commercial licenses available for proprietary use.

Artifact Availability

The source code is available at https://github.com/jcfurtado86/reactify, and the demonstration video is preserved at https://zenodo.org/records/20289495.

References

[1] Anis R. Amna and Geert Poels. 2022. Systematic Literature Mapping of User Story Research. *IEEE Access* 10 (2022), 51723–51746. doi:10.1109/ACCESS.2022.3173745

[2] Atlassian. 2026. Atlassian Rovo: AI-Powered Search, Chat, and Agents. https://www.atlassian.com/software/rovo. Accessed: 2026-05-12.

[3] Atlassian. 2026. Jira: Issue & Project Tracking Software. https://www.atlassian.com/software/jira. Accessed: 2026-05-12.

[4] Atlassian. 2026. Trello: Manage Your Team's Projects. https://trello.com. Accessed: 2026-05-12.

[5] Tobias Hey, Dominik Fuß, Jan Keim, and Anne Koziolek. 2025. Requirements Traceability Link Recovery via Retrieval-Augmented Generation. In *Requirements Engineering: Foundation for Software Quality (REFSQ 2025)*. Springer Nature Switzerland, 381–397. doi:10.1007/978-3-031-88531-0_27

[6] Vaishnavi Kannan, Mujeeb Basit, Puneet Bajaj, Angela Carrington, Imee Donahue, Emily Flahaven, Richard Medford, Tariq Naeem, Luis Padilla, Mauricio Romero, Christoph Lehmann, and Duwayne Willett. 2019. User stories as lightweight requirements for agile clinical decision support development. *Journal of the American Medical Informatics Association* 26, 11 (2019), 1344–1354. doi:10.1093/jamia/ocz123

[7] Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[8] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. 2016. Improving agile requirements: the Quality User Story framework and tool. *Requirements Engineering* 21, 3 (2016), 383–403. doi:10.1007/s00766-016-0250-x

[9] Dipeeka Luitel, Shabnam Hassani, and Mehrdad Sabetzadeh. 2024. Improving requirements completeness: automated assistance through large language models. *Requirements Engineering* 29, 1 (2024), 73–95. doi:10.1007/s00766-024-00416-3

[10] Microsoft. 2026. Azure DevOps Services. https://azure.microsoft.com/en-us/products/devops. Accessed: 2026-05-12.

[11] Modern Requirements. 2026. Copilot4DevOps: AI-Powered Assistant for Azure DevOps. https://copilot4devops.com. Accessed: 2026-05-12.

[12] Kleoson Bruno Correa Santos. 2018. *REACT: Uma Abordagem Ágil de Apoio ao Processo de Desenvolvimento de Requisitos de Software Baseada em Evidências Empíricas.* Master's thesis. Universidade Federal do Pará, Belém, PA, Brasil.

[13] Bleno Wilson Franklin Vale da Silva. 2020. *REACT-M: Uma Evolução da Abordagem REACT para Apoio ao Processo de Gerência de Requisitos de Software Baseado em Evidências Empíricas.* Master's thesis. Universidade Federal do Pará, Belém, PA, Brasil.

[14] Andreas Vogelsang and Jannik Fischbach. 2025. Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline. In *Handbook on Natural Language Processing for Requirements Engineering*. Springer Nature Switzerland, 435–456. doi:10.1007/978-3-031-73143-3_16

[15] Xiangqian Xu, Yajie Dou, Liwei Qian, Jiang Jiang, and Yuejin Tan. 2023. A Requirement Quality Assessment Method Based on User Stories. *Electronics* 12, 10 (2023), 2155. doi:10.3390/electronics12102155

[16] Zheying Zhang, Maruf Rayhan, Tomas Herda, Manuel Goisauf, and Pekka Abrahamsson. 2024. LLM-Based Agents for Automating the Enhancement of User Story Quality: An Early Report. In *Agile Processes in Software Engineering and Extreme Programming*. Springer Nature Switzerland, 117–126. doi:10.1007/978-3-031-61154-4_8