LATER GATOR

A PROJECT PROPOSAL

PREPARED FOR

CEN3031 – Introduction to Software Engineering

FALL 2025 | University of Florida

Professor Lisha Zhou

PREPARED BY

TEAM LATER GATOR:
Danielle Foege
Madison Holt
Carrie Ruble
Samantha Wong

TABLE OF CONTENTS

INTRODUCTION

The purpose of this proposal is to outline our team's plan for successfully developing a project that aligns software engineering best practices to develop a software solution to a current social challenge. Our team brings a range of technical and problem-solving expertise that we are eager to expand by engaging deeply with software engineering methodology. This project presents an opportunity not only to apply our current knowledge but also to strengthen our proficiency with structured development processes that are widely used across modern software teams.

We are particularly focused on engaging with tools and practices that support collaboration, integration, and delivery. These methods are essential for ensuring transparency, consistency, and quality throughout the project life cycle, which in turn promotes confidence and satisfaction among stakeholders. By aligning our workflow with these practices, we can better track progress, anticipate challenges, and deliver reliable results.

Each member of our team will assume a primary role that reflects their strengths while allowing for cross-learning. *Samantha Wong* will focus on **Product Management**, *Madison Holt* on **Scrum Master**, and *Danielle Foege* and *Carrie Ruble* as **Development Team Members**. Together, these roles will ensure a balanced distribution of responsibilities and accountability, while reinforcing our shared commitment to learning and professional growth.

To accomplish the goals of this project, we will define a current social challenge as our "Client's Needs," then we will describe our project vision for addressing this challenge with software. We will also provide a risk management plan and briefly lay out our next steps toward successful project completion.

CLIENT'S NEEDS

The client's need centers on a familiar challenge: the difficulty individuals face in maintaining healthy habits around social media use. What often begins as a quick check of a feed can spiral into extended, unplanned scrolling sessions. People frequently report that they "lose track of time" while browsing, even though the activity itself rarely feels like meaningful engagement. The mismatch between the perception of low effort and the reality of lost time creates frustration and a sense of diminished control.

This behavior contributes directly to unhealthy screen time patterns, where small moments of checking accumulate into hours each day. The problem is not simply the amount of time spent on devices, but the lack of conscious choice in how that time is used. Passive scrolling displaces other priorities, disrupts focus, and can negatively affect mood and well-being.

Previous attempts to address this need have had limited success. Solutions such as screen-time limits, after-use reports, or app timers are typically retrospective, alerting users only after the time has already been lost. Meanwhile, light nudges and reminders often arrive out of context and are easily ignored. Hard lockouts may restrict use temporarily but can feel punitive, leading users to disable the tool or find workarounds. In short, these approaches have not been effective in helping users notice and interrupt the moment when passive use begins.

For the client, the unmet need is clear: a way to help individuals recognize and disrupt the onset of automatic scrolling, transforming passive use into moments of intentional engagement.

PROJECT VISION

FOR students and young adults ***WHO*** struggle with excessive or unconscious screen time, ***Later Gator****, the screen time awareness tool* **is an Android app** ***THAT*** tracks per-app usage and offers real-time, opt-in interventions to exit distracting apps, start a focus timer, and capture quick reflections. ***UNLIKE*** phone-native settings buried in menus or "nag only" apps, ***OUR*** product lets users set limits, take decisive breaks, and build healthier habits with simple prompts and data they control.

RISK MANAGEMENT PLAN

To ensure the successful delivery of our project, it is essential to anticipate and prepare for challenges that may arise during development. By identifying potential risks early and outlining strategies to address them, our team can maintain steady progress, safeguard project scope, and minimize disruptions. The following risk management plan organizes risks into categories relevant to our work and describes both their potential impact and the mitigation strategies we will use to reduce their likelihood or severity.

1. Technology Risk

Example risks:

Permissions Friction (Usage/Accessibility)

- Probability: High (50–75%). Many users hesitate to grant accessibility permissions.
- Effect: Serious. Without permissions, the app cannot function as intended and adoption may stall.
- Mitigation: Guided user onboarding, transparency on local-only data, easily accessible permission status page.

OEM Background Limits/Quirks

- Probability: Moderate (25–50%). OEM background app killing to save battery.
- Effect: Serious. Timers or interventions may fail silently, making the app seem unreliable.
- Mitigation: Minimize background services, use foreground services when needed, document device quirks.

2. People Risk

Example risks:

Team Member Personal Life Interruptions

- Probability: Moderate (25–50%).
- Effect: Tolerable to Serious. Missed deadlines, delayed deliverables, etc..
- Mitigation: Build in buffer time with early internal deadlines, divide responsibilities clearly, maintain open communication channels.

3. Organizational Risk

Example risks:

Scope Creep

- Probability: High (50–75%). Strong temptation to add additional features like gamification and social features.
- Effect: Catastrophic. Jeopardizes stable MVP delivery within timeline.

- Mitigation: Lock MVP scope (usage tracking, limits, prompt, exit, timer, reflection, dashboard), push “stretch features” further into backlog.

4. Tools Risk

Example risks:

Team Member Failed Equipment

- Probability: Low–Moderate (10–25%).
- Effect: Serious. Team member may be blocked from usage.
- Mitigation: Ensure version control (GitHub) is always up to date, maintain backups, use shared documentation, allow alternate equipment if available, utilize paired programming.

5. Requirements Risk

Example risks:

Intrusiveness Concerns

- Probability: Moderate (25–50%). Interventions may be seen as annoying, depending on frequency.
- Effect: Tolerable. May reduce adoption but does not block delivery.
- Mitigation: Ensure interventions remain opt-in, allow users to pause or adjust intervention intensity easily, provide transparent records of the interventions so users understand what happened and why.

6. Estimation Risk

Example risks:

Time Underestimation

- Probability: Moderate (25–50%).
- Effect: Serious. Can cause missed milestones.
- Mitigation: Use Jira for sprint tracking, assign story points conservatively, review scope weekly.

7. Ethical / Data Privacy Risk

Example risks:

User Trust and Data Handling

- Probability: Low (10–20%).
- Effect: Serious. Loss of user adoption, liability.
- Mitigation: Store data locally, make all data visible/exportable, request only necessary permissions, use clear UI language (“your data stays on your device”).

NEXT STEPS

Initial Setup

Our team has completed the foundational setup needed to begin development. A shared GitHub repository has been created (Appendix A) and may be found at https://github.com/smwong36/Later-Gator. All members have been added as collaborators. Branch protection rules have been configured for the main branch to ensure code quality (Appendix B). CircleCI has been integrated as our configuration management tool to support continuous integration and testing (Appendix C), and Jira has been configured as our project management tool to track tasks and progress (Appendix D).

In terms of the development environment, we anticipate using Android Studio as our primary IDE. The front end will primarily be developed in Java and Kotlin, the languages most commonly supported for Android application development. We also anticipate incorporating C++ where performance-critical components or native Android libraries are required, as well as Python for data handling or scripting tasks as needed. Supporting tools include the Android SDK and NDK (for C++), and any required emulators for device testing. This configuration provides the team with a consistent, modern environment aligned with both project requirements and course expectations.

Initial Scope of Work

Looking ahead, the project scope will be guided by upcoming course deliverables. In the short term, this includes creating user stories (Module 5) and developing the product backlog (Module 5), followed by the software architecture models (Module 6). Our first major presentation milestone is the Project Design and Plan Presentation with Sprint Retrospective (Module 7).

Subsequent phases will expand on this foundation, covering software tests (Module 9), iterative sprint presentations and retrospectives (Modules 11 and 12), and culminating in the Final Presentation and Documentation Report (Module 15). By organizing our work around these structured deliverables, we ensure consistent progress and alignment with both academic requirements and project goals.

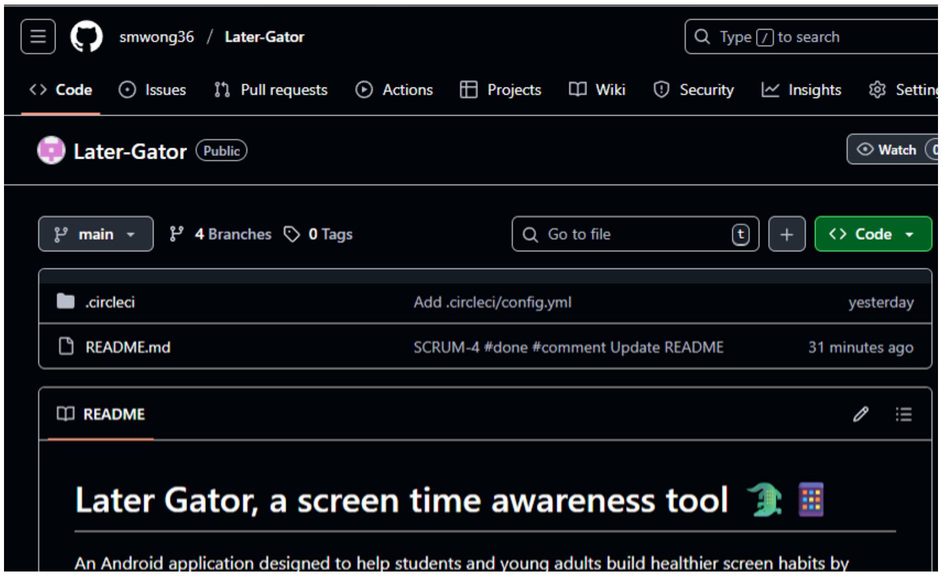
CONCLUSION

The Later Gator project represents both an opportunity to address a pressing social challenge and a chance for our team to strengthen our skills as software engineers. By focusing on the difficulty individuals face in maintaining healthy digital habits, we have defined a clear client need and articulated a vision for a solution that emphasizes user autonomy and intentional engagement. Through structured planning, proactive risk management, and the use of modern development tools, our team is prepared to deliver a reliable and relevant product prototype.

As we move into development, we are committed to following the principles of transparency, collaboration, and accountability that underlie effective software engineering practice. With a well-defined scope and a solid technical foundation already in place, we are confident in our ability to meet project milestones and deliver a product that is both functional and impactful. Ultimately, by maintaining clear processes and user-focused design, we aim to ensure stakeholder satisfaction through consistent progress, trustworthy outcomes, and a solution that meaningfully addresses the problem at hand.

APPENDIX

A. GitHub Repo Created



.circleci	Add .circleci/config.yml	yesterday
README.md	SCRUM-4 #done #comment Update README	31 minutes ago

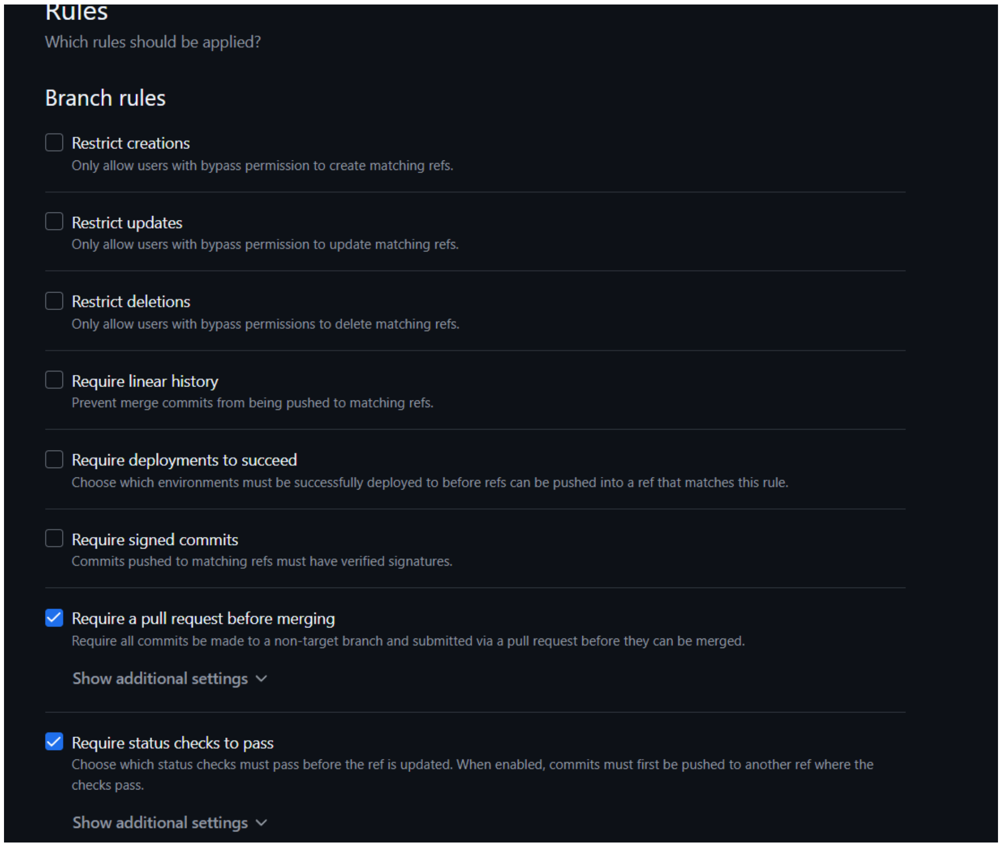
README

Later Gator, a screen time awareness tool 🐊 📱

An Android application designed to help students and young adults build healthier screen habits by

Figure A: GitHub repository “Later-Gator” home page with collaborators added.

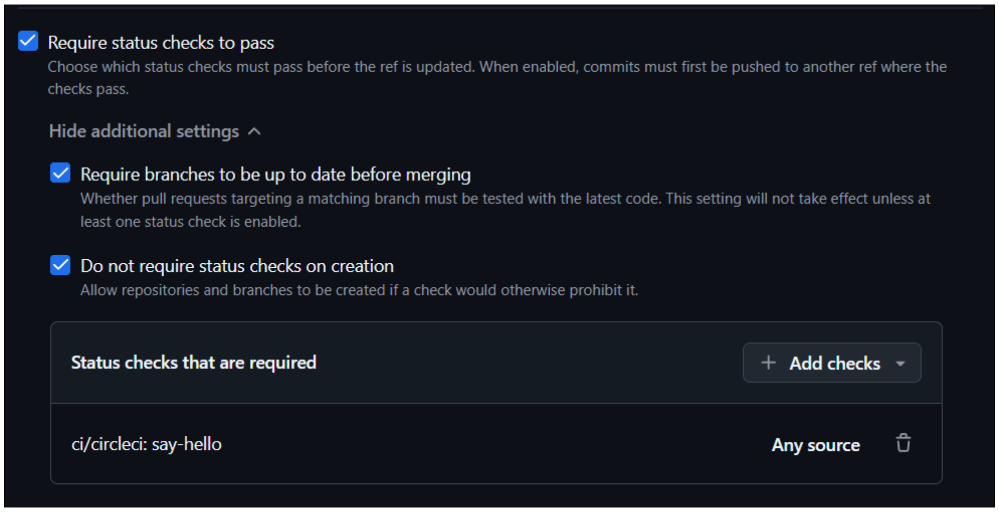
B. Branch Protection Rules


Rules
Which rules should be applied?

Branch rules

- [ ] Restrict creations
  Only allow users with bypass permission to create matching refs.
- [ ] Restrict updates
  Only allow users with bypass permission to update matching refs.
- [ ] Restrict deletions
  Only allow users with bypass permissions to delete matching refs.
- [ ] Require linear history
  Prevent merge commits from being pushed to matching refs.
- [ ] Require deployments to succeed
  Choose which environments must be successfully deployed to before refs can be pushed into a ref that matches this rule.
- [ ] Require signed commits
  Commits pushed to matching refs must have verified signatures.
- [x] Require a pull request before merging
  Require all commits be made to a non-target branch and submitted via a pull request before they can be merged.
  Show additional settings
- [x] Require status checks to pass
  Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass.
  Show additional settings

Figure B1: GitHub main branch rules screenshot.



☑ Require status checks to pass
Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass.

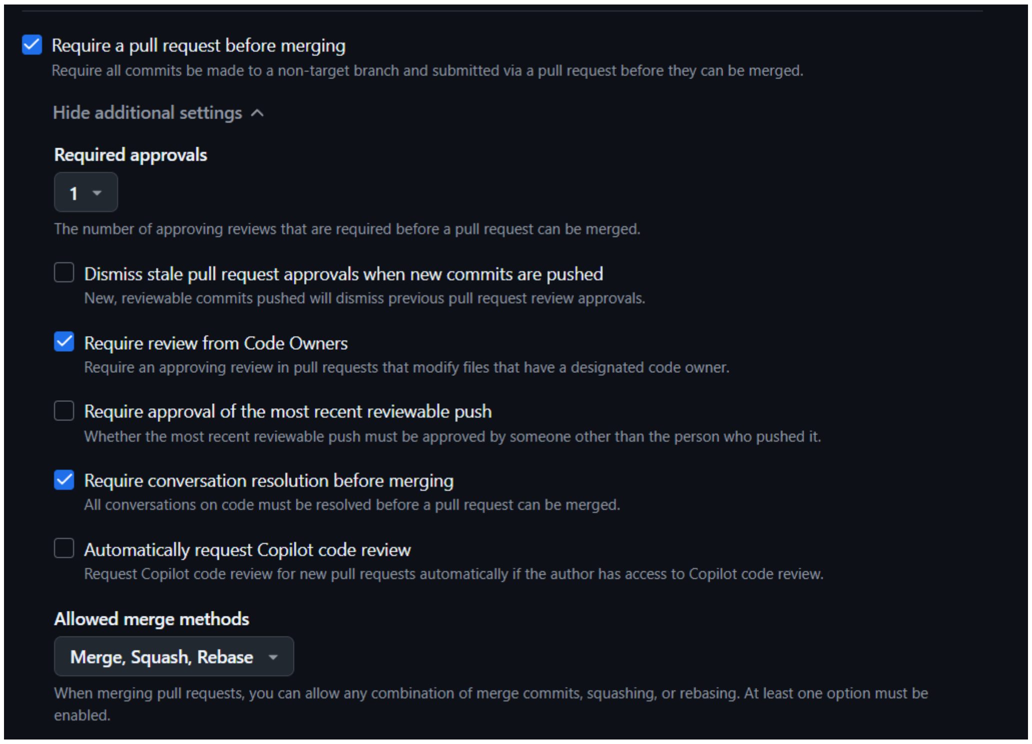
Hide additional settings

☑ Require branches to be up to date before merging
Whether pull requests targeting a matching branch must be tested with the latest code. This setting will not take effect unless at least one status check is enabled.

☑ Do not require status checks on creation
Allow repositories and branches to be created if a check would otherwise prohibit it.

Status checks that are required	Add checks
ci/circleci: say-hello	Any source

Figure B2: GitHub main branch rules screenshot.



☑ Require a pull request before merging
Require all commits be made to a non-target branch and submitted via a pull request before they can be merged.

Hide additional settings

Required approvals
1
The number of approving reviews that are required before a pull request can be merged.

☐ Dismiss stale pull request approvals when new commits are pushed
New, reviewable commits pushed will dismiss previous pull request review approvals.

☑ Require review from Code Owners
Require an approving review in pull requests that modify files that have a designated code owner.

☐ Require approval of the most recent reviewable push
Whether the most recent reviewable push must be approved by someone other than the person who pushed it.

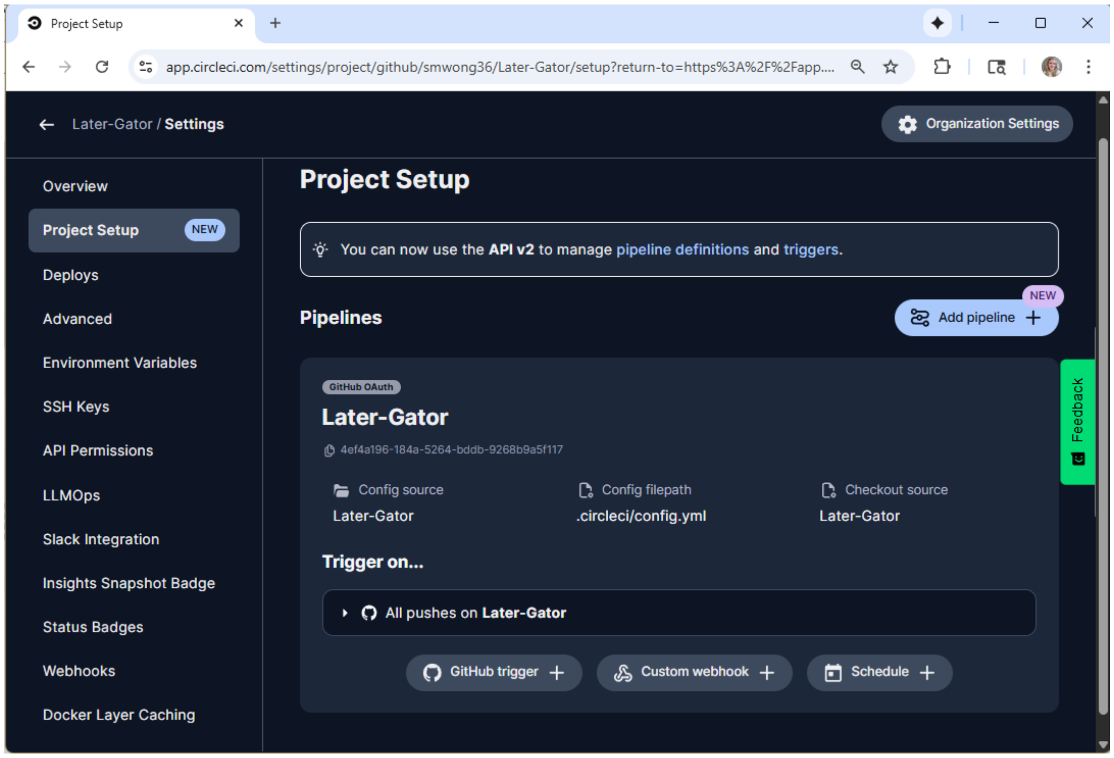
☑ Require conversation resolution before merging
All conversations on code must be resolved before a pull request can be merged.

☐ Automatically request Copilot code review
Request Copilot code review for new pull requests automatically if the author has access to Copilot code review.

Allowed merge methods
Merge, Squash, Rebase
When merging pull requests, you can allow any combination of merge commits, squashing, or rebasing. At least one option must be enabled.

Figure B3: GitHub main branch rules screenshot.

C. Circle CI Integration



Project Setup

You can now use the API v2 to manage pipeline definitions and triggers.

Pipelines

Add pipeline

GitHub OAuth

Later-Gator

4ef4a196-184a-5264-bddb-9268b9a5f117

Config source	Config filepath	Checkout source
Later-Gator	.circleci/config.yml	Later-Gator

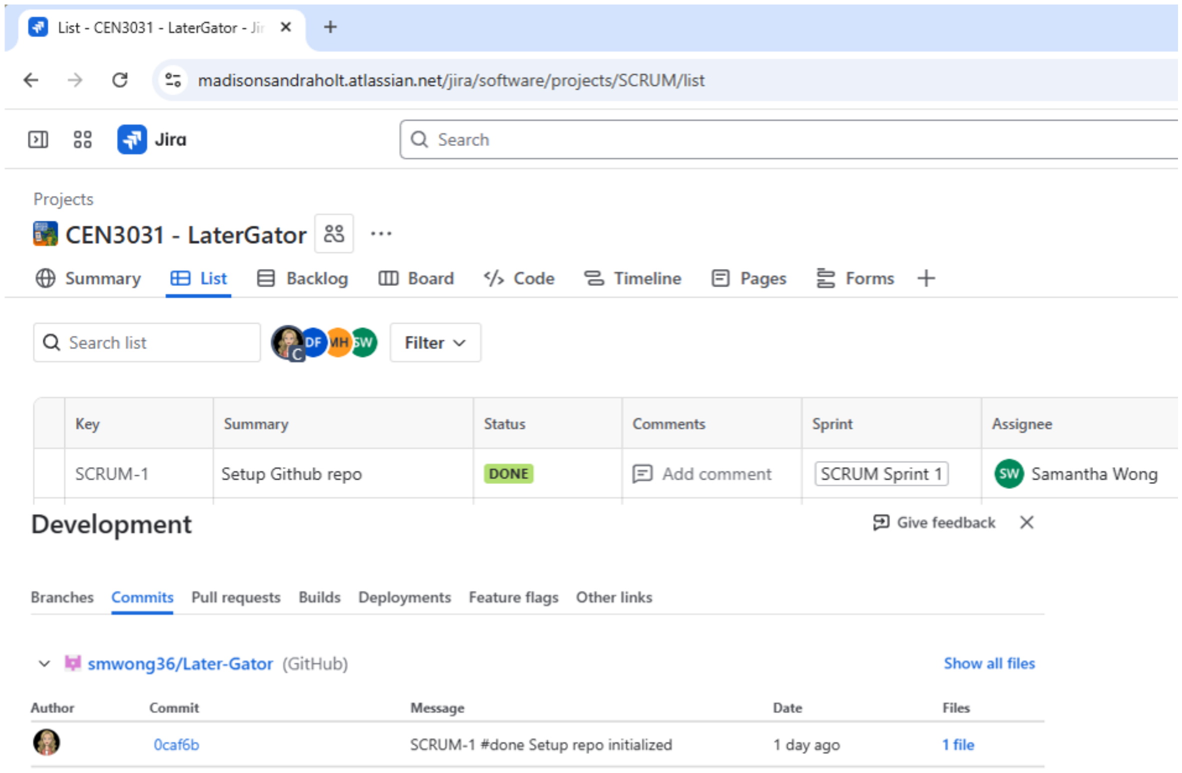
Trigger on...

All pushes on **Later-Gator**

GitHub trigger + | Custom webhook + | Schedule +

Figure C: CircleCI dashboard confirming successful pipeline runs for commits.

D. Jira Integration



Key	Summary	Status	Comments	Sprint	Assignee
SCRUM-1	Setup Github repo	DONE	Add comment	SCRUM Sprint 1	Samantha Wong

Development

Give feedback

Branches | Commits | Pull requests | Builds | Deployments | Feature flags | Other links

smwong36/Later-Gator (GitHub) — Show all files

Author	Commit	Message	Date	Files
	0caf6b	SCRUM-1 #done Setup repo initialized	1 day ago	1 file

Figure D: Jira issue (SCRUM-1) showing GitHub commit linked in the Development panel.