SOFTWEAR ENGINEERING

CodePulse: Role-Based Work Tracking with Consent-Based Activity Monitoring for Software Teams

C. Akanksha[1], N M. Ashitha[1], Ayush Singh[1], A. Prakash[2]

[1]Department of Computer Science and Engineering, Sir M Visvesvaraya Institute of Technology, Bangalore, India
[2]Guide, Department of Computer Science and Engineering, Sir M Visvesvaraya Institute of Technology, Bangalore, India

Published November 2026

Abstract

Our experience with most software teams we have worked with is that they track work in 4-5 different places: an attendance sheet, a task board, a chat channel for status updates and a code repository. No one has a full picture unless they piece it together manually. CodePulse is our attempt to provide that picture in one place. It is a Flask + Firestore web application for 3 types of users (Employees, Team Leads, Managers) with work sessions, tasks, daily updates, timesheet corrections and public GitHub activity.

Server-side access control with team-level scoping is implemented. For any sensitive change, a single decision is recorded from a Manager. GitHub integration is via OAuth with state + PKCE, but no repository permissions are requested. An employee's activity is only shown to supervisors if they opt-in to share. We also implemented an optional, and admittedly more controversial, Smart Activity module. It is a Windows agent that tracks the type of application running in the foreground. It also takes periodic screenshots. It is only active during a work session and only after the employee provides versioned consent, which can be withdrawn at any time. This paper covers design, data model, security/privacy controls and implementation. The code has 183 automated tests running against an in-memory store. When we deliberately broke the permission code, the tests caught it every time. We haven't yet run CodePulse against live Firestore, a registered GitHub app, a real Windows machine. And we haven't done a user study. So, we're careful about what we claim. Our primary argument is that activity data should be a conversation starter, not a productivity score.

Keywords: work tracking; role-based access control; Flask; Firestore; GitHub OAuth; employee monitoring; privacy by design; timesheet corrections

1 Introduction

Ask a Team Lead in a small software company how things are going with their team, and you will probably get several tabs opened. Attendance sits in one place, tasks in another, daily updates in a chat thread and code activity somewhere else again, and there's no single record of who made what decision.

CodePulse brings these together in a single web application. It is based on three roles. Employees can start/end work sessions, keep tasks up to date, post daily update, flag blockers. Team Leads see their own team (only). They can create tasks, clear blockers, escalate timesheet issues up the chain. Managers manage users, teams, settings. They make the final decision on timesheet corrections. They can view the audit log. There are also two optional integrations on top of the core. One shows an

employee's public commits, pull requests on GitHub. The other is Smart Activity. It tracks what type of application is being used during a work session.

That second integration is the one we were most concerned about. Employee monitoring software has a reputation for eroding trust. And, it's easy for the numbers to end up being used to judge people. [1]. Developer productivity research is pretty consistent that software work can't be reduced to simple counts. [2–4]. So, we set three rules for ourselves when building CodePulse.

1. The server decides. Every permission check happens on the server. Hiding a button in the interface is never treated as protection.
2. No monitoring without consent. No quiet monitoring. It is off until it is on. Explicit consent to a specific version of the notice is required, and it can be withdrawn at any time.
3. No scores. Activity is shown as background for a conversation. CodePulse never turns it into a productivity number.

1.1 Contributions

The paper makes five contributions:

1. Role-based design for work tracking. Every request is checked against the user's team. For records outside that scope, return "not found" (not forbidden). That way we don't leak identifiers. 2. Timesheet correction workflow. No overwriting of original record. Only one decision per request. 3. GitHub integration. No repository permissions requested. Encrypt access token. Only share activity with supervisors if employee opts-in. 4. Consent-based Smart Activity module. Server re-classifies whatever the agent sends. Store screenshots privately. Audit access to screenshots. Limit retention. 5. Be honest about what we've been able to verify and what we haven't.

1.2 Scope of this paper

This is a systems paper. We cover design, implementation, automated tests. We haven't run a field study. So, we're not making any claims about usability, productivity, employee acceptance, etc. That's for future work (Section 8).

2 Related work

2.1 Developer work measurement

Forsgren et al.'s SPACE framework looks at developer productivity along five dimensions: satisfaction/well-being, performance, activity, communication/collaboration, efficiency/flow. [2]. Their key warning is that activity measures (e.g., commit counts) only capture one of those. And, they become misleading when used alone or to judge individuals. Meyer et al. came to similar conclusions on the developer side. What makes a day feel productive is mostly about meeting goals and not being interrupted. It's not about how much you produced. [3]. The book edited by Sadowski, Zimmermann goes into detail on this. [4]. In another direction, Mäntylä et al. mined about 700k Jira issues for emotional signals (valence, arousal, dominance). They found only weak connections to issue priority, type, resolution time. They didn't have ground truth to check burnout. [5]. So, taken together, this suggests being cautious with repository/tracker data. That's the approach CodePulse takes. Show what was recorded. Don't go so far as to rank anyone.

2.2 Monitoring tools and self-monitoring

On the commercial side, tools like Hubstaff, [6]. Time Doctor, [7]. combine time tracking with screenshots, app/website usage. And, they usually boil this down to activity levels or productivity ratings for managers to look at. On the other hand, WakaTime, [8]. logs coding time via editor plugins. It mainly shows it to the developer. Meyer et al. looked at self-monitoring among software developers. They argued that these tools should be first and foremost for the developer's own reflection. [9]. CodePulse is somewhere in the middle. Supervisors can see some activity context. But, only after the employee has consented to a specific notice. Only while there is a visible indicator on screen. Tight limits on who can see screenshots. No score.

2.3 Authorization and delegated access

Broken access control has topped the OWASP Top 10 in both the 2021 and 2025 editions [10], which is a good reason to take it seriously in a system that holds data about people. CodePulse uses a role-based access control (RBAC) model as described by Sandhu et al. [11]. CodePulse queries the database for each request to determine the user's role and checks the team scope of each record. CodePulse uses GitHub's OAuth 2.0 authorization code flow [12] with PKCE [13] to tie the response from GitHub to the client that initiated the request. PKCE was originally designed for use by public clients; however, current recommendations advise the use of PKCE with server-side applications such as CodePulse [14].

2.4 Employee monitoring and privacy law

Legal and ethical issues are raised by monitoring software, whereas code metric dashboards do not raise these types of concerns. Processing personal data under the GDPR requires a lawful basis and must adhere to the principles of purpose limitation, data minimization, and storage limitation. Additionally, Article 25 of the GDPR requires data protection by design and by default [15]. India's Digital Personal Data Protection Act, 2023 is built around consent and limits processing to the stated purpose [16]. While the Rules for the Digital Personal Data Protection Act were notified on November 13, 2025, most of the substantive obligations do not take effect until 18 months later [17]. In addition, the Supreme Court of India has recognized privacy as a fundamental right [18]. There is one point specifically relevant to CodePulse: the Article 29 Working Party concluded that consent is rarely a valid legal basis in a work environment, because employees are rarely in a position to freely refuse [19].

We addressed that point. CodePulse treats consent as a safeguard and does not consider it to make processing lawful. Monitoring is off by default, and consent can be withdrawn at any time by the employee. CodePulse records the version of the notice accepted by the employee, and deletes data after a set period of time. These decisions are based on the principles of privacy-by-design [20], and reduce risk; however, none of these measures are substitutes for legal counsel prior to a real deployment.

2.5 Positioning

Table 1 sets CodePulse alongside the approaches discussed above.

Table 1. Positioning of CodePulse against related approaches.

Aspect	Commercial time trackers [6, 7]	Developer self-monitoring [8, 9]	Text and sentiment mining [5]	CodePulse
Main data	Time, app and site usage, screenshots	Editor or app usage	Issue tracker text	Work sessions, tasks, updates; optional GitHub activity and app category
Primary audience	Managers	The developer	Researchers	Employees, Team Leads, Managers
Access control	Organization roles	Single user (typically)	Not applicable	Three roles with team scope, server-enforced
Consent for monitoring	Employer set	Self-chosen	Not applicable	Versioned, revocable, visible indicator
Output	Activity levels or ratings	Personal dashboards	Statistical signals	Context for review; no score

3 System design

3.1 Architecture

CodePulse is a fairly standard server-rendered web application (Figure 1). Browser communicates with Flask app built with the application-factory pattern. Each area of functionality is separated into its own blueprint: authentication, employee, team lead, manager, GitHub, Smart Activity, agent API and legacy pages. Pages are Jinja2 templates using Bootstrap 5, and one CanvasJS chart. One of the decisions that made sense was creating all data access behind a small store layer. This store layer has two implementations with identical interfaces: FirestoreStore for production, and MemoryStore for testing. Thus, we were able to test all permission logic without needing cloud credentials. The store provides two operations that the remainder of the code depends on. Atomic create fails if the document already exists; we utilize this to maintain unique usernames and prevent users from having multiple active sessions.

Compare-and-set update ensures that a request can only be approved or rejected once. Smart Activity agent is an independent Python program running on the employee's Windows machine.

The Smart Activity agent communicates with three JSON endpoints located under /api/activity/agent/ and identifies itself using a bearer token instead of a browser session (Section 4.4).

3.2 Roles and scope

CodePulse stores a user's role on their user record and reads it fresh from the database for each request. Therefore, changing a user's role or disabling their account will have immediate effect rather than waiting until their session expires. The session cookie itself contains only the user ID and the CSRF token. Table 2 outlines the capabilities of each role.

Table 2. Capabilities by role. "Own team" indicates the team the Team Lead is assigned to manage.

Capability	Employee	Team Lead	Manager
Start or end own work session	Yes	Yes	Yes
Update status of own assigned tasks	Yes	Yes	Yes

Capability	Employee	Team Lead	Manager
Post daily update and raise blocker	Yes	Yes	Yes
Request correction to own timesheet	Yes	Yes	Yes
Create or edit tasks	No	Own team	Any team
View sessions and reports	Own only	Own team	All
Resolve blockers	No	Own team	Any team
Escalate blocker or correction request	No	Own team	Not needed
Approve or reject correction request	No	No	Yes
Manage users and teams	No	No	Yes
Change global settings	No	No	Yes
Connect own GitHub account	Yes	Yes	Yes
View another user's GitHub activity	No	If shared by user	If shared by user
View Smart Activity screenshots	Own only	Only if permitted by a Manager	Users they manage

A Team Lead who does not yet have a team sees no users. Managers can additionally see users that do not belong to any team.

3.3 Data model

Table 3. Main logical collections. Names are descriptive; see the source code for exact identifiers.

Collection	Document key	Key fields and relationships
Users	User ID (unique username enforced by atomic create)	Username, password hash, role, team ID, disabled flag, lockout counters
Teams	Team ID	Name, Team Lead user ID, settings
Active sessions	User ID	Start time; at most one per user
Completed sessions	Session ID	User ID, start, end, duration (s); never edited after close
Correction requests	Request ID	User ID, original session ID (if any), requested start/end, reason, status, escalated-by, decided-by
Corrected sessions	Session ID	Created on approval; links to original session and request
Tasks	Task ID	Team ID, assignee, title, description, project, priority, due date, status
Daily updates	Update ID	User ID, date, text
Blockers	Blocker ID	Source update ID, user ID, team ID, status, resolved-by
GitHub connections	User ID	GitHub account ID, encrypted token, sharing flags
Smart Activity	Per user	Consent records (notice version, time), agent token hashes, activity segments, screenshot metadata
Audit log	Event ID (append only)	Actor, action, target, timestamp

Collection	Document key	Key fields and relationships
Contact messages	Message ID	Messages from public contact form

4 Implementation

4.1 Work sessions and timesheets

When an employee clicks "Start Work", the server atomically creates an active-session document keyed by the user's ID. If one already exists, the create fails; this prevents an employee from starting two sessions in two browsers simultaneously. When an employee clicks "End Work", the server reads the active session, writes a completed session with its end time and duration, and deletes the active document. Only the owner of a session can end it; Team Leads and Managers cannot end a session on behalf of another person.

Employees sometimes forget to clock out or clock out and inadvertently start again; thus, employees can request a correction. The server will only accept a correction request if the reason is at least 5 characters long, the corrected session is no longer than 16 hours, and the requested times are in the past according to the server's clock.

A Team Lead can escalate a correction request; however, ultimately only a Manager can make a decision on a correction request. The decision is saved with a compare-and-set update; thus, even if two Managers click at exactly the same time, only one decision will stick (Figure 2). Approving a correction never overwrites anything; if the correction is to an existing session, the server adds a corrected session that points back to both the original session and the correction request, and reports will use the corrected duration from that point forward; if the correction is for a session that was never recorded, the server adds a new corrected session linked to the correction request; either way, the original record remains intact should anyone wish to view it.

4.2 Tasks, updates and blockers

Team Leads (for their own team) and Managers can create task cards (title, description, project, priority, due date) and assign them to employees. Employees can move their tasks to do, in progress and blocked. Tasks that are cancelled cannot be worked on by the employee. Every day, employees can post a short update of what they are doing. If the update contains a blocker, the server creates a blocker record for the employee’s Team Lead. The Team Lead can then either resolve the issue or escalate it to the Manager. GitHub integration: 4.3. An employee can connect their account from the GitHub integration page (Figure 3). The server will generate a random state value and a PKCE code verifier and save them in the session for up to 10 minutes. The server will redirect the user to GitHub with an empty scope. When the user returns from GitHub, the server will check if the state value is correct and hasn’t been used before. It will then exchange the authorization code and verifier for an access token. The server will then check if the GitHub account is already linked to another CodePulse user. The access token is then encrypted with Fernet and saved. The token is never seen by any user, including the employee. The server will fetch up to 30 recent commits and 30 recent pull requests and cache the result for 60 seconds.

4.3 GitHub integration

We only request public activity from GitHub (no repository scope requested). We do not link tasks to commits/pull requests yet. Sharing is off by default. An employee can choose to share their GitHub activity with their Team Lead and Manager. If a supervisor hasn't seen any activity from employees under their responsibility, then employees are not sharing their GitHub activity. If an employee disconnects, then the server will delete the stored access token and ask GitHub to revoke it. If the token is no longer valid for any other reason (e.g. it was revoked on GitHub's side), then the connection is invalid and the employee will be asked to connect again.

4.4 Smart Activity

The Smart Activity module is the most sensitive part of the system and will be described in more detail in Section

How to turn on the Smart Activity module. Smart Activity is off by default. A Manager must enable it for the whole organization. Even after that, nothing will happen for a given employee until he or she accepts the current version of the notice. After that, he or she can generate a token for the Windows agent. This token will replace the old one. If the employee withdraws consent on the Smart Activity settings page, then the token is immediately revoked and the server stops accepting anything from the agent. A Manager can ask employees to consent again when the notice changes.

What the agent does. The agent only runs when the employee is in a work session. It stops recording after 5 minutes of no keyboard or mouse activity. It checks the foreground application every 2 seconds. Back-to-back samples of the same application are merged if the gap between them is less than 15 seconds. These merged samples are then grouped into 300-second activity units and uploaded. When monitoring is active, a red bar is shown on the employee's screen and a badge is shown in the web interface.

What leaves the machine. For each activity segment, the agent sends the name of the application and, for supported browsers, the domain of the site. It does not send the full window titles. For YouTube, the agent runs a keyword check on the video title locally and only sends back whether the video looks learning-related or not. Windows that match the sensitive-window rules (such as password managers or private browsing windows) are logged as "private" and are never shown in screenshots.

How the server classifies activity. We do not trust the agent to label its own data. The server classifies activity using rules that can be edited by a Manager. Development tools are classified as productive. Social media, streaming and games are classified as potentially distracting. All other activity is neutral. YouTube is classified as potentially distracting unless the agent reported the video as learning-related. A distraction event is only recorded once the employee has spent at least 120 seconds on a potentially distracting site. One continuous visit to a potentially distracting site will result in at most one distraction event.

Checking incoming activity segments. The server discards activity segments that are outside the employee's work session, overlap with already received activity segments, are in the future or are longer than 3,600 seconds. A batch with more than 200 activity segments is rejected as a whole.

Screenshots. If screenshots are enabled, the agent takes a screenshot every 10 minutes. It also takes a screenshot the first time a potentially distracting site is opened in a session. Each image is scaled down to at most 1,280 pixels wide and compressed to 800 KB or less before being uploaded. Screenshots are stored in a private, per-user location and can only be fetched through an authorized route.

Who sees what and for how long. Employees can see all activity recorded about themselves. Managers can see reports and screenshots for the people they manage. Team Leads can see reports for their team but can only see screenshots if a Manager explicitly allowed it (this is not the default). Every time someone other than the owner opens a screenshot, this is written to the audit log. Screenshots are deleted after 30 days. Activity segments are deleted after 90 days. This can be changed by a Manager. The deletion is done by a purge script that must be run by an operator (see Section 7.2).

Agent API. The agent uses three endpoints: one to check its token and fetch current settings, one to send activity batches and one to upload screenshots. Every request must carry the bearer token, which is then checked against the stored SHA-256 hash and against the employee's current consent.

5 Supporting tools

A handful of operator scripts take care of the jobs that are not appropriate for the web application: backing up data to JSON; migrating data from the previous version of the application; re-hashing legacy passwords; assigning existing users roles; creating the first Manager account; and purging expired activity data. Every migration script has a dry-run mode, which shows what would change without actually making the change.

6 Security, privacy and ethics

6.1 Threat model

We had three types of adversaries in mind: • A supervisor who is more curious than they should be (e.g., Team Lead trying to view a person outside of their team). Server-side role and scope checks, 404 responses for out-of-scope records, and audited screenshot access help with this. • Employee trying to game the system (e.g., faking time records, tampering with agent data). Server-side validation and re-classification help with this, but the agent is trusted and can be modified (Section 7.2). • Outside attacker trying to get at accounts/tokens. This is where password hashing, login lockout, CSRF protection, security headers and encrypted GitHub tokens come in. We didn't attempt to protect against a compromised server, or malicious operator with direct access to the database. 5.2 Security controls Table 4 lists controls that are currently in place.

Table 4. Security controls.

Area	Control
Passwords	Hashed with scrypt [21] via Werkzeug. Minimum 10 characters, mixed types, must be different from the username. Legacy passwords are re-hashed, and users are encouraged to change their passwords.
Sessions	Signed cookie containing only user ID and CSRF token. HttpOnly. SameSite=Lax. Expires after 8 hours. The Secure flag is configurable, and should be set to true if the application is served over HTTPS. Accounts are locked after 5 failed login attempts.
Authorization	Role is re-read from the database on each request. Server checks role and team scope. Records outside the user's scope return 404 Not Found.
CSRF	Required on each state changing request from the browser. The agent API is exempt since it uses bearer tokens instead of cookies, so it is not vulnerable to cross-site request forgery.

Area	Control
Output and headers	Jinja2 auto-escaping; X-Content-Type-Options: nosniff; frame denial; same-origin referrer policy; no-store caching. Error pages never show exception text.
GitHub	Empty OAuth scope. Single use state, max 10 minutes. PKCE. Token is encrypted at rest with Fernet and never shown to users.
Agent token	Random value with a fixed prefix, shown once, only stored as SHA-256 hash. Revocable.
Secrets	Flask secret key and Fernet key are provided to the server via its deployment configuration. No key rotation automated yet.
Audit events	Append-only log of logins, failed logins, role changes, settings changes, approvals, consent, and screenshot views by anyone other than the owner.

6.2 Smart Activity privacy by design

This is where the privacy measures for Smart Activity are collected together:

- Consent first. It is off by default. Nothing is collected until a Manager enables the module and the employee accepts a specific version of the notice. Consent can be revoked at any time.
- Always visible. There is a badge in the web application and a red bar on the screen that indicates monitoring is on.
- Time limited. It only captures during an active work session. It stops recording after 5 minutes of no input.
- Minimal data. Only application names, site domains and categories leave the machine. Full window titles are kept local. Sensitive windows are never captured in screenshots.
- Logged access. Screenshots can be reached only by an authorised route. It is logged when someone other than the owner views the screenshots.
- Short retention. Screenshots are removed by the purge script, which enforces the retention limit.
- No score. The interface displays time per category and flagged events. It does not roll these up into a rating of the person.

6.3 Ethical and legal note

We understand that consent at work is never fully free. Employees may feel pressure to agree. They need to publish a notice that describes the purpose and retention period. They need to get legal advice. None of this paper is legal advice. 6. Evaluation Currently we are only doing automated testing. 183 tests pass with pytest in ~12 seconds.

7 Testing

We haven't done any user study. So we are not making any claims about usability/acceptance.

7.1 Method

The tests are running the full Flask application on top of the in-memory store. So we don't need a cloud account.

GitHub tests are using canned responses. They never hit the network. Agent tests are replacing Windows system calls with fakes. Table 5 lists 7 test files and what they cover.

Table 5. Test files and what they check.

Test file	What is checked
test_auth.py	Sign-up, login, lockout, forced password change, CSRF rejection, logout, security headers.
test_scope.py	Role and team scope on all role pages. 404 for out-of-scope IDs. Team Lead with no team sees nobody.
test_features.py	Work sessions, tasks, updates, blockers, timesheet rules, escalation, approval, reports, user and team administration.
test_github.py	State and PKCE handling, token encryption, sharing rules, error handling, revocation (mocked GitHub).
test_smart_activity.py	Consent, token handling, category rules, YouTube heuristic, ingest validation, distraction threshold, screenshot limits, access rules, retention.
test_activity_agent.py	Sampling, idle handling, merging, batching, sensitive-window rules (with fakes).
test_migration.py	Apply is idempotent. Password re-hash. Conflict handling. Dry run doesn't change anything.

7.2 Testing the tests

If a test suite cannot fail, it does not tell you much when it passes. We purposely broke permission code four times while building the role system. Each time we had at least one test fail. We view this as a small manual mutation check, not proper mutation testing [22], but it gave us some confidence that we are actually testing the rules.

7.3 What we have not verified

There are a number of things we have not tested.

- How the system behaves on live Firestore, including transactions and query limits.
- A registered GitHub OAuth application with actual tokens and real rate limits.
- The Windows agent on a real computer: detecting the foreground window, idle time, capturing screenshots and the on-screen bar.
- Many users at once, and the interface in different browsers.
- Any user feedback, usability measurement or evidence of employee acceptance.

These are at the top of our list for future work. A green test run means we're confident the rules we wrote are working as intended in the test environment. It doesn't mean the system is bug free.

8 Discussion and limitations

8.1 Discussion

and Limitations Discussion One theme that runs through the design is that the server is the authority. Every sensitive step is a single decision. It's recorded. A Team Lead can escalate a correction, but they can't approve it. An approved correction adds a linked record. It doesn't edit the old one. If you ask for something outside your scope, you get "not found". Having an in-memory twin of the store is what made it realistic to test all of this. That's where we've put most of our verification effort.

Smart Activity is where monitoring systems' tensions manifest themselves the most. The potentially distracting label is just an educated guess based on an application name or site domain. The YouTube

check is just a keyword based rule that will occasionally return false positives. A developer reading documentation in a browser or watching a conference talk, for example, could be flagged as potentially distracting even though they are doing exactly what they should be. That is why the Smart Activity view shows the time per category and flagged events as context and we do not claim to measure the developer's effort or output. This is very much in line with the SPACE framework's warnings on [2].

8.2 Limitations

Table 6 lists the known limitations of the current implementation of CodePulse and what they mean in practice.

Table 6. Known limitations and their effects.

Limitation	Effect
Not run on live Firestore, GitHub or Windows	Real-world behaviour is unconfirmed (Section 6.3).
No user study	Usability and acceptance are unknown.
Login throttling is held in process memory	Resets on restart and is not shared across server processes.
Session times use server local time without a time zone	Wrong results if the server and users are in different time zones.
The agent is a trusted, Windows-only client	A technical user can alter or bypass it; there is no tamper protection.
Purge script is not scheduled	Retention limits apply only when an operator runs the script.
No overlap check for corrected sessions	A correction can overlap another session.
End Work is not a single transaction	A failure between steps can leave a stuck active session until an operator fixes it.
Saving a daily update twice duplicates its blocker	Duplicate blockers.
Managers are peers with no deeper hierarchy	All Managers see all unassigned employees.
Viewing activity reports is not audited	Only screenshot views are logged, although report views are also sensitive.
No two-factor sign-in or e-mail password reset; users and teams can be disabled but not deleted	Password resets go through a Manager.
GitHub: public data only, 30 items, no review or merge status, no task links	Work in private repositories is invisible.
Contact messages have no viewing page; e-mail is not sent	Messages can be read only in the Firestore console.
Most dashboards read the entire collection for the particular view	With large data they are slow. Only the audit log view is paginated.
No Firestore security rules file	Protection relies on the server being the only client; rules should be added.
Agent sample-interval setting is stored but not used	The agent always samples every 2 seconds.

9 Conclusion and future work

CodePulse is a Flask and Firestore application that pulls work sessions, tasks, daily updates, timesheet corrections, GitHub activity and optional consent-based monitoring into one role-based system. We think its strongest points are that role and team scope are enforced on the server; that approvals are single decisions which never overwrite the original records; that the GitHub integration needs no repository permissions and keeps tokens encrypted; and that the monitoring module is off by default, gated on consent, visible, limited to work sessions and subject to retention limits. The system passes all 183 of its automated tests. Still, it has not been run against live services or tried with real users, so how it behaves in practice is an open question.

9.1 Future work

In priority order:

1. Run the system against a real Firestore project (starting with the Firestore emulator), a registered GitHub OAuth application and a real Windows computer, and fix whatever breaks. Add concurrency tests, for example two Start Work requests arriving together, or the same request being approved twice. 13 2. Run a small user study with Employees, Team Leads and Managers, looking at task success, errors and opinions, and in particular at how employees feel about the monitoring. 3. Add Firestore security rules, shared login throttling, time-zone-aware timestamps, a scheduled retention job and audit logging when activity reports are viewed. 4. Check corrected sessions for overlaps, prevent duplicate blockers and add a way to recover stuck sessions. 5. Add two-factor sign-in, e-mail password reset and a way for employees to export their own data. 6. Extend the GitHub integration to private repositories (with explicit consent), review status and links between tasks and pull requests. 7. Improve the activity classifier by comparing it with samples labelled by employees and measuring how often it gets things wrong; report code coverage and run systematic mutation testing. Code availability [Repository URL, licence and version to be added.] CodePulse is built with Flask, Jinja2, Werkzeug, Bootstrap 5, CanvasJS, Google Cloud Firestore (firebase-admin), the Python cryptography library (Fernet) and pytest. References [1] K. Ball, "Workplace surveillance: An overview," Labor History, vol. 51, no. 1, pp. 87–106, 2010.

Code availability

CodePulse is built with Flask, Jinja2, Werkzeug, Bootstrap 5, CanvasJS, Google Cloud Firestore (firebase-admin), the Python cryptography library (Fernet) and pytest.

References

[1] K. Ball, "Workplace surveillance: An overview," *Labor History*, vol. 51, no. 1, pp. 87–106, 2010.

[2] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, "The SPACE of developer productivity: There's more to it than you think," *ACM Queue*, vol. 19, no. 1, 2021, https://doi.org/10.1145/3454122.3454124.

[3] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proc*. 22nd ACM SIGSOFT Int. Symp. Foundations of Software Engineering (FSE 2014), 2014, pp. 19–29.

[4] C. *Sadowski and T. Zimmermann, Eds., Rethinking Productivity in Software Engineering*. Berkeley, CA, USA: Apress, 2019.

[5] M. Mäntylä, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu, "Mining valence, arousal, and dominance: Possibilities for detecting burnout and productivity?," in *Proc*. 13th Int. Conf. Mining Software Repositories (MSR '16), Austin, TX, USA, 2016, arXiv:1603.04287.

[6] Hubstaff. [Online]. Available: https://hubstaff.com.

[7] Time Doctor. [Online]. Available: https://www.timedoctor.com.

[8] WakaTime. [Online]. Available: https://wakatime.com.

[9] A. N. Meyer, G. C. Murphy, T. Zimmermann, and T. Fritz, "Design recommendations for self-monitoring in the workplace: Studies in software development," *Proc*. ACM Hum.-Comput. Interact., vol. 1, no. CSCW, 2017.

[10] OWASP Foundation, "A01:2025 Broken Access Control," *OWASP Top* 10:2025. [Online]. Available: https://owasp.org/Top10/2025/A01_2025-Broken_Access_Control/.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[12] D. Hardt, Ed., "The OAuth 2.0 Authorization Framework," *IETF RFC* 6749, Oct. 2012.

[13] N. Sakimura, J. Bradley, and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients," *IETF RFC* 7636 (Sept 2015).

[14] T. Lodderstedt, J. Bradley, A. Labunets and D. Fett "Best Current Practice for OAuth 2.0 Security," *IETF RFC* 9700 (Jan 2025).

[15] European Parliament and Council, Regulation (EU) 2016/679 (General Data Protection).