An Automated Time and Activity Tracking System for VFX Production Pipelines

Siddhant Bhausaheb Sawkar (s5629020)

Master's Project Thesis

MSc Computer Animation and Visual Effects

Bournemouth University

August 2025

Acknowledgement

I would like to express my sincere gratitude to my supervisors, Jon Macey, and Jian Chang for their invaluable guidance, support, and encouragement throughout this project. Their expertise and feedback were instrumental in shaping the direction and successful completion of this thesis.

Additionally, I would like to thank the academic and technical staff at the National Centre for Computer Animation for providing the resources and knowledge that made this project possible.

Abstract

In the global Visual Effects (VFX) industry, how artists are paid can differ greatly from one region to another. A key reason for this disparity is the challenge of accurately tracking work hours, as manual methods are often unreliable. This thesis presents a practical solution: an automated time-tracking system designed specifically for the way VFX artists work. The system plugs directly into major creative software like Autodesk Maya and Blender, using their native Python APIs to monitor activity. It intelligently detects when an artist is working and when they are idle, ensuring that only active work time is recorded. The tool is built on a client-server architecture, with a central Flask server managing the data in a SQLite database. The final result is a reliable platform that provides artists, studios, and freelancers with a clear and verifiable record of their work hours, complete with reporting and data export capabilities. This project demonstrates how a targeted technical solution can provide the foundation for more transparent and equitable pay in the global creative industry.

Table of Contents

List of Figures

1. Introduction

The Visual Effects (VFX) industry is a fiercely competitive, time-sensitive place. To stay profitable, studios of all sizes must accurately bid on projects and effectively manage their resources. To keep clients' trust, freelance artists who make up a sizable portion of the workforce must provide clear and accurate billing. Accurate data, specifically the precise amount of time spent on each task in the production pipeline, is essential to these business operations.

Accurate time tracking, however, has long been a problem for the sector. A major bottleneck that frequently results in inconsistent and unreliable data is the use of manual logging. This absence of precise data has broad ramifications that impact the entire industry globally. It can result in underbidding on projects, which can cost studios money or force them to work "crunch" overtime. It causes blind spots, which makes it challenging for production managers to spot inefficiencies and distribute resources wisely.

Inaccurate data is a fundamental issue that contributes to systemic labor problems. According to a recent survey conducted by the VFX-Union, roughly 70% of VFX employees said they had worked unpaid overtime (VFX-Union, 2022). The risk is high, and the reward is frequently low for artists in what has been called a culture of "insane hours" and low pay (Welk, B, 2023). The fundamental requirement for a verifiable record of work is universal, even though these problems are more severe in areas with project-based pay structures.

Through the development and implementation of an automated time and activity tracking system, this project tackles this industry wide issue. A client-server application, the system is designed to work directly with industry-standard Digital Content Creation (DCC) programs like Blender and Autodesk Maya. The system offers a precise, verifiable, and non-intrusive

record of hours worked by automating the tracking process and integrating features like idle detection. The goal is to develop a tool that benefits all parties involved: giving studios the business data they require for precise management and bidding, giving independent contractors an effective billing system, and guaranteeing that every artist has an open record of their work.

2. Related Work

2.1 Commercial Production Tracking Solutions

Production tracking is a well-established concept in the creative industries, and there are a number of robust commercial software options available. An analysis of these systems serves as a basis for comprehending the issue domain and determining the areas in which this project makes an additional contribution.

Flow Production Tracking (formerly ShotGrid) from Autodesk is one of the most well-known solutions (Autodesk, 2024). In addition to time tracking, this all-inclusive platform provides asset management, review and approval processes, and deep integration with numerous DCC apps. Similar to this, **ftrack** offers a versatile production tracking and project management solution that is popular in the media and entertainment sector (Foundry, 2025).

What's Missing: Despite their strength, these platforms' main drawbacks are their expense and complexity. For smaller studios, teams, or independent contractors whose primary need is precise time tracking, their all-inclusive studio management system design may be too much. Entry barriers are created by their large feature sets, which frequently come with a high learning curve and a substantial financial investment.

Solution: In order to close this gap, this project concentrates on one crucial function: automated time tracking that is integrated with DCC. It offers the essential advantages of precise time logging without the overhead and expense of a full-scale production management suite by offering an open-source, lightweight, and simple-to-deploy substitute.

2.2 General-Purpose Time Tracking Software

There are many general-purpose time-tracking apps, such as Toggl Track (Toggl, 2025) and Harvest (Harvest, 2025), in addition to the VFX-specific tools. Freelancers and consultants from a wide range of industries use these tools extensively because they are great for basic time logging across multiple tasks.

What's Missing: These tools' primary flaw in a VFX setting is their shallow integration with DCC software. Artists are required to manually start, stop, and annotate timers because they function outside of the creative applications. Human error, like forgetting to start or stop the timer, is common in this manual process, which also fails to automatically capture the context of the work being done (e.g., the specific project or task).

Solution: The VFX Time Tracker addresses this issue by operating entirely within the DCC applications. In order to guarantee that the tracked time accurately reflects the work being done, it uses activity monitoring and idle detection in addition to automating the start and stop process based on file events. This gives the artist much more dependable data and relieves them of the burden of manual tracking.

2.3 Open-Source and Alternative Solutions

Popular substitutes have also been created by the open-source community. For example, Kitsu is a popular production tracking tool for VFX and animation (CGWire, 2025). Prism Pipeline is another useful tool that offers a framework for handling VFX workflows.

What's Missing: Although these tools are easier to use than their commercial counterparts, automated time tracking is frequently not given priority. Prism is more concerned with asset and pipeline management, while Kitsu primarily uses manual time tracking. Although they show a desire for open, community-driven solutions, their main selling point is not a specific, automated time tracking feature.

Solution: The goal of this project is to address this particular need in the open-source community. It can be used as a stand-alone application or possibly incorporated into other open-source pipelines to fill the gap in automated, transparent time tracking.

2.4 Academic and Industry Research

Research in this field, both academic and commercial, frequently focuses on the relationship between technology, workflow effectiveness, and the human elements of creative production. Reports from the industry, including those issued by groups like the Visual Effects Society, usually emphasize the necessity of increased production practice standardization and efficiency (Visual Effects Society, 2020). Although these resources usually don't offer comprehensive technical designs for time tracking systems, they do provide insightful information about the demands of the industry and the difficulties that a program such as the VFX Time Tracker seeks to solve. Furthermore, studies on human-computer interaction (HCI) indicate that users are more likely to embrace automated, non-intrusive data collection techniques than those that necessitate frequent manual input (Shneiderman, 2010).

3. Technical Background

3.1 System Architecture: The Client-Server Model

The VFX Time Tracker system was developed using a combination of programming concepts and well-established software technologies. The client-server model upon which the architecture is built enables centralized data management and the communication of numerous, dispersed clients with a single, authoritative source. The main.py, dcc_client.py, and server.py files all use this architecture, which was selected for its scalability and separation. The client (the reporting application and the DCC plugins) are in charge of the user interface and interaction, while the server manages all business logic and data persistence.

3.2 Backend Technologies

The system's backend, which is stored in server.py, is in charge of managing data, authentication, and business logic.

Flask Web Framework: Flask, a lightweight and adaptable Python web framework, is used in the construction of the server (Ronacher, A., 2010). Flask was selected because of its ease of use and low boilerplate, which make it perfect for developing a targeted RESTful API. The server.py-defined API offers endpoints for data retrieval (/api/get_logs), session control (/api/session/start, /api/session/stop), and user management (/api/register, /api/login). The implementation handles requests and routes according to standard Flask patterns (Ronacher, 2010). To ensure that user credentials are not saved in plain text, the werkzeug.security library is used to generate and verify password hashes.

SQLite Database: The system uses SQLite, a standalone, serverless SQL database engine, for data persistence. Because SQLite doesn't require a separate database server, it's easy to set up and deploy. Schema.sql defines the database schema, which is intended to store data about tasks, users, and work sessions in an organized, relational structure. The built-in sqlite3 module in Python is used to manage the database interaction, adhering to the guidelines outlined in the official Python documentation (Python Software Foundation, 2025).

3.3 DCC Integration Technologies

This project's deep integration with DCC software is one of its primary features. This is accomplished by utilizing the native Python scripting APIs offered by each application in a hybrid method that blends automatic event handling with a user-driven workflow.

Blender Python API (bpy): A standard extension is used to implement the Blender integration. Blender_tracker_integration.py's main logic makes use of the bpy.app.handlers module. These handlers serve as a "safety net" to guarantee that sessions are saved and are persistent callback routines that Blender runs in response to particular events, including quit_post (before to Blender exiting) (Blender Foundation, 2025).

The Maya integration follows a similar methodology to the Autodesk Maya Python API (maya.cmds). Upon starting, Maya automatically runs the userSetup.py script. cmds.scriptJob() is the foundation of the activity tracking in maya_tracker_integration.py. This command generates background-running jobs that run code in response to SelectionChanged and other events. This gives the server a reliable way to receive regular heartbeats, which is necessary for the idle detection system (Autodesk, 2024).

3.4 Client-Side and Reporting Technologies

The DCC plugins and two separate reporting apps are the two client kinds that are part of the system.

- HTTP Communication (Requests): The requests library is used by all clients to communicate with the Flask server. Making HTTP queries to the API endpoints is made easier with the help of this library.

- Concurrency for Idle Detection (threading): Python's built-in threading module is used to build the idle detection technique in dcc_client.py. To monitor for user inactivity without freezing the main DCC application, a session starts a secondary, non-blocking thread (Python Software Foundation, 2025).

- Desktop GUI (Tkinter): Tkinter, the standard Python GUI toolkit, is used to create the stand-alone artist reporting application in main.py. The sv_ttk theme is used for a more contemporary appearance and feel.

- Data Visualization (Matplotlib & Pandas): The system creates charts using the Matplotlib library to present informative reports in the desktop application. The Pandas library is used to initially collect the data for these charts (McKinney, 2010).

- Web Frontend (HTML, Tailwind CSS, Chart.js): This web application is the dashboard that the manager sees. Tailwind CSS, a utility-first CSS framework, is used to style the index.html file. A well-known JavaScript charting package called Chart.js

is used to render the dashboard's interactive charts (Chart.js, 2025). The client-side JavaScript also includes functionality to convert the currently filtered data into a CSV file for download.

4. Implementation Overview

4.1 System Architecture and Execution Flow

A client-server architecture was used in the project's construction. The run.py script starts the process by first determining whether the SQLite database is there and, if not, initializing it with the schema.sql file. The Flask server is then launched, enabling access to the web dashboard and API.

4.2 Server and API Development

The server.py file is the project's central component. This script creates a number of API endpoints that customers can communicate with using Flask (Ronacher, A., 2010). For example, the Maya plugin sends user data to the /api/login endpoint when a user submits their information. As a typical and safe procedure, the server then uses functions from the werkzeug.security package to compare the supplied password with a hash that is stored in the database (Ronacher, 2010). Passwords are never saved in plain text. A confirmation is sent back by the server if the login is successful. In a similar manner, when an artist selects "Start Tracking," the client makes a request to /api/session/start, and the server adds the current timestamp to a new item in the database's sessions table.

4.3 Database Schema Design

The schema.sql file defines the whole data structure, which is then controlled by a SQLite database. Because SQLite doesn't require a separate server process, this decision was made for simplicity's sake. The four main tables that make up the schema are users, tasks, sessions, and activity_events. A specified list of production tasks, such as "Modeling" or "Animation," is contained in the tasks table, while login information is stored in the users table. Every work period is documented in the sessions table, which is the most crucial table. In addition to storing important information like the start time, finish time, and total length, each row in this database represents a block of time and is associated with a particular user and work.

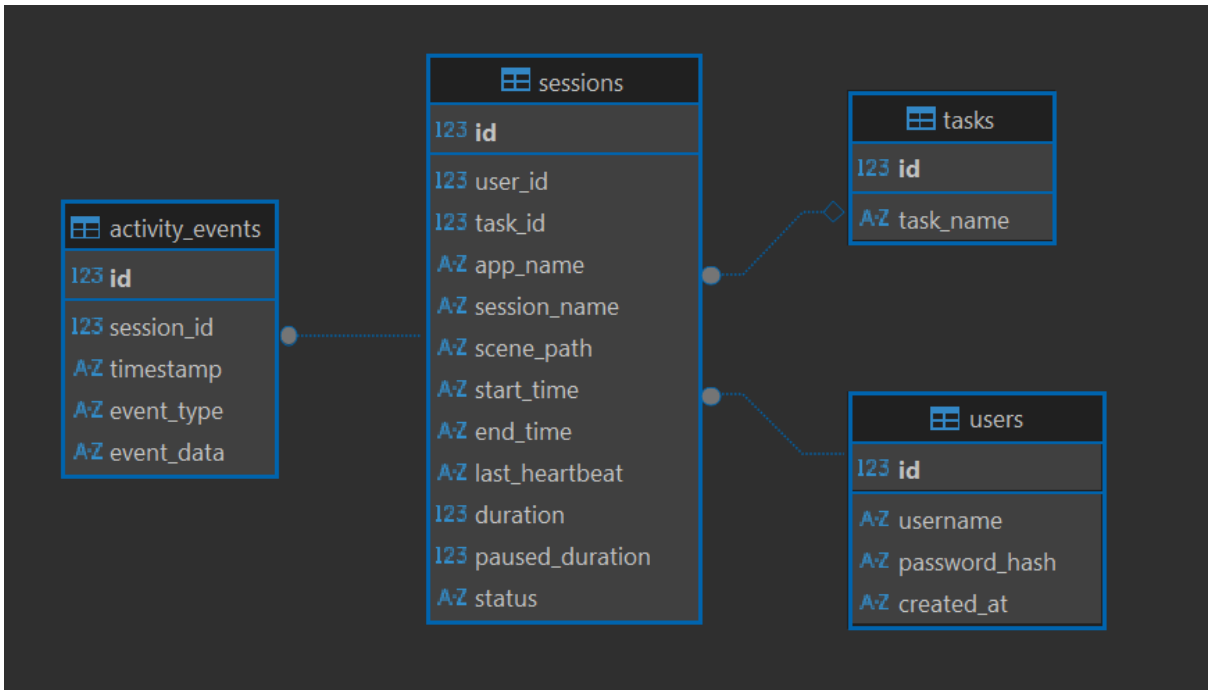


Figure 4.1: Diagram of the database schema.

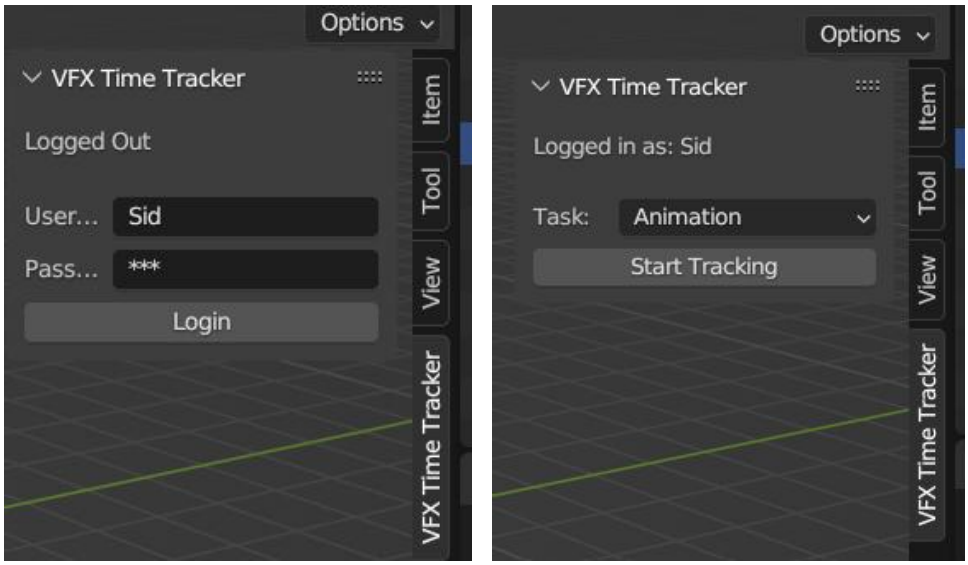
4.4 The dcc_client Module and Idle Detection

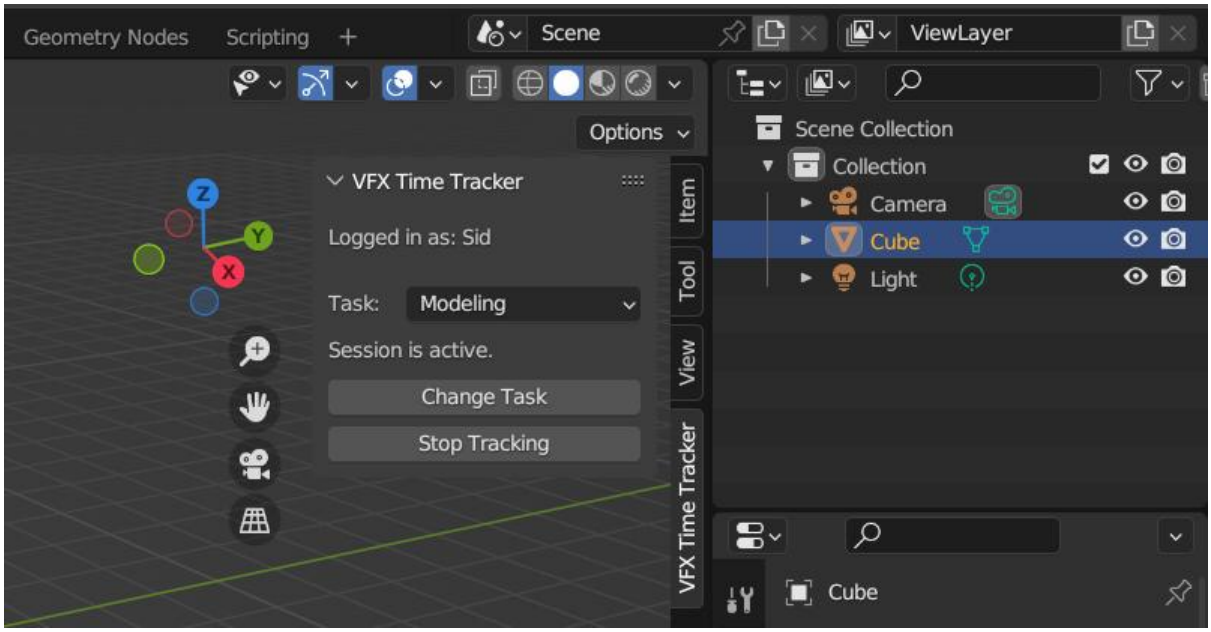
In order to prevent writing identical code for the Blender and Maya plugins, a shared dcc_client.py module was developed. A DCCClient class, which manages all server communication, is included in this module. One of the main functions of the project is idle detection, which is the client's responsibility when an artist begins a session. Python's

threading module is used to launch a background thread (Python Software Foundation, 2025). Because doing the idle check in the main DCC thread would block the entire application, this was a crucial technical detail.A straightforward loop that determines the time since the last activity was recorded is executed by the background thread. In order to prevent time spent away from the keyboard from being charged, the client automatically asks the server to pause the session if this duration is over a certain threshold (for example, ten minutes).

4.5 Blender Addon Integration

The Blender integration was developed as a standard add-on. The user interface is configured by the main __init__.py file, which adds a new panel to the sidebar of the 3D View. The artist enters in to this panel, chooses their task from a drop-down menu that is filled with server data, and then manually hits "Start Tracking." Additionally, blender_tracker_integration.py contains the actual event handling functionality. As a safety net, it makes use of Blender's application handlers (bpy.app.handlers). To prevent data loss, the quit_post handler, for example, makes sure that the session is automatically halted and stored if an artist exits Blender without manually stopping the timer (Blender Foundation, 2024).

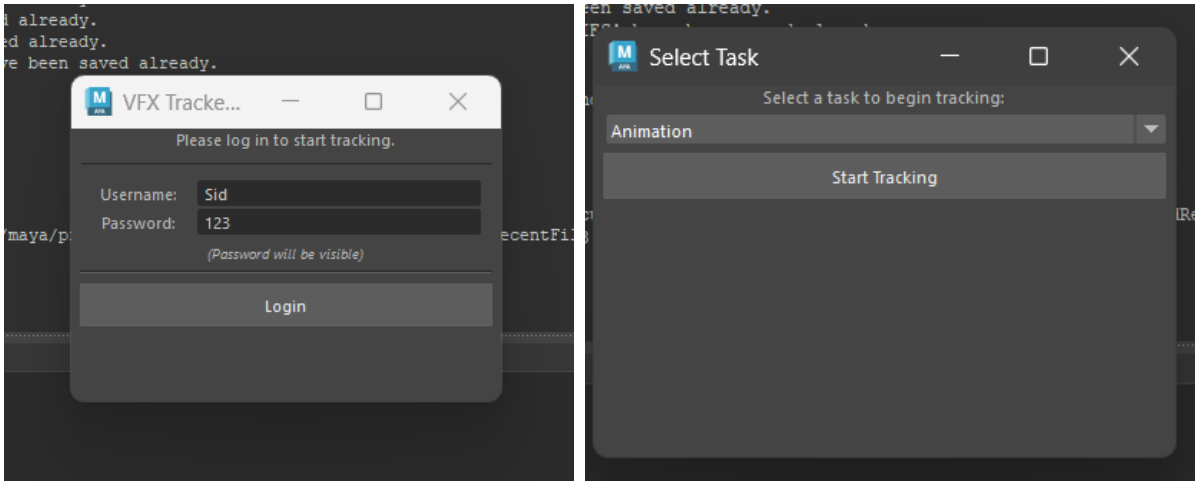




4.2 Blender UI panel showing the login and task selection state.

4.6 Maya Scripting Integration

Because of the way Maya handles startup scripts, the methodology was a little unusual. When Maya is launched, a userSetup.py file is placed in the scripts directory and executed automatically. This script calls the main integration code from maya_tracker_integration.py using the evalDeferred command. This is a crucial step since, as Autodesk (2024) notes, it avoids attempting to generate the tracker's login window until Maya's user interface has completely loaded. Activity monitoring starts as soon as the user logs in and chooses a task. ScriptJob commands, which are strong event listeners, are used to do this. To listen for events such as SelectionChanged, a job is established.This event resets the idle timer each time it occurs by sending a "heartbeat" to the server to indicate that the user is engaged.



4.3 Maya UI window for login and task selection.

4.7 The Artist Reporting Application

Tkinter was used to create the main.py file, a stand-alone desktop application that allows artists to view their own monitored data. The user is shown a dashboard with a calendar to choose a certain day after logging in. The application sends a call to the server's /api/get_logs endpoint when a day is chosen. The Pandas library is then used to process the returned data and aggregate the time spent on each job and application (McKinney, 2010). Matplotlib is then used to show this condensed data. Using the FigureCanvasTkAgg class, a common integration technique, the charts—like a pie chart for task breakdown—are embedded straight into the Tkinter window (Hunter, 2007).

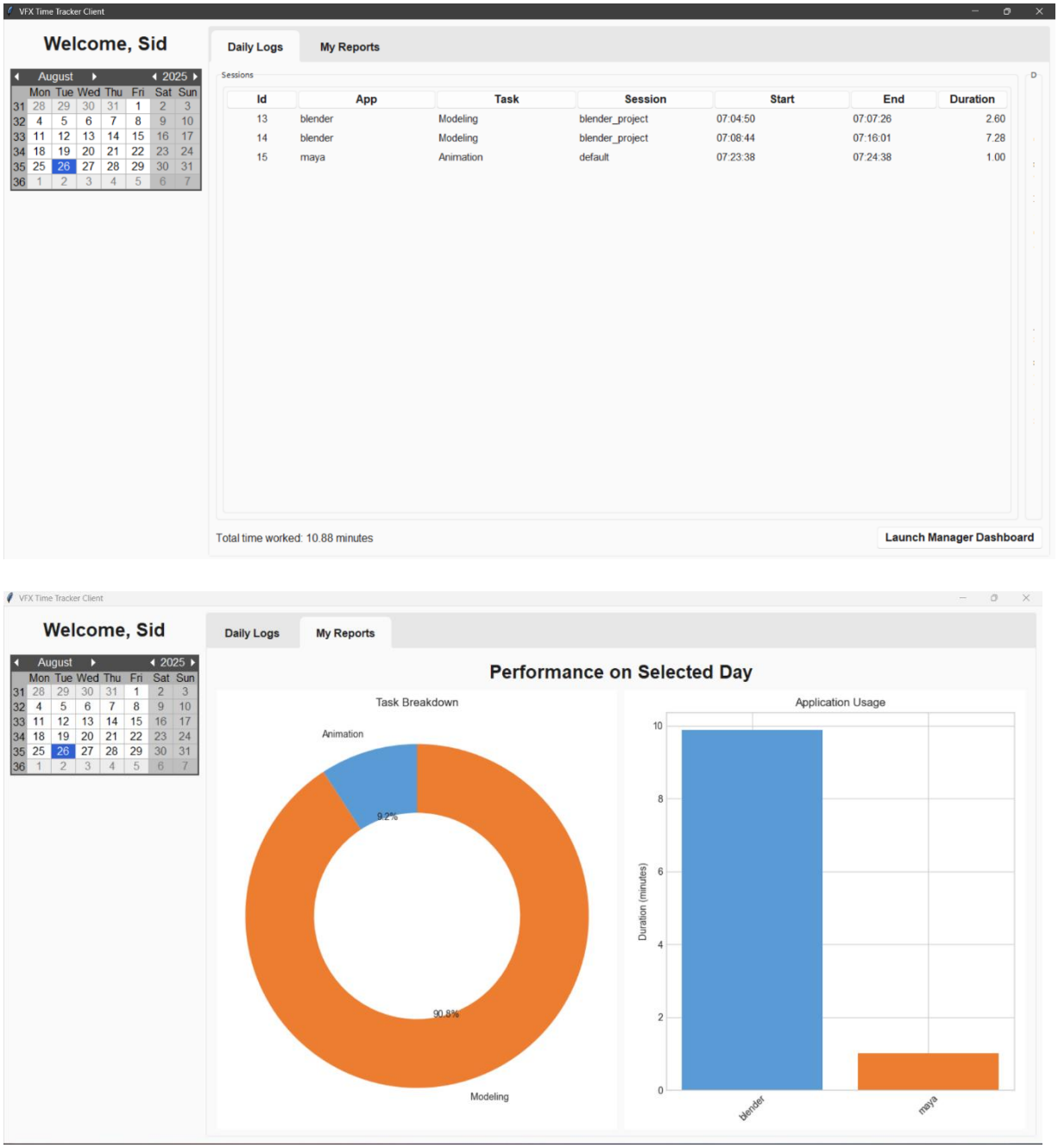


Figure 4.4: Artist reporting application's dashboard.

4.8 The Web-Based Manager Dashboard

The system has a web-based dashboard for managers, which is provided in index.html, in addition to the artist-facing capabilities. A Flask route in server.py serves this dashboard directly. It offers a comprehensive summary of studio output. For a contemporary, responsive

design, Tailwind CSS is used to style the page after it was constructed using regular HTML. Client-side JavaScript powers all of the dynamic information, including tables and charts. The site calls the server's /api/dashboard_stats endpoint via AJAX when it loads. The pertinent data is retrieved by the server, processed by Pandas, and returned as a JSON object.The data is then displayed as interactive pie and bar charts by the JavaScript on the page using the Chart.js library (Chart.js, 2025). This enables managers to filter data by date or artist and view real-time updates to the findings. The "Export to CSV" button is a crucial component of this dashboard. The client-side JavaScript handles all of this functionality. It provides managers with an easy and efficient way to obtain the raw data for additional analysis in other software by taking the currently filtered dataset, formatting it into a CSV string, and then using a Blob object and an anchor tag to initiate a file download in the browser.

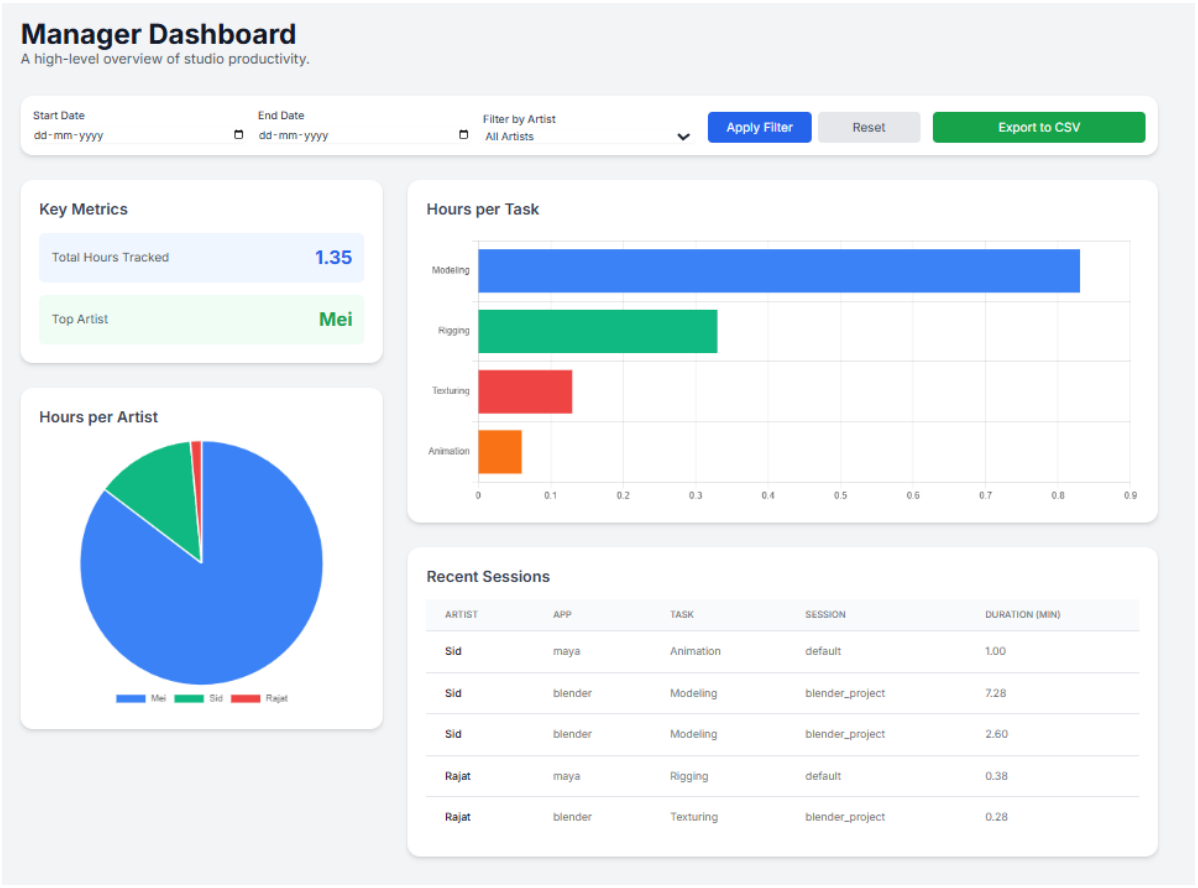


Figure 4.5: Web-based manager dashboard.

4.9 Core Time Calculation Logic

The reasoning used to determine the length of work sessions, especially how it takes into account times of inactivity, determines how accurate the time tracker is. When a session is paused or terminated, this logic is implemented by the server-side code in server.py.

The length of the pause is computed and added to the session's total paused_duration when a session is resumed following a period of inactivity. Let T_pause_start be the timestamp at which the idle state was initially identified, and T_resume be the timestamp at which the user returns to an active state. The following formula is used to determine the current pause's duration, $D\_pause\_current$, in minutes:

$$\boldsymbol{D\_pause\_current = 60(T\_resume - T\_pause\_start)}$$

The pause time that has already been accrued for that session is then increased by this value.

The total active duration (D_active) is computed at the end of a session. Let $T\_start, T\_end, and\ D\_paused\_total$ represent the start and end times of the session, respectively, and the total accumulated pause duration in minutes. Next, the active duration is provided by:

$$\boldsymbol{D\_active = 60(T\_end - T\_start) - D\_paused\_total}$$

This guarantees that the artist's labor is measured more fairly and accurately by only recording the time they were actively working.

5. Conclusion

This project effectively illustrates how the entire VFX industry can gain a great deal from an automated time tracking system that integrates with DCC. Studios can bid on projects more precisely, allocate resources more effectively, and communicate with clients more openly thanks to the system's accurate data. It provides freelance artists with a dependable and verifiable way to bill, fostering stronger business ties and guaranteeing just compensation. It offers a vital tool for verifying hours worked and guarding against unpaid overtime for all artists, regardless of location or payment model.

The created application offers a reliable, non-intrusive, and user-friendly solution to the problems with manual time logging. The main goal of developing a precise and automated time tracker was accomplished.

This project's future work could concentrate on a few important areas:

- Improved Reporting: More advanced reporting features, like tracking project budgets, artist performance metrics, and payroll system data export options, could be added in the future.
- Further DCC Integration: The system might be expanded to accommodate additional popular DCC programs used in the visual effects sector, like Adobe Substance Painter, SideFX's Houdini, and Foundry's Nuke.
- Machine Learning for Task Prediction: To further reduce manual input, an advanced feature might automatically suggest or assign the right task based on an analysis of an artist's workflow.
- Cloud Deployment and Scalability: Using more scalable database solutions like PostgreSQL, the system could be modified for deployment on cloud platforms like AWS or Google Cloud in order to accommodate larger studios.

References

Autodesk, 2024. *Maya Python API Documentation*. [online]. Available from:https://help.autodesk.com/view/MAYAUL/2024/ENU/?guid=Maya_SDK_Maya_Python_API_html[Accessed 26 August 2025].

Autodesk, 2024. *Flow Production Tracking*. [online]. Available from: https://www.autodesk.com/products/flow-production-tracking/overview [Accessed 26 August 2025].

Blender Foundation, 2025. *Blender Python API Documentation*. [online]. Available from: https://docs.blender.org/api/current/ [Accessed 26 August 2025].

CGWire, 2025. *Kitsu*. [online]. Available from: https://www.cg-wire.com/kitsu [Accessed 26 August 2025].

Chart.js, 2025. *Chart.js Documentation*. [online]. Available from: https://www.chartjs.org/docs/latest/ [Accessed 26 August 2025].

Foundry, 2025. *ftrack*. [online]. Available from: https://www.ftrack.com/ [Accessed 26 August 2025].

Harvest, 2025. *Harvest: Time Tracking Software With Invoicing*. [online]. Available from: https://www.getharvest.com/ [Accessed 26 August 2025].

Hunter, J. D., 2007. Matplotlib: *A 2D Graphics Environment. Computing in Science & Engineering*, 9(3), pp.90-95. Available from: https://ieeexplore.ieee.org/document/4160265 [Accessed 26 August 2025].

McKinney, W., 2010. *Data Structures for Statistical Computing in Python. In: Proceedings of the 9th Python in Science Conference,* pp. 51-56. Available from: https://proceedings.scipy.org/articles/Majora-92bf1922-00a[Accessed 26 August 2025].

Prism Pipeline, 2025. *Prism Pipeline - Animation and VFX Pipeline*. [online]. Available from: https://prism-pipeline.com/ [Accessed 26 August 2025].

Python Software Foundation, 2025. *The Python Standard Library*. [online]. Available from: https://docs.python.org/3/library/ [Accessed 26 August 2025].

Ronacher, A., 2010. *Flask Documentation*. [online]. Available from: https://flask.palletsprojects.com/ [Accessed 26 August 2025].

Welk, B., 2023. *VFX Workers Are Ready to Unionize: 'The Risk Is So High, the Reward Is So Low*'. [online]. IndieWire. Available from: https://www.indiewire.com/features/general/vfx-

workers-unionize-high-risks-low-wages-insane-hours-1234814830/ [Accessed 26 August 2025].

Shneiderman, B., 2010. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 5th ed. Pearson.

Toggl, 2025. *Toggl Track*. [online]. Available from: https://toggl.com/track/ [Accessed 26 August 2025].

VFX-Union, 2022. *2022 Survey Results*. [online]. Available from: https://vfxunion.org/2022-survey-results/ [Accessed 26 August 2025].

Visual Effects Society (VES), 2020. *The VES Handbook of Visual Effects: Industry Standard VFX Practices and Procedures*. 3rd ed. Focal Press.

Appendices

Source Code

https://github.com/ohsidno/VFX_Time_Tracker.git

API Endpoint Documentation

This section provides a summary of the key API endpoints created in server.py that enable communication between the clients and the server.

User Management

- Endpoint: /api/register
- Method: POST
- Description: Creates a new user account.
- Request Body (JSON):
- username (string): The desired username.
- password (string): The desired password.
- Success Response (201):
- status: "success"

- message: "User created successfully."
- Error Response (409):
- status: "error"
- message: "User [username] is already registered."
- Endpoint: /api/login
- Method: POST
- Description: Authenticates a user and returns their information.
- Request Body (JSON):
- username (string): The user's username.
- password (string): The user's password.
- Success Response (200):
- status: "success"
- user: { "id": (int), "username": (string) }
- Error Response (401):
- status: "error"
- message: "Invalid username or password."
- Session Management

- Endpoint: /api/session/start
- Method: POST
- Description: Starts a new time tracking session.
- Request Body (JSON):
- user_id (int): The ID of the user.
- task_id (int): The ID of the selected task.
- dcc_name (string): The name of the DCC application (e.g., "maya").
- project_name (string): The name of the project.
- scene_name (string): The name of the scene file.
- Success Response (201):
- status: "success"
- session_id: The ID of the new session.
- Endpoint: /api/session/stop
- Method: POST
- Description: Stops an active session and calculates the final duration.
- Request Body (JSON):

- session_id (int): The ID of the session to stop.
- Success Response (200):
- status: "session_stopped"
- Endpoint: /api/session/heartbeat
- Method: POST
- Description: Updates the last_heartbeat timestamp for an active session.
- Request Body (JSON):
- session_id (int): The ID of the active session.
- Success Response (200):
- status: "acknowledged"
- Data Retrieval
- Endpoint: /api/tasks
- Method: GET
- Description: Retrieves the list of all available tasks.
- Success Response (200):
- status: "success"
- tasks: [{ "id": (int), "task_name": (string) }, ...]

- Endpoint: /api/get_logs
- Method: GET
- Description: Retrieves all stopped sessions for a specific user on a specific date.
- Query Parameters:
- user_id (int): The ID of the user.
- date (string): The date in 'YYYY-MM-DD' format.
- Success Response (200):
- status: "success"
- logs: [{...session details...}]