Bournemouth University

NCCA MSc CAVE Thesis

Production Tracking Software for VFX and Animation

Alesia Karchevskaia

August 2025

Abstract

This thesis presents Tracktor VFX, a web-based production tracking tool developed to provide a free, accessible alternative to subscription-based trackers for VFX and animation, targeting individuals, hobbyists, and small student teams. The paper covers the review of existing production tracking solutions and technologies used in web application development, as well as a detailed coverage of Tracktor's architecture. All in all, Tracktor is implemented as a single-page application (SPA) using React+Flask+SQLite tech stack, providing basic production tracking capabilities, such as project creation, management of shot/asset statuses and notetaking.

Contents

1 Introduction

Animation and VFX production is a lengthy and complicated process. The data flow between departments and constant back-and-forth, requires special care and attention to the state of affairs. Therefore, in large teams it is very common to employ multiple production managers and coordinators and even more assistants.

Yet a production department would not be able to do its duties most effectively without specific tools that can aid in their ability to track and allocate resources, as well as communicate with artists on tasks and priorities. This is a very challenging task to accomplish, relying solely on work chats and spreadsheets. Hence, enter the production management softwares.

The market for animation and CGI is full of highly specialised software, and the production aspect has not escaped it. Currently, Autodesk - one of the largest players on the CGI market supports its own tracking software with an ever-changing name, which at the moment of writing is Autodesk Flow. Autodesk is still the largest player on the market, though other contestants like Kitsu or ftrack are getting more and more traction (Rousseau, 2022). What makes these softwares so popular is their specific catering to the nature of a film/VFX project, which might be lacking in other production trackers, for example, data entities like shots, assets, versions and complicated data-flow in between.

However, despite the existing software, it has been an ongoing challenge against the exclusively proprietary software, which leaves out a lot of hobbyists and students, just entering the craft (Silver Monkey Studio, 2025). Production tracking has not been immune to these challenges either, since most of the tracking softwares mentioned do not offer an indie or even free tier, with some offering a courtesy trial period of one week (CG-Wire, 2025). This leaves a place for a smaller-scale tracker with the basic functionality available for free, which can still aid the smaller production with its vital features, but might not have all the bells and whistles that Autodesk Flow has, for example.

Hence, the purpose of this thesis is to research and implement a basic tracking software for VFX and animation, targeted at individuals and small teams (freelancers, students), which can be shared under a free and open-source model.

2 Previous Work

Production trackers aren't a new phenomenon, and neither are they unique to the animation and VFX field. Some of the most common dedicated production software include brands like Jira, Trello, Notion, Asana and many others, with the Webmasters team estimating that Jira remains the most popular production management tool in the world, with a market share of 38.7% (WMTips, 2025). Such services often provide features like planning, Gantt charts, task assignment, etc (Atlassian, 2025).

However, such products do not always satisfy the requirements of a media production, especially related to film work. Thankfully, the computer graphics industry has come up with its own solutions - often proprietary - which best cater to the specifics of the production. A giant in CAD and CGI development - Autodesk, has been supporting its production management platform with the ever-changing name, currently called Autodesk Flow, for more than 15 years (Autodesk, 2025). Other competitors in the field include Kitsu (CG-Wire, 2025), ftrack (ftrack, 2025) and some smaller tools like Palette (Palette, 2025) and Ayon (AYON, 2025).

While the big players provide the most functions and support, they are often targeted at large studios and primarily commercial use, which makes them very oriented towards a company as a consumer, rather than an individual. This means that the most popular trackers often do not have a free tier for hobbyists or even lover lower-priced deal for freelancers, essentially gatekeeping those, who do not have a studio backing them up.

Smaller tools like Pallette and Ayon seemed to be addressing this issue by introducing features catering to smaller teams and providing a self-hosting free tier. However, as of August 2025, the free tier of Pallete is unavailable (Palette, 2025), and Ayon only offers the self-hosted solution for individuals (AYON, 2025), which is limiting in case of a small hobbyist/student team, and can be too complicated to set up for artists with no technical background.

Therefore, the niche for a simple, free and open-source tracker is still open and can be a good alternative for individuals and teams, who would like to track their project, but also do it in a more specialised environment than Google Sheets.

3 Technical Background

Most of these popular solutions are SaaS (Software as a Service) platforms in one way or another, which means that they provide their users with complete infrastructure, usually over the web (Microsoft, 2025). Such a business model offers several advantages for production management: users can access the software from anywhere, updates are handled centrally by the vendor, and regardless of whether users are self-hosting or calling the application from the vendor's server, the trackers are usually accessed through a web browser, which enables collaboration between different users.

Users pay only for the software they want, which comes complete from the web-app of their choice. There is no requirement to extend it to start using the software.

However, a lot of these applications also offer an API which Pipeline TDs and Developers can use to integrate the connection into their studio's pipeline or expand the functionality of the tool, according to their studio's needs.

Hence, to implement a similar solution, would be to build a **web application** which can safely store all the users' data and also present the users with a visual platform to access and interact with it.

A web application is a software that runs in a web browser, rather than being natively installed on a user's computer. They usually require a server that will manage data and requests, and this feature enables simultaneous collaboration between several users, potentially on any device that has access to a browser (Fowler and Stanwick, 2004, pp. 22-34).

Generally, web applications use client-server computing model. According to Yadav and Singh (2009, p. 1) "this model is based on distribution of functions between two types of independent and autonomous processes: Server and Client. A Client is any process that requests specific services from the server process. A Server is a process that provides requested services for the Client. Client and Server processes can reside in same computer or in different computers linked by a network." This architecture can allow simultaneous data exchange, possibly from several clients, depending on how the app is built.

The **frontend** usually runs on the client-side and is the part of the application that is intended to be visible to the user. Normally, this is the only part of the app the user can interact with, and in turn, it will present it with the relevant data, according to the user's requests. Typical technologies include HTML, CSS and JavaScript, since the frontend is usually run in a browser. Frameworks like React and Vue can provide additional development functionality and ease up the workload on the developers.

The **backend** operates on the server-side and handles the application logic. It processes any requests coming from the client, and often connects to the **database** to retrieve the necessary data. It is the backend that defines all the core functionalities of how the application will behave.

To connect between frontend and backend, many web applications use a **RESTful API**, which facilitates the aforementioned client-server requests, usually in the JSON format (Doglio, 2018, pp. 8-9). This allows the client to communicate with the backend endpoint (a specific URL that handles requests) using methods such as GET or POST, etc. Together, these define which resource the client can access and how it can work with them, for example, by retrieving data or writing it into the database.

The said database will store any data that needs to be saved for the app functionality. Usually, the preference stands with the **relational databases** , which organise the data into consistent tables, structured of rows and columns, and the most popular tech are different flavours of SQL databases (SQLite, MySQL, PostgreSQL, etc); **Non-relational databases** do not have this rigid composition, so they work better with unstructured data, such as blogs, where each entry can look different and will struggle to match SQL defined structure. Common tech in this area is MongoDB (document-based) and Cassandra (column-based).

In addition, with the client-server architecture, one server can handle multiple clients. Which means, that given access to the backend API, a knowledgeable Technical Director would be able to connect a different piece of software as an alternative client for the backend. This is essentially what powers the DCC integration in larger studios with production tracking tools.

4 Tracktor VFX

4.1 Goals and Requirements

At the start of the development, a Requirements document was written to outline the goals and scope of the production. It states that: "The aim of the project is to produce a basic production tracking tool. First and foremost, it should aid users in managing and tracking their shot work in a website-like manner" (Karchevskaia, 2025*b*). The project is also expected to be targeted towards the smaller teams and individuals, providing them with the core functionality of a project tracker. The core functionality is defined as the ability to create a project and populate it with shots/assets, status of which can be tracked (Karchevskaia, 2025*b*). Some of the extended functionality includes "User login/sign up system", "Storing only projects relevant for the user in the user's account" and "Enabling sharing of the project by users between multiple users" (Karchevskaia, 2025*b*).

It was decided that the project should take the form of a web application, to make it accessible for those without technical knowledge, as well as enable people to share the projects they want to track without having to use cloud or network drives.

The requirements document also introduces the tech stack for the development, which reads as follows:

- **Node.js**: Serves as a JavaScript run-time environment, which enables executing JavaScript outside of the browser.
- **Vite**: Frontend build tool for JavaScript.
- **React**: A JavaScript framework based on component design, which simplifies creating interactive interfaces.
- **Tailwind CSS**: A CSS framework that provides predefined classes for common styling tasks, which speed up the workflow, while maintaining the flexibility of the styling.
- **Flask (Python)**: A lightweight Python framework used to facilitate API requests.
- **SQLite**: A relational serverless database, well-suited for the small scope of the project, with the potential to switch to another SQL-flavoured database.

These core functionalities make up a proof of concept, which is the development goal of this thesis, with the purpose of demonstrating the project's basic functionality and potential for improvement and growth.

As a part of this demonstration, it would be nice to showcase a potential real-life application of the project. Since bigger studios often integrate their production trackers into their pipelines and storage systems, a similar extension could be thought for the project.

Open-source pipeline tools like TIK Manager (TikWorks, 2025) provide an organised and professional-level environment for anyone to build their project in. It already has Autodesk Flow and Kitsu integrations, so mimicking one of those for the thesis project can be a good example of a potential integration with DCC packages. The integration is defined as the ability for the TIK manager's plugin to demonstrate the connectivity with the app's API and pull a project from the web-app.

Thus, the project will consist of two parts: the first one is the web application itself. And to showcase its potential as a full-scale solution, a test integration into a DCC pipeline (a plugin), using an existing pipeline tool, will be the second.

The app was provisionally named Tracktor VFX, and later in this paper will be referred to as such.

4.2 Design

Next, a Design Document has followed, which outlined the approaches taken to implement the Requirements in more detail. It is available in the project's repository and serves as a reference point for the design decisions discussed in this section.

To start with, it is best to have a look at an application design on the highest level: "The architecture of Tracktor is based on a modular full-stack design that separates concerns between the frontend user interface, backend logic, and persistent storage. Each component communicates through a defined set of RESTful API endpoints, ensuring a clean separation and scalability. "(Karchevskaia, 2025*a*)

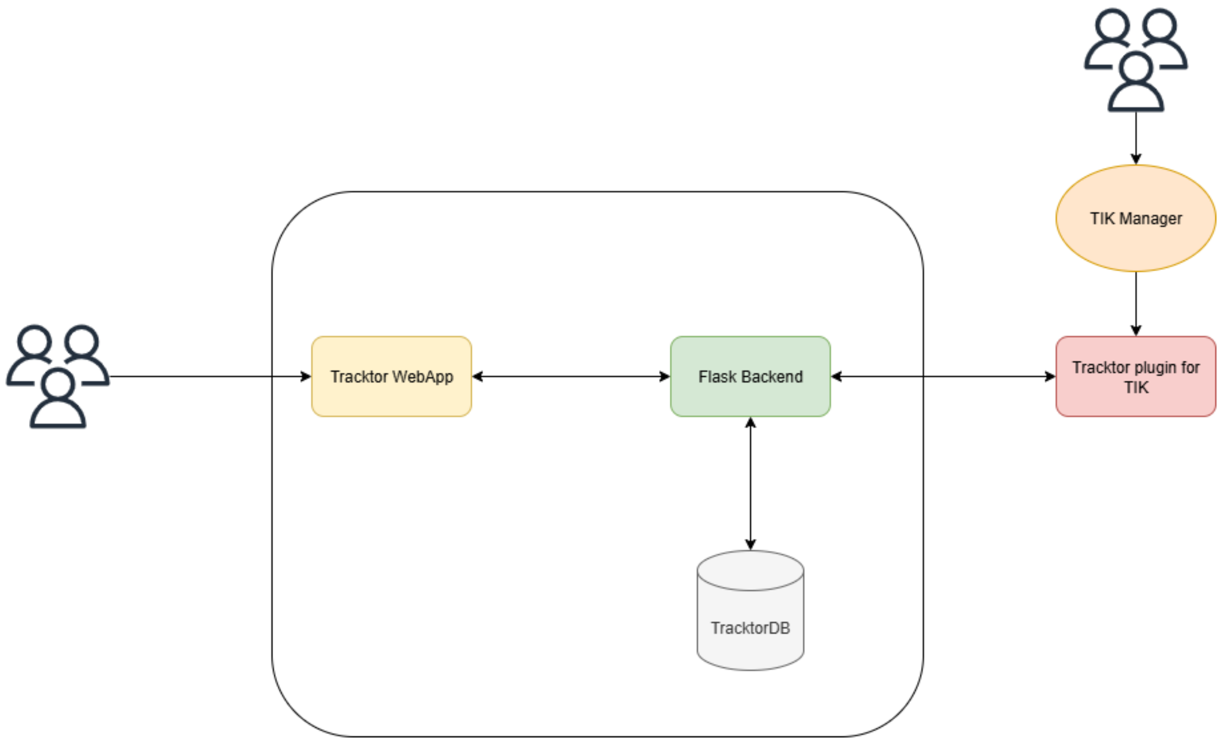


Figure 1: Tracktor VFX High-level design diagram

4.2.1 Frontend

The frontend for Tracktor VFX is a React single-page (SPA) application, with Tailwind CSS used for styling. It handles all of the user's interactions, HTTP requests, as well as data presentation.

As a single-page application, it loads a single HTML page and dynamically updates content without full page reloads, providing a smoother user experience (Fenollosa, 2022, pp. 159-187). Routing between pages is handled by App.jsx, which maps URLs to different page components.

The source code is organised into *components* and *pages* directories. The *components* directory includes reusable UI elements, some of which are written from scratch, and some feature other libraries like Headless UI. The *pages* directory is home to the combination of components that form the final view for the user and handle the HTTP requests.

Each such request (e.g. GET, POST, etc.) is defined by a specific function in a relevant page file. Those functions also often take in props like projectId or itemType, to dynamically fetch data from backend, depending on the URL. For example, given the URL with a projectId, the app will only fetch shots and assets relevant to that project. Together, this routing and request-handling mechanism allows the frontend to present interactive and up-to-date content to the user.

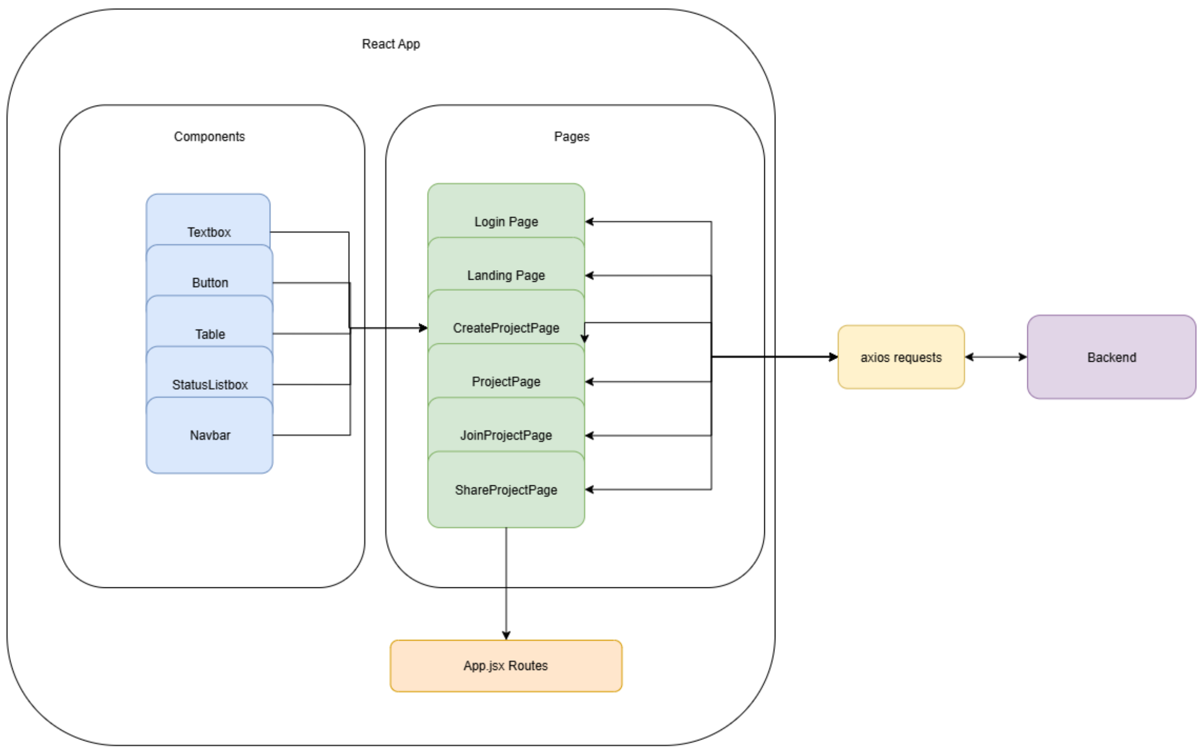


Figure 2: React Frontend structure diagram

UI design

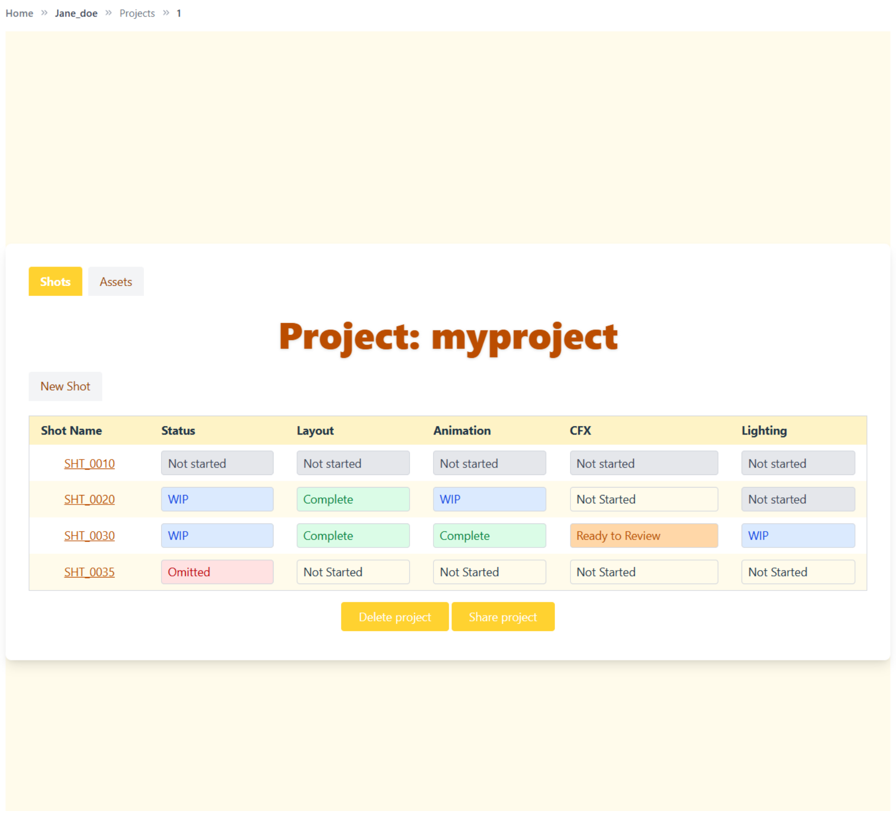
While UI/UX design wasn't the goal for this project, some conscious decisions have been made regarding how the user is supposed to interact with the application. For example, the shots/assets were to be displayed as a table, which is conventional in one way or another in other tracking softwares.

Also, to keep all the relevant information on one page and not make the user reload too much, tabs were introduced for the same functionality. For example, once the user is on the item page, they can switch between departments by using tabs, rather than dropdowns or nested reloads.

The statuses of the items were given a colour to differentiate them from each other, as well as the colour contrast of certain buttons was improved, after feedback from peers and academics, to increase readability.

In addition, to prevent users from constantly pressing the "back" button in their browser, a UI element known as "breadcrumbs" was added to enable backwards navigation. While it uses the while URL as its base, the sections unavailable for the app's URLs are unclickable, so only the existing pages can be accessed.

Overall, this is a functional UI for the proof of concept. However, it definitely needs improvement for a bigger release, and such arrangements will be in place in the Tracktor roadmap.



Project: myproject

Shot Name	Status	Layout	Animation	CFX	Lighting
SHT_0010	Not started	Not started	Not started	Not started	Not started
SHT_0020	WIP	Complete	WIP	Not Started	Not started
SHT_0030	WIP	Complete	Complete	Ready to Review	WIP
SHT_0035	Omitted	Not Started	Not Started	Not Started	Not Started

Figure 3: Sample UI for project page

Users

The Tracktor requires the user system to keep track of projects, assigned only to the users who are supposed to work on them. Since Tracktor has a common database for all the data that may be created, having this data segregated on a user basis is a vital functionality that had to be implemented early on.

User generation and authentication are based on a username and a hashed password at this point, no email address or personal data is necessary. This way is simple to implement and build other features on, which satisfies the requirement for the proof of concept. However, for a complete and deployed product, this will need to be improved in terms of security and proper authentication.

Potential improvements include implementing a token system to enable persistent user login sessions and user roles and permissions.

4.2.2 Backend

Tracktor's backend consists of the Flask application that handles the HTTPS requests, and helper table classes that abstract each database table and handle the SQL queries. With this approach, the backend is database agnostic and can be connected with a different flavour of SQL by just changing the SQL queries in a relevant class, or swapping the class files.

main.py is the main entry point for the Flask application, as it imports and initialises all the table classes, as well as handles the frontend requests. The initiated table classes together assemble the database. The backend then keeps running, either on a server or locally, and listens to any incoming requests. After receiving one, Flask routes it to the matching endpoint and executes the related function. The result of the function is then relayed to the client, and then the frontend processes it.

An example class structure can be found in this pseudocode:

Listing 1: Pseudocode for Project table class

```
class ProjectsTable:
    # Initialise with database connection
    def __init__(self, db_connection):
        self.db = db_connection

    # Create a new project
    def create_project(name, type, etc):
        INSERT INTO projects (...) VALUES (...)

    # Get project by ID
    def get_project(project_id):
        SELECT * FROM projects WHERE id = project_id

    ... Any other functions required...

    # Delete a project
    def delete_project(project_id):
        DELETE FROM projects WHERE id = project_id
```

Then, when the user is working with the Create Project page, the createProject() function at the frontend will take all the user's input, send it as a JSON to the "/api/projects" endpoint, which in turn will call the backend function create_project(). This function then runs a relevant method for a Projects class to create the user's project in the project table. Finally, Flask sends a response which includes a dictionary with the new project's information. Later, when the project dashboard is updated, for example, React will pick up the new data as well and update the frontend for the user with their new project. Similar exchange of data occurs at the rest of the endpoints.

The backend functionality was verified through a series of tests. The database helper classes were tested independently by simulating database interactions, ensuring that data could be correctly read, written, and updated. The RESTful API routes were tested using mocked requests to verify that each endpoint correctly invoked the relevant class methods and returned the expected JSON responses. This testing approach confirmed that both the individual components and their integration worked as intended, and can also become a base for a more thorough testing of the whole application.

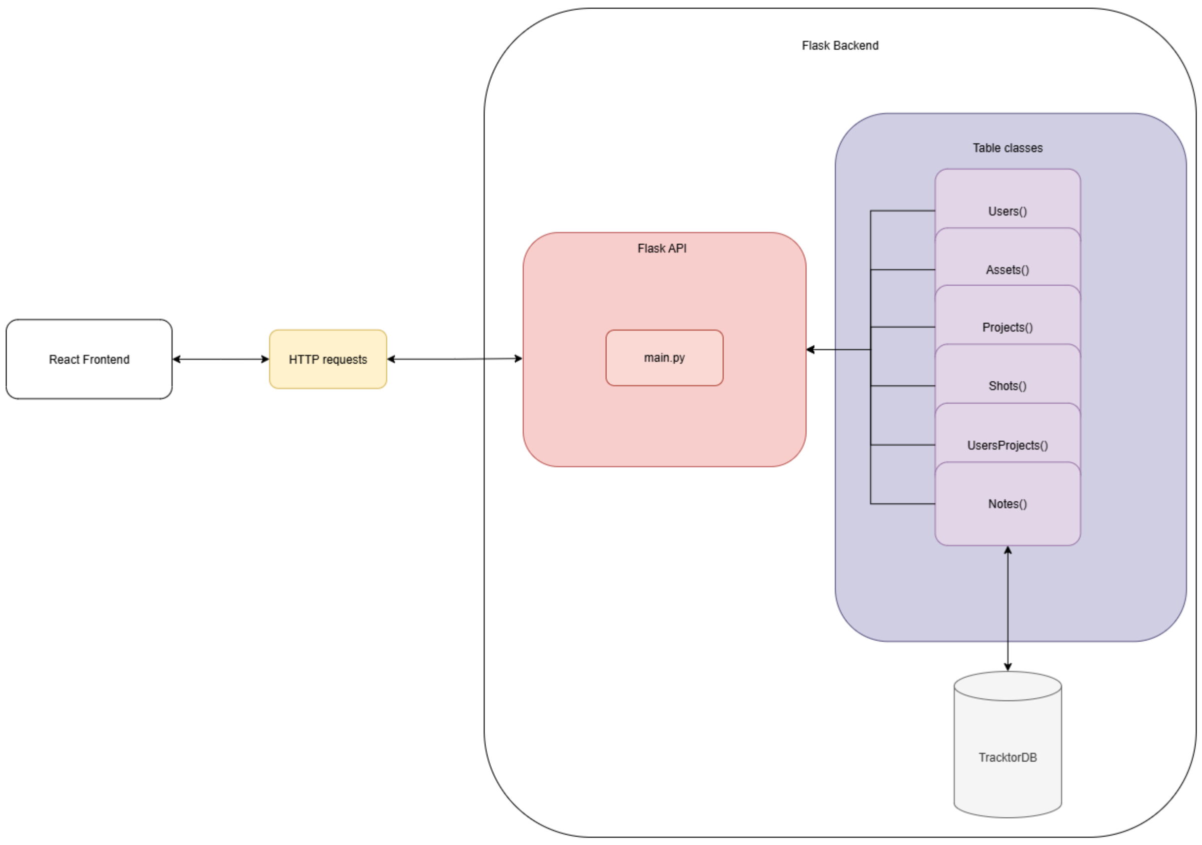


Figure 4: Flask backend design diagram

4.2.3 Database

The Tracktor database is a relational SQLite database that holds every entity (users, projects, etc) as a separate table. This matches the purpose of the application quite well, since it allows to enforce data integrity. All the necessary relationships between the tables are established by foreign keys. This structure ensures that data is organised, reduces redundancy, and maintains consistency across related records (Rob and Coronel, 2006, pp. 61-101).

Each table is wrapped by a corresponding helper class in the backend (table classes), which abstracts SQL queries and, when imported, opens these methods to the REST-

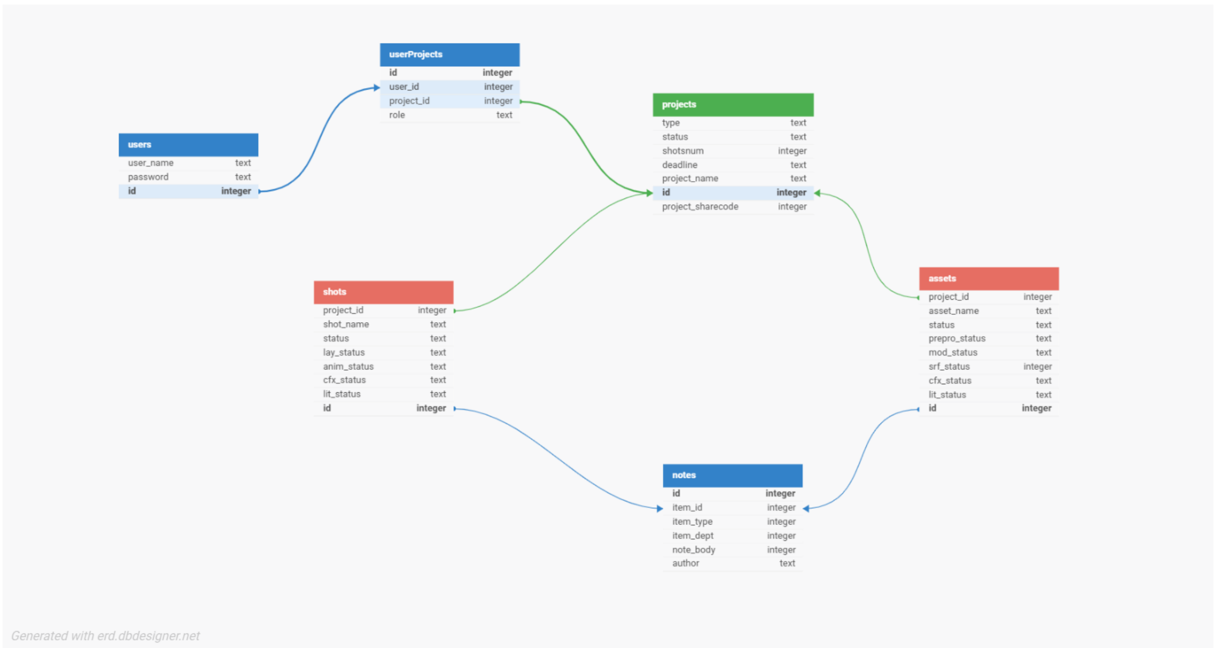


Figure 5: Tracktor SQLite database diagram

ful API endpoints. Such design allows the backend to interact with the database in a structured way while keeping the API interface consistent for the frontend and any other clients.

In some cases, the application needed to process two similar tables at the same time (e.g. assets and shots). To support this, a generic "item" abstraction was implemented in code itself to switch between enquiring from assets to shots table by an itemType. This approach simplifies integration with the frontend and works well for the current scope. However, for larger deployment, as a unified items table might be useful, in case there rises a need for more complicated queries.

For the same scope reasons, SQLite was chosen as the database for Tracktor due to its simplicity and minimal setup requirements. Unlike server-based databases such as MySQL or PostgreSQL, SQLite stores the entire database in a single file on disk, eliminating the need to install or configure a separate database server. SQLite does lack support for simultaneous writing, however, the scale of the app does not require this kind of complexity yet. With the growth of Tracktor, so can grow the database design.

This database diagram illustrates the relationships between the tables in the Tracktor database, as well as shows the foreign keys and overall data structure of the app.

4.3 Deployment

The full-scale deployment for Tracktor is planned on the server with both frontend and backend wrapped in the Podman containers. However, for the proof of concept demonstration, a local distribution was made available. It is still based on both Podman containers, which the user (or tester/reviewer in this case) can build and run with one Podman command from the main Tracktor's directory. This is not as artist-friendly as a conventional deployment on the web, but it is a good first step for any demonstrations.

The long-term vision is therefore to provide a hosted deployment. In such a model, the user experience would reduce to navigating to a URL, registering an account, and immediately creating projects. This would eliminate the technical requirements of self-hosting and lower the barrier to entry considerably.

4.4 Available functionality

So far, Trackor VFX is a locally hosted web app, which enables users to register their credentials and then immediately proceed to creating projects to track, which will be associated with the credentials, unless shared.

The creation of the project also includes some the basic metadata, which at a later stage will be wrapped in further functionality.

At the creation stage, a number of shots is defined, which will be generated automatically. Those shots form a core functionality by having a department status available to track each department. The user is then able to add more shots or add assets. Tracktor does not have a function to delete shots for data preservation purposes, so an Omitted status is implemented in its place.

Then it is up to the user to change the status of the items according to where their production is. They can also add notes to a specific department of a specific item, for example, an animation note for shot_0010, or a rig note for Ball_RIG asset.

To handle multiple users, every project can be shared by generating a project code, which then can be used to assign a new user to the said project. Then, changes will be available and visible for everyone. Simultaneous changes might be restricted, due to SQLite constraints, but for the small scale Tracktor is intended for this might be just enough.

4.4.1 TIK Manager Integration

Tracktor integration for TIK Manager is based on TIK's existing integrations with Autodesk Flow and Kitsu (TikWorks, 2023)).

The integration mainly consists of the UI extension, which provides a new menu in TIK and is a Python Qt file, and a ProductionPlatform class, which is an instance of ManagementCore class, that handles the integration, provided that the required methods are implemented. Additional files include tracktor_API, to map the TIK client requests to the Tracktor backend, and a login.py, which helps TIK to authenticate the user in the external platform and run its own requests.

The functionality is simple at this stage - the user is able to authenticate and pull one of their projects from Tracktor, by recreating it in TIK automatically. Later, if there was any change to the number or naming of shots/assets in Tracktor - those can be synced with the TIK. There is, of course, more that can be done, but it very much depends on the expansion of the Tracktor's functionality itself.

The data flow was chosen to be one-sided - this will aid the production consistency and prevent accidental changes or overwrites. Tracktor is the only true status log, which can be updated into the TIK, but it is ultimately Tracktor that has priority. For example, production managers can create new shots/tasks only in Tracktor and not in TIK (if they do so in TIK, it will not sync to Tracktor), making every change conscious and deliberate.

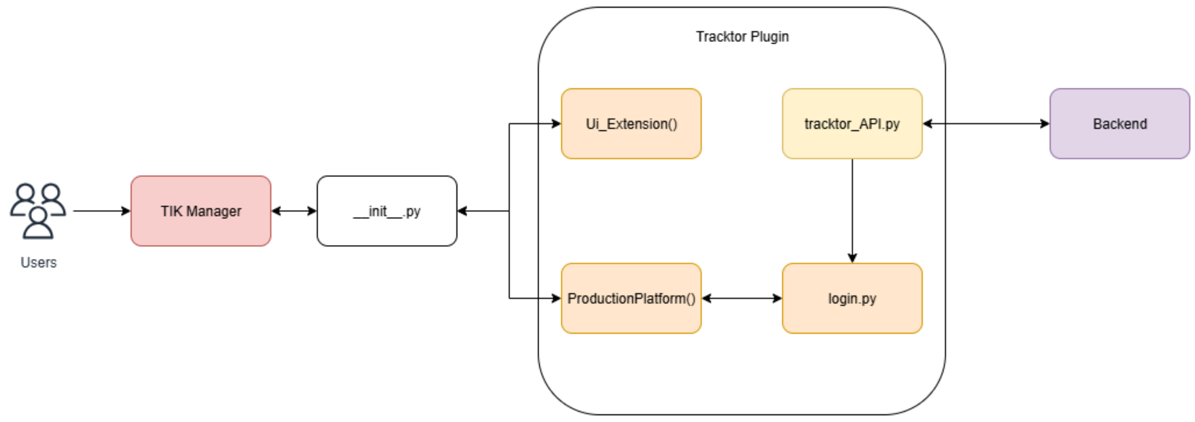


Figure 6: Tracktor plugin for TIK Manager design diagram

5 Conclusion

Overall, Tracktor met the defined MVP criteria and is a decent example of a first-time web development project. It has all the defined basic functionalities of a production tracking software and can already be used for that purpose. It is available for a test run for anyone curious via GitHub. Users can already create projects and manage the shots/assets inside them, mainly by tracking their status, and can also leave notes to themselves and other users assigned to the project.

Given the scope of the project, a lot of time was spent on familiarising with React and web development practices, which might not have been implemented to the best standard or principles due to lack of experience and a pressing time frame. Tracktor does what it is expected to do, which is a plus, but it doesn't mean that it does it in the most efficient way, and finding this balance can definitely be an area of improvement.

Still, there is great potential for expanding Tracktor's features. The current build only provides basic functionality, which could be improved in its own right - including user permissions, ability to edit notes, useful metadata, etc, as well as integrating completely new functions, which are common in the production trackers, like versioning, task assignment or review capabilities.

Similar conclusions can be made about the UI. Though the UI was not the focus of the project, with functionality coming first, it is still paramount to give users a pleasant and intuitive experience when engaging with a tracking software, when it goes live. Those improvements can include both aesthetic changes that can make Tracktor look more pleasant and familiar to the audience, as well as functional improvements that will make using it more intuitive.

Due to the project's time constraints and available resources, Tracktor has not yet been fully deployed online. Hosting plans exist, and the system is all but done for the deployment, since the same two Podman containers are going to appear on the server. They just need to be put there, and the server configured.

All of those features can make the product even stronger and prepare it for wider release on the web, which is all part of the roadmap for future work.

To sum up, Tracktor has a good start and can grow into a notable production management platform, but at this stage requires polish and a few more features to become available for a wide release.

References

Atlassian (2025), 'Jira product discovery', `https://www.atlassian.com/software/jira/product-discovery`. [Accessed: 21 August 2025].

Autodesk (2025), 'Autodesk shotgrid', `https://www.autodesk.com/products/shotgrid/overview`. Accessed: 2025-08-21.

AYON (2025), 'Ayon pricing', `https://ynput.io/ayon/pricing/`. [Accessed: 21 August 2025].

CG-Wire (2025), 'Kitsu production tracking', `https://www.cg-wire.com/kitsu.html`. [Accessed: 2025-08-21].

Doglio, F. (2018), *REST API Development with Node.js*, Apress.

Fenollosa, A. (2022), *Building SPAs with Django and HTML over the wire : learn to build real-time single page applications with Python*, Birmingham : Packt Publishing, Limited.

Fowler, S. and Stanwick, V. (2004), *Web Application Design Handbook: Best Practices for Web-Based Software*, Burlington : Elsevier.

ftrack (2025), 'ftrack studio', `https://www.ftrack.com/studio`. Accessed: 2025-08-21.

Karchevskaia, A. (2025*a*), 'Design document', `https://github.com/alesiakarch/VFX_ProdTracker/blob/main/Design_Doc.md`. Accessed: 21 August 2025.

Karchevskaia, A. (2025*b*), 'Requirements document', `https://github.com/alesiakarch/VFX_ProdTracker/blob/main/Requirements_Doc.md`. Accessed: 21 August 2025.

Microsoft (2025), 'What is software as a service (saas)?', `https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-saas`. [Accessed: 21 August 2025].

Palette (2025), 'Palette — prodcution management for digital artists', `https://www.palette.tools/`. [Accessed: 21 August 2025].

Rob, P. and Coronel, C. (2006), *Database systems : design, implementation, and management*, Cambridge, Mass. : Thomson Course Technology.

Rousseau, F. (2022), 'Our new responsibilities', `https://blog.cg-wire.com/our-new-responsibilities/`. [Accessed: 21 August 2025].

Silver Monkey Studio (2025), ‘Open-source tools in animation and vfx: Embracing collaboration’, `https://silvermonkey.studio/open-source-tools-animation-vfx/`. [Accessed: 21 August 2025].

TikWorks (2023), ‘Welcome to tik manager4 documentation!’, `https://tik-manager4.readthedocs.io/en/latest/`. Accessed: 21 August 2025.

TikWorks (2025), ‘Tik manager’, `https://tik-manager.com/`. Accessed: 21 August 2025.

WMTips (2025), ‘Technologies, project management’, `https://www.wmtips.com/technologies/project-management/?utm_source=chatgpt.com`. [Accessed: 21 August 2025].

Yadav, S. C. and Singh, S. K. (2009), *An introduction to client/server computing*, New Delhi : New Age International.

Appendix

A Design Document

Direct link to the requirements document.
Requierments Document

B Design Document

Direct link to the design document:
Design Document

C Tracktor Roadmap

Below is a non-exhaustive roadmap for Tracktor VFX. It aims to improve existing features and user experience, as well as add new functionality.

- **User System:** Add roles, permissions, and authentication tokens
- **Project Management:** Editable notes, versioning, task assignment
- **Technical Improvements:** Move to server-based SQL when necessary, implement media uploads (thumbnails, playblasts)
- **UI/UX:** Enhance navigation, responsive design, aesthetics
- **Deployment:** Configure and Host on a web server
- **Review functionality:** Media annotations, overlay of versions