



Instinto

Lince Documentation

Lince Version: 0.6.1

Typst Version: 0.14.2

Documentation Start: 2025-12-15

Documentation Print: 2026-02-05

Days of Construction: 52

Source: [Lince](#) (@lince-social/lince)¹

¹“<https://github.com/lince-social/lince>”

Contents

1.	Birth	4
1.1.	Summary	5
1.2.	Installation	6
1.3.	Disclaimers	6
1.4.	Organization	7
1.5.	Tech Stack	7
2.	Aging	9
2.1.	Record	10
2.2.	Collection	11
2.3.	Views	12
2.4.	Karma	13
2.5.	Examples	16
2.6.	Configuration	19
2.7.	Transfer Proposals	20
2.8.	History	20
2.9.	Dna	21
3.	Sickness	22
3.1.	Roadmap	23
3.2.	Good practices	23
4.	The Death of Lince	25
4.1.	Let's Dream A Little	26
4.2.	Serious Mode	27
4.3.	Why Lince?	27
5.	Tasks	28

1. Birth

It's a Lince!

— Docta

1.1. Summary

Lince is a tool for registry, interconnection and automation of Needs and Contributions with open scope. The phrase above is the densest way to explain Lince. The current page serves as an overview, and the subsequent chapters cover it in much more detail. Let's start with the registry part:

Registry

Users will start their Lince application for the first time, and that will create a personal database (for UNIX systems [Linux and macOS] it is located at ‘./config/lince/lince.db’).

That database is a Sqlite file, that is referred by the application as a DNA. That file can be altered to make the app behave differently. The behavior can be of: setting personal/professional tasks, modeling personal items, scheduling computer actions, act out economic trades between parties, and many more, the sky, your imagination and computing power is the limit.

Any app behavior mentioned above can be boiled down to a Need, or a Contribution to a Need. There are many ways to use the application for that end, one of them is the following: the differentiator of need or contribution is a Quantity. If such quantity is negative, that will be a Need. Having -2 Apples means you need to go and get two apples to be neutral, to satiate your needs. The same applies to a habit, having a -1 Quantity in Exercise means you need to exercise. The previous description is the basis of the whole app, everything revolves around it.

Automation

If there is predictability in your Needs, and a fixed frequency is enough to describe them, then a simple automation can be constructed. One can set the automatic every day change of the ‘Studying’ habit’s Quantity to -1. And with another feature, if that Quantity is -1 a ‘Command’ is sent to the computer that closes all programs that are not related to it.

Another point of automation is the forementioned DNA. One aspect of constructing systems is that many great ideas often are lost with time, memory and the passing away of those contributors. Common patterns and knowledge about production and maintenance of commodities, economic trades, access to resources and ways of living is scattered.

These patterns could be saved into some form of database, that when fed into a program like Lince, will create Needs and Contributions, so users can have a headstart in any endeavor. From production according to a bill-of-materials to morning routines and parties organization, these activities create Needs for gathering resources and acting out tasks.

Contributions to Lince DNAs are like creating blueprints for others to get up and running. They might not be a perfect fit, but allow for great customization. Currently the way of centralizing DNAs is the [Github Repository](#).

Interconnection

With great customizability and automation of people’s DNAs, the next barrier is the connection of those individual apps, turning each machine into a p2p node, also called a

1. Birth

Lince Cell. The same way you create triggers and rules for the quantites of your DNA, you can do so for others.

If one Market has 200 apples and you need 5, you can create a Transfer Proposal. The Quantity of money diminishes on your side, and the Apple one increases. On the Market side, a manual price can be inputed and accepted when proposals come, or it can be adaptive. Donations of clothes, and borrowing tools for quick use would be similar exchanges, just without the money part.

1.2. Installation

You can run Lince as an executable, here are the [releases](#). Pick the latest one for your machine and operating system. After the download, unzip it and run the binary:

```
./lince karma gui
```

Or you can install from the crates.io repository with:

```
cargo install lince
```

If you prefer to compile the source code, the easiest way is to download the repo and run:

```
cargo run -- karma gui
```

If you don't have Cargo installed, you can install it with [Rustup](#). It comes with Rust and its toolchain. If cargo isn't in your PATH, add it to your shell configuration (~/.bashrc or ~/.zshrc) so you don't have to type the full path every time (i.e., ~/.cargo/bin/{program}). Run this in your terminal:

```
echo 'export PATH="$HOME/.cargo/bin:$PATH"' >> ~/.bashrc
echo 'export PATH="$HOME/.cargo/bin:$PATH"' >> ~/.zshrc
```

This uses [Typst](#) to generate the documentation. I recommend installing [mask](#) with cargo install mask, then running mask docs. If you are in Arch or MacOS systems it will automatically install typst and [tinymist](#) (web server), else it will probably not work automatically.

1.3. Disclaimers

All the repositories in the Lince ecosystem have the [GNU GPLv3](#) license. Lince is a non-profit project and crowdfunding is the source of development compensation:

[GitHub Sponsors](#) | [Patreon](#) | [Apoia.se](#)

Lince tries to facilitate and automate the connection between people and resources, by transforming needs and contributions into data. The gains and losses related to the interaction, such as transportation, production and services themselves, remain the responsibility and risk of the parties involved.

1.4. Organization

The workflow of contributing to the Lince ecosystem can be of simply making a Pull Request. But if one wants to have certainty of developing a wanted feature, the best way is to see the DNA of Lince programming tasks.

The online resource to communicate is the [Discord](#).

The git repositories hosted currently in the [GitHub organization](#) mean this:

- [lince](#): the app's code and this documentation.
- [lince-social.github.io](#): the website, at [lince.social](#).
- [dna](#): the databases for different Lince behaviors.
- [lingua](#): A future programming language Lince evaluates to execute karma, transactions and more.
- [github](#): general stuff, like photos, icons and information about GitHub sponsors.
- [den](#): git submodules for all the repos in one place.
- [rustart](#): A template to start a Rust project.

1.5. Tech Stack

Backend:

- Programming Language: [Rust](#)
- Database: [Sqlite](#) | Easiest to replicate the concept of DNA. Way too simple, way too reproduceable.
- Database Driver: [SQLx](#) | Used for raw and typesetted queries

Architecture:

- Application
 - Services for anything complex, business ruly, that calls repositories or manipulates data.
- Domain:
 - Entities:
 - Clean: the most accurate representation of the main drivers of the application. The source of truth of Lince's concepts.
 - Dirty: structure of data that comes in or goes out through the endpoints, or an alias for a data structure as a Type so it doesn't clutter the screen.
- Infrastructure:
 - Database:
 - Management: database connection, migrations and schema.
 - Repositories: traits and their implementations for database operations like get, update...
 - Utils: functions for quality of life, like logging.
 - Cross_cutting: dependency injection for a collection of many methods in different layers. Mostly for single database connection initiation.
- Presentation

1. Birth

- ▶ GPUI: GPU accelerated Rust dependency used by Zed.

(Some Rust) Learning Resources

1. [Rust Book](#): chapters 1–6 will give you a simple base if you just want to hack something together. Passing through the entire book, doing the examples and understanding them deeply, will set you up for great vibe coding.
2. [Rustlings](#): for when you are done with the book. You will complete exercises on a broad spectrum of the language's features.
3. [Rust for Rustaceans](#): for a deeper dive in the language, good practices and crate recommendations.
4. [This Week in Rust](#): a weekly newsletter about Rust projects, crates and language changes. Amazing for keeping up to date with the ecosystem, very useful.
5. Youtube Channels: [Jon Gjengset](#) | [No Boilerplate](#) | [The Rustagen](#) | [Tsoding](#) | [Code to the Moon](#) | [fasterthanlime](#) | [chris biscardi](#) | [Mike Code](#) | [Developer Voices](#)
6. Podcast: [Rustacean Station](#)
7. Discord, Matrix and many more online communities are very welcoming to newbies.

2. Aging

Ouch! My back!
— Sedentary Person, Modern

2. Aging

This chapter explains Lince's components in more detail. It is the worse chapter by far. The project is too much in an Alpha stage and the Schema of the Database changes too much and a definitive one hasn't been chosen (was Postgres at first, now it's SQLite). Some of the properties in here might be out of date or not implemented yet. Explanation of the tables *should* be mostly focused on the ideas, the solutions brought to a problem yet to be found.

2.1. Record

record
id
quantity
head
body
location
time

Lince is centered on the 'record' table, but like, according to the creator... like... what does he know?

A 'record' can be anything: an action, an item, a plan.

record	DATA TYPE
id	INTEGER
quantity	FLOAT
head	TEXT
body	TEXT
location	3D POINT (still thinking)

'id's are automatically generated.

'quantity' represents the availability of the record, if negative it is a Necessity, if positive, a Contribution, zero makes it.

'head' and 'body' are meant to be parts of a whole, where one can be used for a short summary and the other a description, or one has all the information and the other holds tags for filtering through views. With a pubsub protocol, one can send a short information of the record, in this case it can be the head, and put the rest in the body. Only those interested in the head will ask for the body of the record. That way the minimum amount of information is sent over the network, making it faster and stuff, I think.

'location' is an important information for interactions outside of computers (they exist, it's insane) or any other use you want to give it. 'time' is a property like location, not implemented yet. It could be a hard-coded property that maps to a concept in the real world. Not a generic one like quantity. It's probably better to just have multiple custom properties attached to a record, like cost of time, location, cost of money as data on the

2. Aging

user's DNA, not as code in Lince repo. If we do it in a generic way, each record can have many different properties Lince doesn't have to predict to fit many use cases. The question is, how to we interlink a record with those properties? Other records? Such properties need to have the effect of their names: negative Time in Record A needs to remove that cost of time from the user's Time Record's quantity?

record	DATA TYPE	USER INPUT
id	INTEGER	
quantity	FLOAT	-1
head	TEXT	Eat Apple
body	TEXT	
location	POINT	

So, for an example, imagine that you like apples and you want to create a task to eat it today.

You create a 'record', giving it '-1' to the 'quantity', for that action is a Necessity in your life right now, and 'Eat Apple' to the 'head'.

—

record	DATA TYPE	USER INPUT	ACTUAL RECORD
id	INTEGER		1
quantity	FLOAT	-1	-1
head	TEXT	Eat Apple	Eat Apple
body	TEXT		NULL
location	POINT		NULL

The end result, on the database, is this record.

—

Here is an example of different possible records for individual items and actions.

id	quantity	head	body	location
1	-1	Eat Apple	NULL	NULL
2	-1	Apple	Item, Food	NULL
3	0	Brush Teeth	Action, Hygiene	NULL
4	3	Toothbrush	Item, Hygiene	NULL
5	-1	Meditate	Action	NULL

2.2. Collection

COLUMNS	DATA TYPE
---------	-----------

2. Aging

id	INTEGER
name	TEXT
quantity	INTEGER

A Collection is the name of a list of Views. Only one Collection is active at a time, with quantity 1, the rest is 0.

2.3. Views

COLUMNS	DATA TYPE
id	INTEGER
name	TEXT
query	TEXT

Views are ways to select data, to see what records exist with filters or ordering... They can also be used to see what Collections, Karma, Configurations... exist.

A view will have a name, so it is human readable and a query to display different data.

The query is made with SQL allowing you to select what columns you want to see, filtered, ordered and much more, just the way you want it.

The ‘query’ of a view can also be a special code, like ‘karma_view’, in the HTML (Web Browser) frontend. It is a specially constructed html to interact with Karma in a more complete and sane way, to bring ergonomics to day-to-day Lince. Instead of returning an automatically constructed table with data from an SQL query, the Backend adds customized HTML components to the page sent to the user (normal frontend behavior).

Examples:

Let's grab the record table shown before.

id	quantity	head	body	location
1	-1	Eat Apple	NULL	NULL
2	-1	Apple	Item, Food	NULL
3	0	Brush Teeth	Action, Hygiene	NULL
4	3	Toothbrush	Item, Hygiene	NULL
5	-1	Meditate	Action	NULL

A view of things you need to buy may look like this:

COLUMN	VALUE
id	1
name	Buy

2. Aging

query	SELECT * FROM record WHERE LOWER(body) LIKE '%item%' AND quantity < 0
-------	---

And in this case, will display only:

id	quantity	head	body	location
2	-1	Apple	Item, Food	NULL

Since no other records with a body containing 'Item' (lower, upper, pascal case...) exist with a negative quantity.

Collection View

COLUMNS	DATA TYPE
id	INTEGER
quantity	INTEGER
collection_id	INTEGER
view_id	INTEGER

A Collection View is an intermediate table for grouping Views into a Collection. The quantity is the order in which they appear and if positive they are shown, negative ones are not.

2.4. Karma

karma	DATA TYPE
id	INTEGER
quantity	INTEGER
name	TEXT
condition_id	INTEGER
operator	TEXT
consequence_id	INTEGER

Karma is a condition checker and an action taker. A constructor of if/then, behavior, inside your DNA. It does it by selecting data or running commands, then if the data fits some criteria, changing records, running commands and more. It replaces a syntax that references parts of Lince with its actual values, then evaluating it as a mathematical equation and according to the Operator. If the operator is '=' it lets only non zero values take effect, if it is '='* it doesn't have that constraint.

This is the hardest thing to understand in Lince and also the most useful. The full description, plus examples, will take place a little further ahead. This process is called a Delivery and it is run every 60 seconds, and sometimes instantly when wanted.

Karma Condition

2. Aging

COLUMN NAME	DATA TYPE
id	INTEGER
quantity	INTEGER
name	TEXT
condition	TEXT

A Karma Condition is something checked by replacing parts of the string (text) with real values. A record with id 1 has a quantity of 5. When we set a Karma Condition to be 'rq1' we are saying the value evaluated will be 5 (at that Delivery).

Karma Consequence

COLUMN NAME	DATA TYPE
id	INTEGER
quantity	INTEGER
name	TEXT
consequence	TEXT

The Karma Consequence works the same way as Condition but instead of getting values it is responsible for setting what is supposed to change. It can be the activation of a Shell/SQL command, the changing of the value of a Record and more; in the future it would be cool to make transactions, changing Quantities of other tables like Configuration and Collection Views, maybe even altering other Karma, or changing DNAs.

Frequency

frequency	DATA TYPE
id	INTEGER
quantity	INTEGER
day_week	INTEGER
months	INTEGER
days	INTEGER
seconds	INTEGER
next_date	STRING
finish_date	STRING
catch_up_sum	INTEGER

The frequency table holds account of a fixed period for returning a value of 1 or more. If a frequency occurs every 1 day and 60 seconds it might start, for example at 2030-01-01 10:00:00. When that time comes, if this Frequency exists in a Karma Condition it will be checked and updated, setting next_date to 2030-01-02 10:01:00.

The catch_up_sum is a multiplier, if many days have passed since the last check like 4, instead of returning one each minute, it catches up to the present in 'next_date' and returns the periods passed or only one.

2. Aging

In Karma, Frequencies are represented by the letter ‘f’. So setting in a Condition ‘-1 * f1’ means Karma Delivery will check the returned number for the specific Frequency with Id 1 and multiply that by ‘-1’.

Command

command	DATA TYPE
id	INTEGER
quantity	INTEGER
name	TEXT
command	TEXT

The Command is a Shell command you can run in a bash Shell.

Example:

id	quantity	command
1		touch grass.el

It is referenced in Karma Condition and/or Consequence as the letter ‘c’, followed by the id number, so this example would be ‘c1’. The command above creates the file grass.el.

Query

COLUMN NAME	DATA TYPE
id	INTEGER
quantity	INTEGER
query	TEXT

The Query is an SQL command you can run and affect your current DNA.

Example:

```
[“id”, “quantity”, “query”], [“1”, “”, “Robert”); DROP TABLE users; -”], ]
```

It is referenced in Karma Condition and/or Consequence as the characters ‘sql’, followed by the id number, so this example would be ‘sql1’.

Sum

sum	DATA TYPE
id	INTEGER
quantity	INTEGER
record_id	INTEGER
interval_relative	BOOL
interval_length	INTERVAL
sum_mode	INTEGER

2. Aging

end_lag	INTERVAL
end_date	TIMESTAMP

This table is responsible for returning a sum of change. Change of a record's quantity over time. You have the 'record_id' to know what record to look for. 'interval_length' is a period, can be 1 day, 8 months, etc. If 'interval_relative' is TRUE, will sum all the changes ending now, starting at the past based on the 'interval_length' (ex: 'interval_length' of 8 months and 'interval_relative' is TRUE will make it look at the past 8 months). If FALSE an 'end_date' will need to be supplied. so the past 8 months ending in a certain date. 'end_lag' will give a space between now and the end point of a relative interval. So if the 'interval_length' is 8 months and 'interval_relative' is TRUE, an 'end_lag' of 2 months will make the sum look at the 2 months back to 10 months back, starting 10 months ago, ending 2 months ago. 'sum_mode' tells Lince what to sum, if 0 it's a delta, essentially doing a quantity now - quantity earlier. If it's negative, it sums all negative changes, positive, idem.

2.5. Examples

So you now know the parts of Karma. It's time to put everything together. Lets take this example:

Record

id	quantity	head	body
1	0	Exercise	

Frequency

id	quantity	days	next_date
1		1	2030-01-01 10:00:00

Condition

id	quantity	name	condition
1		Daily	-1 * f1

Consequence

id	quantity	name	consequence
1		Exercise (Automatically set)	rq1

Karma

id	quantity	name	condition_id	operator	consequence_id
1		Daily Exercise (Automatically set)	1	=	1

2. Aging

With this data, every Karma Delivery (60 seconds) will check the validity of each condition. Frequency with id 1 'f1' will return 0 in all minutes of the day safe for one time, where it will return 1. Since the operator of the Karma with id 1 is '=' the expression in it's Condition ' $-1 * f1$ ' will almost always be ' $-1 * 0$ ' and will not have a consequence.

In the Karma delivery that is past the next_date of Frequency, the next_date is updated and the Condition is ' $-1 * 1$ '. Since that is equal to ' -1 ' the Consequence 'rq1' will change the quantity of Record with Id 1, turning it into ' -1 '.

We go from this:

id	quantity	head	body
1	0	Exercise	

To this:

id	quantity	head	body
1	-1	Exercise	

One could make Conditions like ' $(rq1 - 1) * f1$ ' and a Consequence of 'rq2' so the number would not be fixated at -1 , but would decrease by one every day based on rq1. The expressions can become very complicated, but unless you are doing finance, joining multiple sources of data into two or three records like monthly costs/gains and runaway you will do simple expressions.

Example: Command

With the command table

Let's say you have a command:

id	quantity	name	command
3	1	Radarada	touch grass.el

This command with ID 3 can be referenced in a Karma Consequence with 'c3'.

Consequence

id	quantity	name	consequence
2		Radarada (Automatically set)	c3

We can reuse the Frequency, and Condition, creating just a Consequence and Karma to automate the running of this command.

Frequency

id	quantity	days	next_date
1		1	2030-01-01 10:00:00

Condition

2. Aging

id	quantity	name	condition
1		Daily	-1 * f1

Karma

id	quantity	name	condition_id	operator	consequence_id
1		Daily Radarada (Automatically set)	1	=	2

This will daily run the Command ‘touch grass.el’.

—

Alternatively, you can make it so the Command is part of the Condition.

```
echo $(find ~/books/technology -type f | wc -l)
```

This command will output a number, which we can use inside a Condition.

Command

id	quantity	name	command
4		Tech Books Count	echo \$(find /books/technology -type f \ wc -l)
5		Open Current Tech Book	pdfrreader /books/technology/current/*

Condition

id	quantity	name	condition
3		Read Tech Task	-1 * f1 * c4

Consequence

id	quantity	name	consequence
3		Open Current Tech Book	c5

Karma

id	quantity	name	condition_id	operator	consequence_id
1		Daily Radarada (Automatically set)	3	=	3

That will automatically open the current technology book one is reading. If the automatic opening is not ideal one can set it so the Consequence is changing the quantity of a Record:

id	quantity	head	body
1	-1	Read Tech Book	

When typing 1 in operation the quantity of this Record will be zero. If you have a Condition that is ‘rq1 == 0’ that will be evaluated to ‘1’ and can trigger the Consequence

2. Aging

'c5'. Though this method needs one Karma expression to run 'c5' if 'rq1 == 0' and another to make it so if 'rq1 == 0' Condition then 'rq1' Consequence, making 'rq1' be '1' and not triggering the infinite opening of the Tech Book. In simpler terms: you set two different Karma, one checks if 'Read Book' has quantity 0, if yes, make it 1. The other Karma opens the book if the quantity of 'Read Book' is zero. So when we set it through zero it will open the book once, for we have a rule that quickly moves it to not zero.

2.6. Configuration

configuration	DATA TYPE
id	INTEGER
quantity	INTEGER
language	TEXT
timezone	TEXT
style	TEXT

The Configuration table alters some behavior of the Lince application. Language and Timezone are what they appear, and style is for setting what colorscheme is used.

Operations

There is a keyboard way of navigating through Lince, changing records and running commands. That is done through the Operations. When one types anything in the frontend, an input will appear.

If you type '4c', a modal for creating a Record will pop up. Typing 'a' with a number, like 'a2' will make Configuration with id 2 be active, and others inactive, an easy way to change colorscheme. One handy one is 's' with a number, 's1' to run a Shell command.

Just typing a number, like '23' will make the Record with id 23 have its quantity become zero.

Writing 'k3' will change the active collection to be the one with Id 3, effectively changing screens.

Its possible to click in the collection to change it, and setting the quantity to zero by manually editing the field in the table. But those operations are very frequent and can be more ergonomic for those that like to remember shortcuts. Those that dont will have a hard time currently :(

[#] Name	[Key] Action
[0] Configuration	[c] Create
[1] Collection	[q] SQL Query
[2] View	[k] Change Collection
[3] collection_View	[s] Shell Command
[4] Record	[a] Activate Configuration

2. Aging

[5] Karma_Condition	
[6] Karma_Consequence	
[7] Karma	
[8] Command	
[9] Frequency	
[10] Sum	
[11] History	
[12] DNA	
[13] Transfer	

2.7. Transfer Proposals

transfer	DATA TYPE
id	INTEGER
records_received	JSON
records_contributed	JSON
agreement	JSON
agreement_time	TIMESTAMP
transfer_confirmation	JSON
transfer_time	TIMESTAMP

Transfer Proposals are a way of exchanging record quantities between different DNAs. It is thought to be used for executing economic exchanges like buying goods and services, donations, organizing events, work tasks...

‘records_received’ is a collection of records and their quantities that will interact with our records, things you will receive. ‘records_contributed’ are the records you will contribute and their quantities, to the records of other parties, can be more than one party. So you receive 1 in your ‘Apple’ Record’s quantity, and give out 5 moneys to the other party involved.

‘agreement’ is a collection of agreement by all parties involved. ‘agreement_time’ is the moment every party agreed for the conditions of the trade, who will receive what.

‘transfer_confirmation’ is also a collection but with a confirmation from all parties that the transfer was successful, and ‘transfer_time’ for saving the event’s moment.

2.8. History

> This is from the old Postgres Schema and idea of History, it is kept here whilst it is in TODO.

2. Aging

history	DATA TYPE
id	INTEGER
record_id	INTEGER
change_time	STRING
old_quantity	FLOAT
new_quantity	FLOAT

History is a table that automatically logs the change of a record's quantity, to use the 'sum' table. It is possible to do a history of Karma too, that would be useful probably.

2.9. Dna

In practice, each DNA is a different database. In essence, a different Lince behavior. Be it by swapping databases the app is pointing to or aggregating data from multiple databases. It is the data the program has access to understand how to function, what karma rules are applied to what records.

Cough

3. Sickness

'Morbus est medicus naturae': Disease is the doctor of nature,
illness forces correction

— ChatGPT (circa 2025)

3. Sickness

Lince is not great, it always needs to get better, until it dies, then it doesn't. The way to do it is by giving it medicine (teaching people how to use it) while we cure it (making it simpler and featureful). The tasks and community to improve are found in the Discord server. But this document also has some of the...

3.1. Roadmap

- v1.0.0: Todo
 - Rewrite of Frontend in GPUI
 - Todo
 - Table
 - Kanban
 - Calendar
 - Shows Records changing with Karma. If they have a time cost, it occupies time from the calendar.
 - Finance
 - Table
 - Graph
 - Calendar
 - Connection
 - CRUD of cells (your node) and organs (group of nodes).
 - Public/private rows for what organ (group of cells).
 - Transaction of quantities between cells (nodes) in p2p network.
- v1.1.0: AI
 - Be able to run an AI model to look at your DNA and change it to fit your needs.
- v1.2.0: Stock
 - Screens to help with stock management for small to big companies.

For detailed tasks, go to the end of this book.

3.2. Good practices

Making a Lince DNA, modeling items and actions to perform is a continuous effort, if it makes sense to you. With that said, here are some ideas to get started:

– Create a view or more with all the data and configuration for managing your DNA, so Configurations, Collections, Views...

Karma

Until Lince has Deterministic Simulation Testing (DST), which in this context is a way to grab a DNA and simulate years of usage, you have to be mindfull of the Command table you produce, every command you set may possibly break your system if you don't tidy things up. If you have logic of running a command every hour if one record has quantity >

3. Sickness

X and you forget about it, any simple change will trigger it, so put guardrails for running things, be it Commands, Queries or even changing Record Quantities. Changes might cascade and deliver unforeseen consequences.

With DST this is easier to do, the plan is to have a containerized environment that runs a simulation of your system, isolated from the outside world. Your DNA (your personal configuration of Lince) is a seed that can be run multiple times arriving at the same result (hopefully). Being able to run it with a high speed, changing the date (affecting the Frequency table) and record quantities will bring reproducible results. When you want to add something to your DNA you can check it's effects with a simulation and get info if it breaks anything in an edge case.

Command

Lince works with Sqlite files for it's database. It is recommended to frequently backup your DNAs, weekly, daily or hourly if you are paranoid. If some error or mistake happens, your information is safe.

Personal Tasks – Create Views for understanding your tasks, select the ones that have negative quantity, check this SQL:

```
SELECT * FROM record WHERE quantity < 0 AND LOWER(head) LIKE '%task%'
```

– One can use the ‘head’ column in ‘record’ table as a tag holder. So Record’s heads with Items and Tasks that have negative quantities appear in my ‘Negatives’ view.

Finance Currently not very featureful. You can model in Records your monthly earnings and expenditures and make a Karma with a condition of (Records of Earnings) - (Records of Expenditures)

Live

4. The Death of Lince

Death is only the beginning
— The Mummy (1999)

4.1. Let's Dream A Little

The quote of “The Mummy (1999): Death is only the beginning” makes total sense with the following performative explanations on The Death of Lince. Building subsets of the perfect functionality of this app is repetitive and makes most Software Engineering a bullshit job if the vision is correct (already is even if it’s wrong).

Once the operational part of life is modeled, automated and interconnected, the effects of having built a robust and generic base for interaction with systems will start to show. If there is no incentive to rebuild a management system/platform (because there is a clearly G.O.A.T.ed one), more efforts will have to be focused on new highly specific innovative and optimizing ideas. People will enjoy having previously centralized, private and paywalled capabilities in the palm of their hands or in the chips of their brains if you are into that stuff. If apps are islands and people are like jet-skis then the current landscape is like an archipelago, Lince aims to create a Pangea.

Think of Amazon and Uber, they are companies that get a cut, to orchestrate something that computationally a global peer-to-peer network personal devices could easily manage. Think of how much your much time your devices spend processing your online purchases and how much it could be plugged into the LNHM (Lince-Net-Hive-Mind).

Amazon is a platform for Records of item contributions, your purchases are Transfer Proposals. The rides you make on Uber are Transfer Proposals of money for transportation from point A to B. Any automation they make inside their platforms is limited to the data they have on you, which is a minuscule amount of your whole life (at least I hope it is). If you were to set up hacky ways to use those platforms automatically to fit your needs, you would be doing it to every app you have, connecting them in ways they were not meant to, possibly programming something to organize all those changing external apps. It doesn't need to be that inefficient, you can have one central program that you can trust to work offline for you, acting just the way you want, featureful, built by many people, for it is the main one. The model of Lince can be a superset of many apps, and can spark many different markets and support people with its usage.

Lince is actually not the app you install, that's a Lince Cell. When you have a group of Cells (or p2p nodes) you have a Lince Organ; people interacting to meet their needs (or others'). Lince is actually when all Cells and Organs work together. It's only a complete animal when independent of platform, language, nationality [TODO: insert the rest of ‘imagine all the people’ by john lenon’s lyrics here]... or football team, people are working together like the image below:

4. The Death of Lince

A world where film
freelancers don't have to
worry about a lack of work:



Figure 2: Society if freelancers had a lot of work: Green utopia with a gal petting a tiger
careful it might bite

4.2. Serious Mode

Like every good contribution, the goal is to not have to contribute anymore; because it was a one time need, or the frequent need has been solved.

Lince is a contribution to the need of having Lince, of having all production possibly connected, so there is predictability in resource usage and logistics coordination.

It's a contribution to the disorganized construction of solutions glued by the internet, centralized in platforms that worsen by the day.

It's a contribution to understanding the whole of possibilities of how you can connect with other people, after seeing needs you didn't know others had, sparking ideas of contributions you didn't know you could make.

The death of Lince is the death of the need of Lince. If there is ever any configuration of resources that makes Lince useless, then its job is fulfilled, there is nothing else to be done, no commit, no push, just vibes. Until then, a system for self organization that can turn into a dance of the world is to the writer something useful and exciting to build.

A team is being assembled, to fight the battle of killing Lince. Will you join the cult?

4.3. Why Lince?

Here is a list how Lince might help the needs of different people to emotionally manipulate you into helping build it, feel free to add yours for the pyramid scheme:

– **Eduardo de Melo Xavier:** I want Lince to be productive and help others with basic needs. After that I want to register resources to meditate.

// TODO

5. Tasks

Don't hallucinate while you implement the following features...

— Not me

Work In Progress

(5 tasks)

Frontend

(2)

Table component*Contributors: @chicogborba [WIP](#)*

I think this is the best way, sending the column size information to the db... but it's kinda weird? Table needs to have these properties:

- Custom column width. If a table has too many columns we need to decide if we:
 - (A) reduce that columns' size to fit the screen (hard) (B) make the central area with the tables be like a 2d canvas with vertical/horizontal scrollable space (maybe even zoom). If that's the case a (return to content is cool but for now ok not having).
 - Column Resize: saves information on individual columns' size.

For each collection, every view that is a table should be able to have each column with a custom width.

- Have word-wrapping by default, in the cells, being able to toggle it in Configuration.
- Any changes made to the db have all fit inside one or more migrations.

Editable cell*Contributors: @xaviduds [WIP](#)*

Each cell when hovered should:

- Have an icon to edit it in a resizeable text buffer.
- Be editable in place if clicked, maintaining the same size,

but with a big text field, instead of just a text cell.

Bureaucracy

(1)

Lince Institute

Contributors: @xaviduds [WIP](#)

- Consolidate the 'Ata de Fundação' and 'Estatuto Social'
- Use the 'Ata de Fundação' and 'Estatuto Social' to create the 'Associação privada' called 'Instituto Lince'
- CNPJ (Cora (more famous) or Conta Simples)
- Bank Account (which bank? Banco do Brasil?)

In Brazil there is a model of legal organization called Associação privada with a non-profit declaration. It has the best legal model for Lince in the several years to come. It's main purpose is to be a legal body in case the project needs to be represented or have a relationship with other parties. These relationships include donations, contracts with developers, legal operational reports and more.

The legal category of the organization is an 'Associação privada'. The plan is for the public legal names 'Razão Social' and 'Nome Fantasia' to be 'Insituto Lince'. And to be commonly known when reffering to the project as Lince and have the institute as a supporting piece.

The following documents explain all the papers needed for the entire lifecycle of the Instituto Lince, the legal entity in Brazil to operate and represent the bureaucracy of the project.

Ata de Fundação Document used to officially create the organization, needs at minimum the founder Eduardo another associate (they dont need to be involved in the project after the creation).

Estatuto Social The rules and permissions the project has. The rights and duties it proposes to follow.

Procuração Document to be updated and adapted whenever a new legal/financial task needs to be complete, and it's not done by the Executive Director.

Design

(1)

New Logo Items: First Batch

Contributors: Nika [WIP](#), @xaviduds [TODO](#)

- New vетorized logo
- Hering's Super Cotton for t-shirts
- Stickers: get tip from Nika
- 3D Keychain Items: @tecnopuc_crialab

5. Tasks

Karma

(1)

Frequency

Contributors: @DiogoTeixeiraDEV WIP

Frequency has two features to be done so we can complete it:

Catch Up Sum: When a frequency hasn't been activated for a long time, like for a 1 Day frequency with a next_date stuck three months ago, if something references it, every Karma Delivery (60s) will update it to one day closer to tomorrow (2 months 29 days now). The catch_up_sum is something that takes all of the possible times the frequency would activate and moves the next_date until it reaches stability.

catch_up_sum == 0 => dont do anything, just calculate frequency normally one time.
catch_up_sum => positive, make the next_date jump the number of times the value of catch_up_sum, never jumping if next_date is already in the future. In other words: if its 1, its the same as zero, you jump the next_date one time based on the frequency (1 day) and go on. If it's two, you jump two times so it would go from 3 months ago to 2 months and 28 days. If its negative dont do anything.

Days of the Week: There already is a commented try at this in the frequency function. The goal is to make something easy to write to say that it should fall in a day of the week. So if the frequency only contains info about jumping every monday and tuesday then the day_of_week would be something like 1, 2 or 12 or something else, you who knows. If the frequency is months: 2, day_of_week: 5 it will first jump to the next friday, then jump two months. Or maybe it should first go to the two months and then fall on a friday. There should be a mechanism to easily set a preference between the two behaviors.

Feel free to refactor this a lot. With those two unfinished parts the frequency will be able to cover many cases, if you have more periodicities in mind to cover even more cases please refactor.

TODO

(13 tasks)

Frontend

(5)

I can have a view that is the Creation Modal of any table

Contributors: N1 *TODO*

One of the Views of any collection is the creation modal, now a View. So the person will see tables and next to it the component used to create a new record, so they will be able to create new records easily. This is good for todo behavior, being able to pin the view of record creation means a quick todo creation

Collection's Views CRUD

Contributors: N1 *TODO*

Be able to:

- Update view's name, query
- Add views
- Remove Views
- Create Views

Be able to Pin Views

Contributors: N1 *TODO*

I can pin a View of Collection A when I am in collection B, making it appear on the screen with higher Z index and stuck to a place

Table Extra Ergonomics

Contributors: N1 *TODO*

- Delete row (with confirmation, by default, with option in configuration to not ask)
- Sortable (On runtime, updates should respect this sort, even though data comes with another sort or unsorted)

5. Tasks

Canvas

Contributors: N1 *TODO*

Make an expansive 2d canvas, views dragging to adjust the position. When im in any collection, i can drag and drop my views, resize them, and their positions and

Design

(1)

Find Social Media Manager

Contributors: N1 *TODO*

Train them into using Typst to automate it, make simple content that delivers only important news and concepts with mininum words, connecting it with trying out Lince.

Youtube (this_month_in_lince)

Insta (social_media_posts)

TikTok

Website?? (Blogpost)

Karma

(7)

Karma Refactor

Contributors: N1 *TODO*

Karma Conditions poderem ser referenciadas em outras conditions tipo kd2 + kd6. Garantir que seja possível ter cadeias infinitas de condições: karma: kd2 = kd6 = ks2

Extensions

Contributors: N1 *TODO*

Be able to receive information about ESP and put it into a Karma condition.

5. Tasks

Note-taking-like

Contributors: N1 [TODO](#)

Ter uma forma boa de editar notas, conectando possivelmente notas de objetivos com karma, pra que cada workflow tenha sua justificativa e possa-se criar primeiro os objetivos e completar eles com os passos pra chegar lá

IA de recomendação de Karma

Contributors: N1 [TODO](#)

fazer otimizações balanceamento de atividades ao longo da semana pra nao sobrecarregar um dia. Sugerir habitos novos...

Execute single Karma with visualization on the condition being evaluated

Contributors: N1 [TODO](#)

Maybe a modal

syntax highlighting and lsp (TreeSitter?) for Commands

Contributors: N1 [TODO](#)

Being able to see based on the language syntax highlighting. So if in a Command block there is not a language set default to bash, if there is rust use the highlight for Rust, use lsp to see if its wrong, be able to run every command and see the result.

Deterministic Simulation Testing

Contributors: @xaviduds *TODO*, @DiogoTeixeiraDEV *TODO*

DST is amazing! The idea (I think) is to have three things:

1. The Seed: the user's DNA (la ele).
2. The Rules: What events should be bookmarked or stop the simulation?
3. The Engine: How will this simulation happen? With the normal flow of time, or a tampered one? Connecting to the outside world with Commands?

This way we can create futures shown to the user so they can see to the end of their Karma and catch bugs or unintended behavior. This is useful in financial simulation, or for understanding the costs of time for doing tasks (like the Calendar feature).

With DST we may duplicate the DNA to change it freely without affecting the user's data, or perhaps not changing persistent data at all, just manipulating data inside the program.

TigerBeetle is the GOATED db for this, perhaps Lince can learn from it, fork it, or use it with a different schema for Transaction of Records. <https://www.youtube.com/watch?v=JoYjji1DZCE>