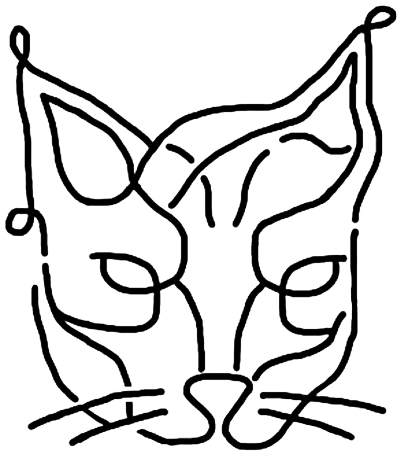

Instinto

Lince Documentation

Lince Version: 0.6.1

Typst Version: 0.14.2

Documentation Start: 2025-12-15

Documentation Print: 2025-12-18

Days of Construction: 3

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

The one that first wrote this is a functional illiterate and a broken poet, the phrasing needs improvement much more yes.

[1] "https://github.com/lince-social/lince"

Contents

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 reffered 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 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
```

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
```

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.
- dna: the databases for different Lince behaviors.
- rustart: A template to start a Rust project.
- lingua: A future programming language Lince evaluates to execute karma, transactions and more.
- .github: general stuff, like photos, icons and information about GitHub sponsors.

1.5. Tech Stack

Backend:
- Programming Language: Rust
- Server: Axum
- Database: Sqlite | Easiest to replicate the concept of DNA. But way to simple...
- Database Driver: SQLx | Fun async, previously was Rusqlite but didn't prototype that well.

Frontend:
- Web:
  - Base: HTML, CSS
  - Templating (Backend): Maud
  - Framework: HTMX + Datastar (transitioning into)
- GPUI:
  - Experimenting

Dirty 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 doesnt clutter the screen.
- Infrastructure:
  - Http:
    - Routers: the paths and arguments of endpoints.
    - Handlers: the functions that are called by endpoints, that receive the query params, path variables, header, body if wanted.

    Responsible for calling the services and/or returning presentation functions (html frontend).
  - 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
  - Html: the templates made with maud, or sometimes just big strings. that are returned from functions. These HTML may contain HTMX and/or Datastar (in the process of removing HTMX). The directories are divided rudimentary by concepts/entities like 'operation', 'table', 'pages'.
  - GPUI: for a featureful and immersive GUI, the future for the frontend, currently in implementation.

(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 recomendations.

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

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 attatched to a record, like cost of time, location, cost of money as data on the 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
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
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 it's 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 doesnt 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

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.

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
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

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, joinning 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

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	pdfreader /books/techology/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 '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 it’s quantity become zero.

Writting ‘k3’ will change the active collection to be the one with Id 3, effectivelly 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
[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.

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)

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.

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 > 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 recomended 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.

3.2.1. 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 optmizing 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 miniscule amount of your whole life (at least I hope it is). If you where 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 where 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 it's 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 meed 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 footbal team, people are working together like the image below:

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 didnt 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 it's job is fullfilled, there is nothing else to be done, no commit, no push, just vibes. Untill 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 piramid 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.