SaaF: Software as a Fact

Software should not pretend to be scarce when it is not.

Software should not turn the user's problem into rent.

Software should not be made worse so payment can make it better.

SaaF — Software as a Fact — is a constraint for building complete software.
It means the app exists to do its job, not to capture the user, extract from the user, or hold the workflow hostage.

A SaaF app is not a funnel with a tool attached.
It is the tool.

The Promise

If an app claims to help the user complete a workflow, the app must ship with the minimum functionality required to complete that workflow from start to finish.

No artificial caps.
No broken free tier.
No "almost useful."
No "upgrade to finish."
No paid removal of pain deliberately placed there.

If the feature is necessary for the promise, it belongs in the product.

No Artificial Scarcity

SaaF rejects limits that exist only to force payment.

No maximum photos unless the device cannot handle more.
No maximum pages unless the format or system requires it.
No maximum folders, colors, projects, exports, saves, retries, or uses unless the limit is real.

Real limits are allowed: hardware, operating system behavior, storage, memory, platform rules, legal constraints, safety constraints, and genuine service costs.

Fake limits are not.

Local First

A SaaF app should run on the user's device whenever reasonably possible.

If something can be done locally, it should not require an account, a cloud, a server, a subscription, or permission from a company.

Complexity is not an excuse to take custody of the user's work.

Cloud features may exist, but only when they provide something that cannot reasonably exist locally, or when the user explicitly wants them.

The local app must remain whole.

No Ads

No ads.

Ads turn attention into inventory.
SaaF does not.

No Hostage Features

No tiers that carve a complete tool into incomplete versions.

No paid undo.
No paid comfort.
No paid speed.
No paid batch action.
No paid relief.
No paid removal of artificial friction.

Friction may exist only when it serves the user: safety, clarity, pacing, control, comprehension, or better experience.

Friction must never exist as a payment lever.

Data Minimalism

A SaaF app collects the least data possible.

Ideally, none.

No unnecessary accounts.
No behavioral profiling.

No analytics by default.
No quiet appetite for personal information.

If data is not required for the function, the app should not ask for it.

Honest Payment

SaaF is not anti-money.

Good software deserves to be paid for.

But payment should support the software, not unlock the parts that make it whole.

The preferred model is simple: pay once, own the tool.

Pricing should be honest, finite, and proportional. In general, a SaaF app should aim to cost around the price of a single month of the SaaS competitor it replaces, though the exact price may vary by product.

Updates Are Not Toll Booths

If a new feature is only code, and it does not create ongoing external cost beyond normal development and maintenance, it should be an update.

A feature should not become IAP just because it is useful.

If it improves the promised workflow and can run locally, it belongs in the app.

Ongoing Fees Require Ongoing Costs

Subscriptions, passes, and recurring fees are allowed only for functions that create recurring external costs.

Servers.
Storage.
Hosted sync.
Hosted processing.
Global matchmaking.
Persistent worlds.
Live infrastructure.
Moderation.
Human support.
APIs that cost money to run.

A recurring fee must correspond to a recurring burden.

It must not be a rent machine attached to local functionality.

Server Features Must Be Optional

Any hosted service must sit outside the core local workflow.

The local app must remain complete without it.

A server may extend the app.
It may not become the spine of something that could have stayed local.

Purchased Things Stay Purchased

Anything bought outright should remain with the user.

Anything created, exported, unlocked, acquired, or produced during a paid period should remain available after that paid period ends, unless the purchase was clearly temporary access to an ongoing hosted service.

A subscription ending should not erase the user's work.

SaaF for Games

Games are allowed to be exciting.

Games are allowed to be sticky, surprising, difficult, rewarding, colorful, unfair, strange, tense, and full of "one more try."

SaaF does not remove fun.

SaaF removes the cash register from the wound.

No game loop may reduce to money.

When a normal game would ask for gems, coins, ads, paid continues, paid boosters, paid skips, paid revives, paid timers, or paid relief, a SaaF game must instead lead to gameplay.

More gameplay.
A consequence.
A recovery route.
A branch.
A remix.
A different mode.
A way to own the fail state.

Failure is part of games.
Failure must never become a payment prompt.

Lootboxes may exist because surprise is fun.
Lootboxes may not cost real money, premium currency, ad views, or any monetized proxy.

Randomness belongs inside the game.
Not inside the wallet.

Games that naturally create addictive “one more try” loops must include decompression through gameplay.
Not a lockout. Not a timer. Not a payment screen. A different rhythm of play.

The game may tempt the player to play again.
It must never tempt the player to pay again.

The Challenge

SaaF is a design constraint.

It asks:

Can this app compete without artificial scarcity?
Can this tool be complete without becoming a funnel?
Can this game be compelling without becoming a trap?
Can this software earn money without making itself worse first?

Anyone can monetize the pain point.

SaaF is the harder problem: solve the pain point, ship the whole tool, charge honestly, and still stand in the same arena as the extraction machines.

No ads.
No artificial limits.
No hostage features.
No subscription theater.
No fake scarcity.
No workflow ransom.

Complete by default.

That is the fact.