What is our brand's voice and tone

All Resend content should read as though written by a developer.

Here are some voice, tone, and language guidelines.

Simple

Get to the point and avoid unnecessary complexity, redundancy, and verbal garnish. Our content should not be thinly veiled sales pitches or lazy fluff for the sake of volume.

> Good: Resend is an email API for developers.

> Not good: Resend is a revolutionary platform that connects world-class busineses with their users via a state-of-the-art communications interface.

Factual

Use positive assertions that you can provide evidence for. Engineers are on alert for claims they can poke holes in, and once found, their trust in us is shattered.

> Good: Resend delivers emails under 600 ms (p95).

> Not good: Resend is the fastest email solution in the market.

Technical

We expect our audience to feel comfortable reading a fairly technical piece. Including technical details can not only be clarifying, but also make it clear that our content is not just an arm of the marketing function.

> Good: Resend provides two types of email encryption configuration such as Enforced TLS and Opportunistic TLS.

Not good: Resend gives you controls to make your emails safe and secure.

What are the most common use cases

Resend is designed for two specific use cases: transactional and marketing emails.

Here's how they differ from each other.

Transactional Emails

One-to-one emails that contain information that completes a transaction or process the recipient has started with you.

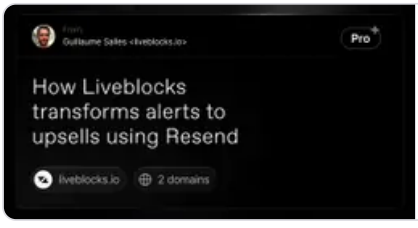
- Password reset emails
- Receipt and invoice emails
- Trial expiration emails
- User invitation emails
- Notification emails
- Welcome emails
- Double opt-in

Related customer stories:

How Anghami sends 3M emails every month with ...

The Resend team didn't feel like an external entity, but rat...

resend.com/customers/anghami



How Liveblocks transforms alerts to upsells using ...

We were pleasantly surprised that we didn't need to reach...

resend.com/customers/liveblocks

Marketing Emails

One-to-many emails that contains commercial and/or promotional content.

- Newsletters
- Changelogs
- Product announcements
- Promotions
- Updates to Terms of Service

Related customer stories:

How Nuxt built their deployment platform with Res...

I've used other email providers for years, and Resend's de...

resend.com/customers/nuxt

How Theya is sending launch week emails using R...

It has become an indispensable tool for starting and maint...

resend.com/customers/theya

How we think about swag

Creating company swag isn't just about visibility—it's about connection. It's a unique opportunity to extend the essence of our brand into items that both our team and customers will desire and use in their daily lives, not stash away in a drawer. Here are key considerations to help design the perfect swag.

Focus on the brand, not just the logo

Traditional swag often makes the mistake of being too logo-centric. **It's about the vibe, not just the logo**. Our swag should embody the brand's spirit—its tone, moments, and color schemes integrated subtly. This makes our merch feel less like an advertisement and more like a curated part of our community's lifestyle.

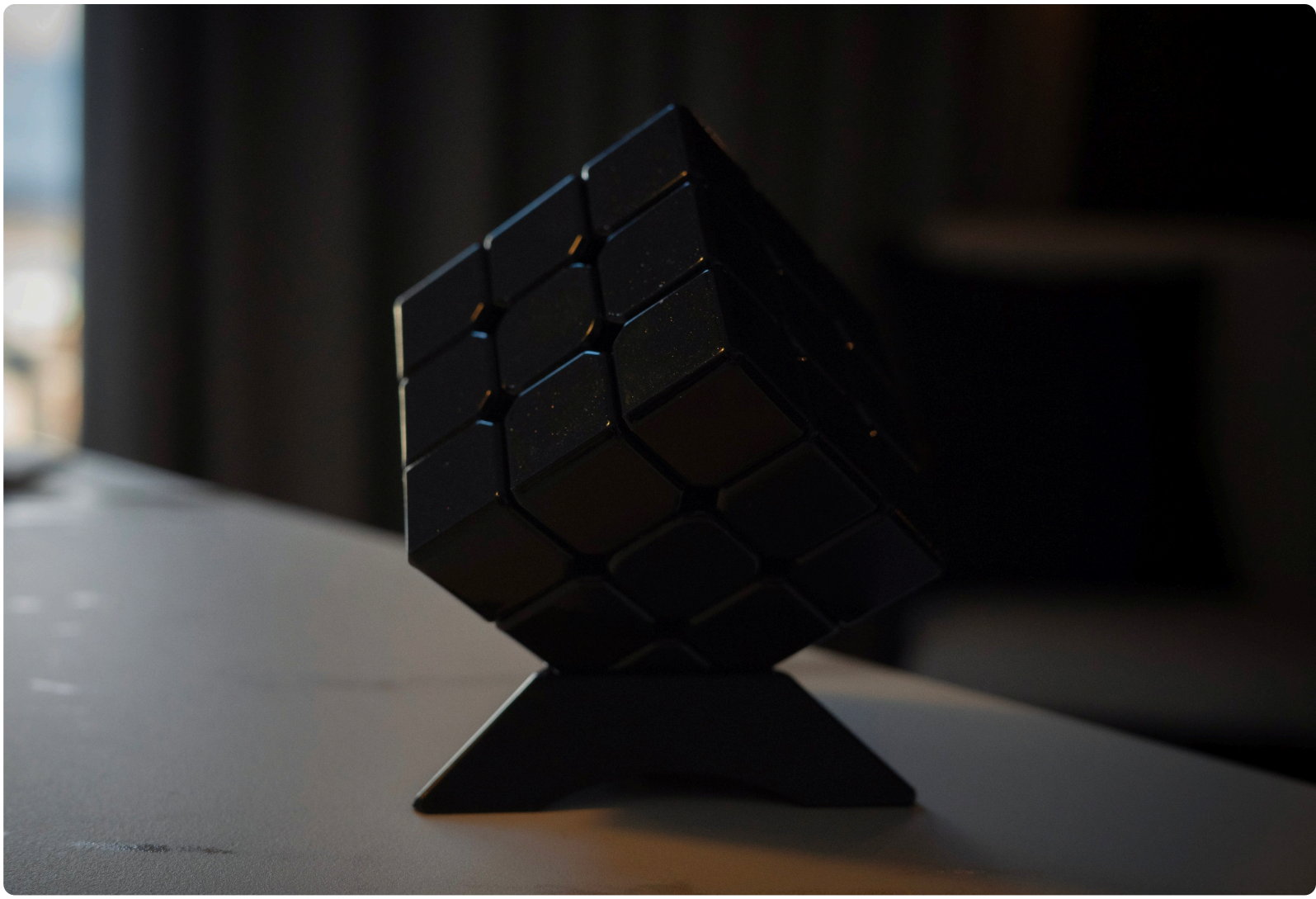


Photo from @tcosta_co

Quality matters

The choice of vendor says everything. Opt for those who align with our commitment to quality and aesthetics. This involves deep dives into reviews, sample evaluations, and vendor histories to ensure our **swag is both appealing and durable**.

Beyond the basics

Engage the audience with items that resonate with their lifestyles. **Innovation doesn't always mean new; it means thoughtful**. Think beyond standard corporate gifts—personalize, customize, play. The packaging is also super important, and it doesn't mean physical packaging.

Send it

Make it desirable

Great swag makes the receiver feel special and valued. When designed with care, these items not only promote the brand but also enhance the owner's pride in using/having them. This approach can lead to our swag being a talking point on social media or in casual street conversations.

How we write customer stories

Why

Stories let us share information in a way that creates human connection. Every customer has a story to share, and it's up to us to find the right angle to tell it.

There are two main reasons why we spend time writing customer stories:

1. Attract potential customers - There is no better selling point to prospective customers than having direct proof and real-world examples of how our product is helping other companies. This is particularly impactful if a customer story features a market leader, a trending product, a competitor, and or if it describes a challenge that others might be experiencing. The thought process here is: *"I admire company X, if company X is using Resend, we should use it too"*.

2. Retain existing customers - Customer stories are not only a way to attract new logos by association, but it also helps to turn customers into advocates making them more loyal in the long run. By describing their experience and documenting everything, it makes them reflect and articulate the value we provide.

Impact on the team

Apart from the external factors, there are also some internal benefits to writing these customers stories:

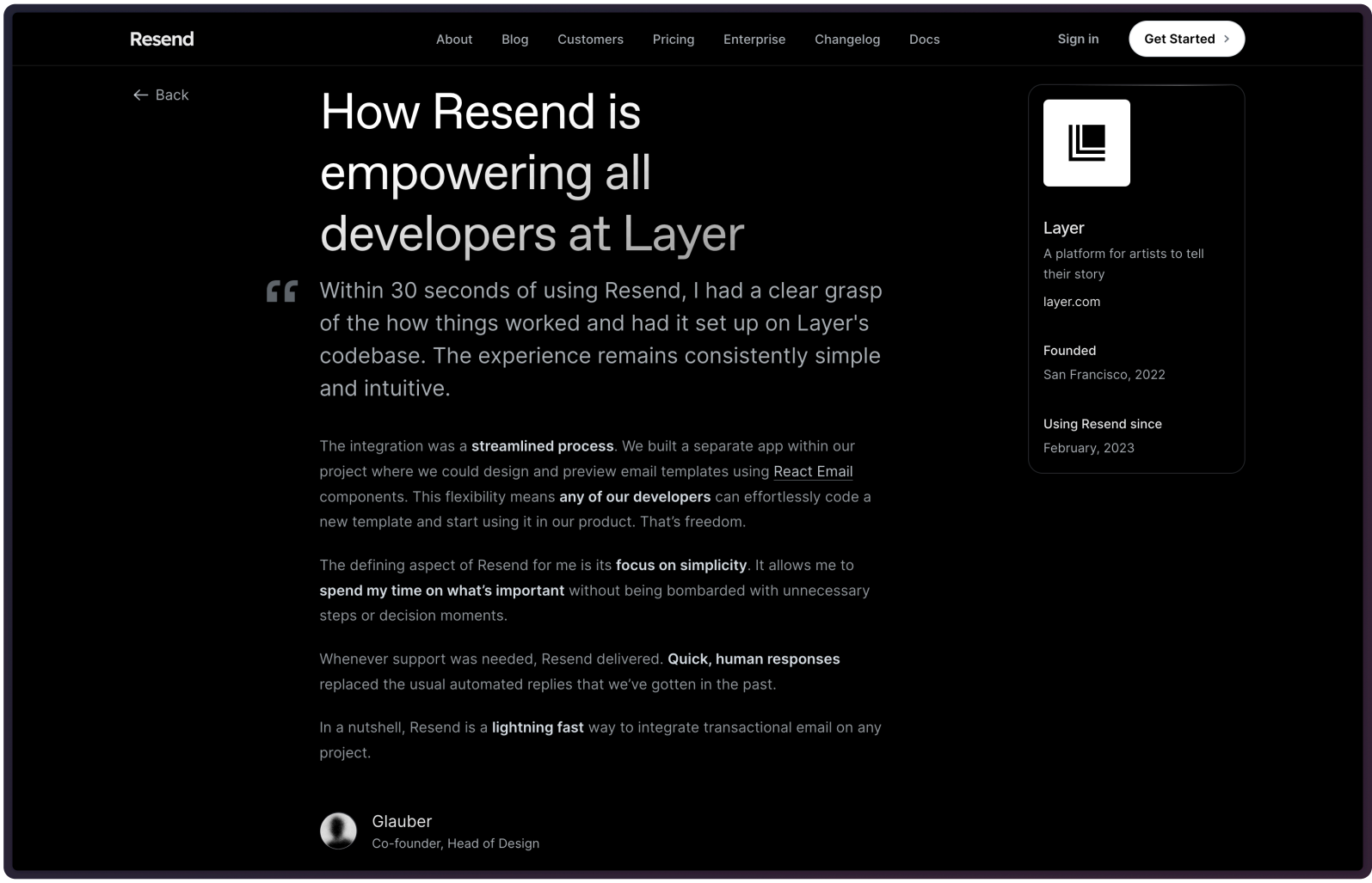
1. Better understand our key customers - By doing this exercise, it forces us to run a complete research and have full clarity of how key customers are using our products. We can generate insights from it and use that new understanding to make better product decisions.

2. Motivate the team - Having a nice logo using our product is a reason to be proud. Not every person in the team has the opportunity to interact with customers directly, so this a way for us to keep the momentum and have the team closer to who we are serving.

Format we use

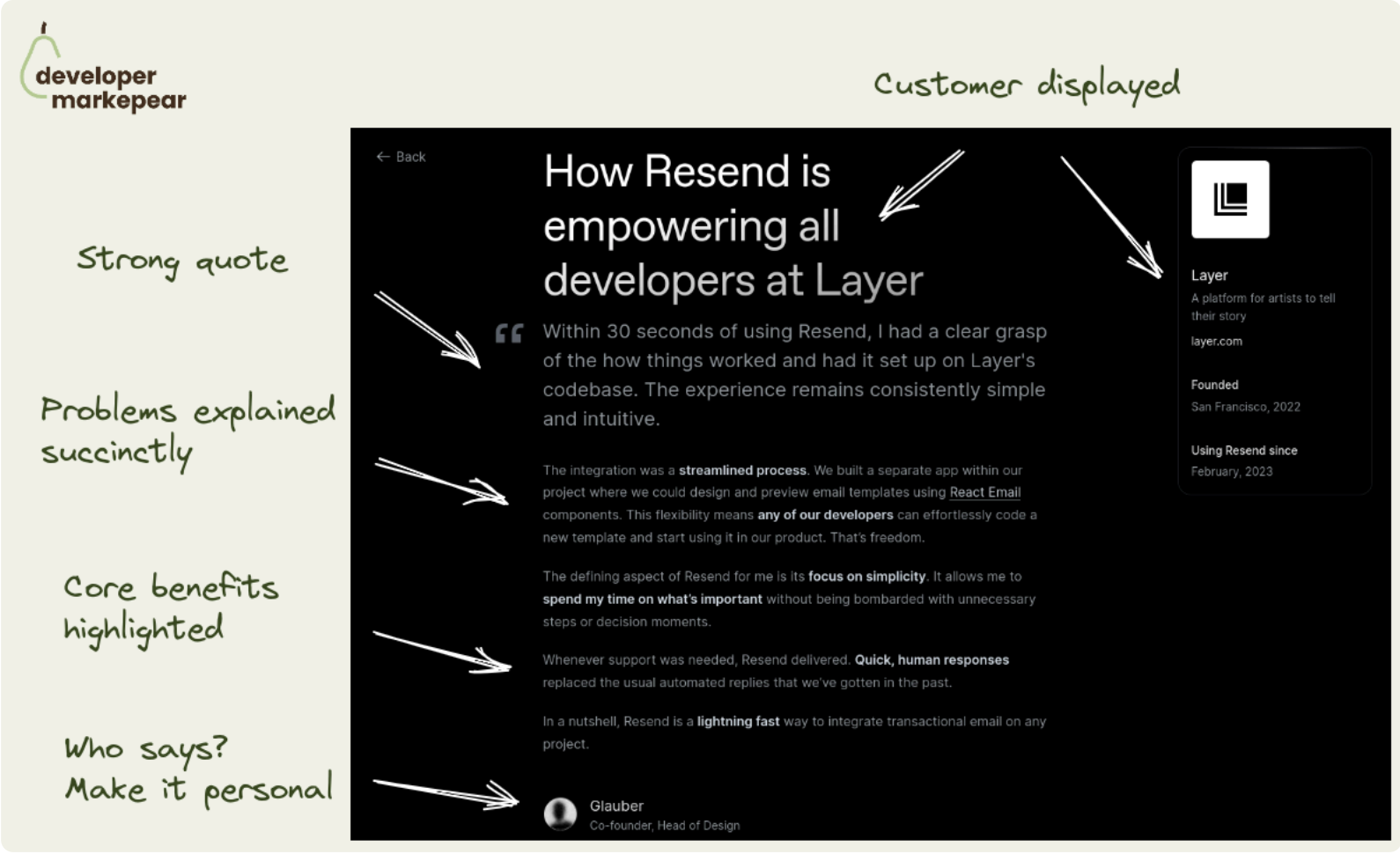
We believe that most developers don't have time to read a 5-page customer story. That's why we keep things simple and aim to fit everything on a single viewport.

- **Title**: it needs to be descriptive and shouldn't be longer than 3 lines.
- **Quote**: it needs to be powerful and get people's attention.
- **Content**: it needs to contain highlights of specific keywords.



An example of a Resend customer story

The fine folks from Developer Markepear dissected our format, so if you want to learn more we suggest reading this page.



The anatomy of a Resend customer story by Developer Markepear

Questions we ask

When approaching a customer to write a story, here are the questions we ask:

1. Tell us about your sending. Why are emails important to your business?
2. Walk us through the experience taking Resend into production.
3. What would you miss the most if you had to stop using Resend?
4. How was the help when you ran into problems?
5. Describe Resend to a friend in one sentence.

After they send the answers, we're responsible for crafting an engaging story.

Once the company approves the draft, we are ready to publish the customer story to all of our socials.

Publishing cadence

We aim to publish 1 customer story per week.

Not every customer should have a case study, but we should still follow an agenda and have a consistent approach to it.

How we think about our website

Storytelling

A fancy website with no substance is like a beautiful car with no engine.

We believe that a well-crafted narrative has the potential to transform a mundane URL into a memorable experience.

That's why we put a lot of effort into crafting a compelling story that can be told as you scroll down the page. This should be reflected not only on the home page, but in every page.

Just like a good book, a good website should have a beginning, a middle, and an end. What content to include is as important as where content should be placed.

Memorable and authentic

Many people ask us why we have a rotating Rubik's Cube on the home page.

We value technical excellence, we value design and craft, and we value being doing things differently. We could write down all of those things on a separate page that talks about our company mission. Or we could demonstrate those values in the most important real estate we have on the web.

Because we're in a highly competitive and extremely crowded market, it's crucial that we stand out. We do this in many ways, including our website.

> *“A great company must be able to impute its values from the first impression it makes.” - Mike Markkula, first angel investor for Apple*

Care and attention to detail

During our trip to Portugal, Jonni Lundy said he had an idea for the website, but it was too silly to share. We insisted, and he said: “What if we changed the title based on the time zone of the user?"

We all stopped and agreed that it was a fantastic idea.

This is the end result:

If you open the website from 9am until 12pm, you'll see the morning messaging, then the afternoon messaging, then the evening messaging. If you visit the website from Friday at 5pm until Sunday at 11:59pm, you'll see the weekend messaging.

If you pay close attention to the icon, you'll notice that the direction of the light changes based on the time of the day. For the morning icon, the light comes from the left, the light for the afternoon icon comes from the center, and for the evening icon, the light comes from the right.

Most people won't notice those details, but for the ones that do, it shows that we care.

Playful and interactive

Although it's possible to tell a fantastic story with only static text and nothing else, we believe that the more interactive elements we can add, the more rich the story can be.

> *“There are three responses to a piece of design. YES, NO and WOW! Wow is the one to aim for” - Milton Glaser, designer of the ‘I Love NY’ logo*

We avoid having a "wall-of-text" with large amounts of text presented without breaks or visual relief, such as paragraphs, bullet points, or images.

The web is a visual medium, and we want to take full advantage of that canvas. That's why we like to explore different animations, hover effects, and other interactive elements.

Cross-functional effort

When building a product, you need a multitude of skill sets. The same is true for creating a website.

As a small team, we won't have world-class domain experts for every skill needed, so working with contractors for specific projects is a pretty effective strategy.

As a reference, here are the people involved in the launch of our website.

- A 3D artist for the WebGL objects
- An industrial designer for the illustrations
- A brand designer for the Figma mockups
- A design-engineer for the implementation
- The CEO writing content and storytelling

The website is the sales team

We don't have a team of account executives who build and maintain customer relationships. What we do have is a product that **people can use and buy by themselves**.

When we first built our website, it took many months to ship and a *lot* of investment. There were times when we were skeptical that all that effort would actually be worth it.

After we launched, we realized the impact it had on sign-ups, and we understood that our website is actually our sales team.

How we approach marketing

When it comes to marketing, everything we do is about driving word-of-mouth growth.

The idea is to create a machine that **amplifies externally what is being built internally**.

On top of that, we want to be seen as experts in our field, so delivering genuinely useful tips to people is crucial.

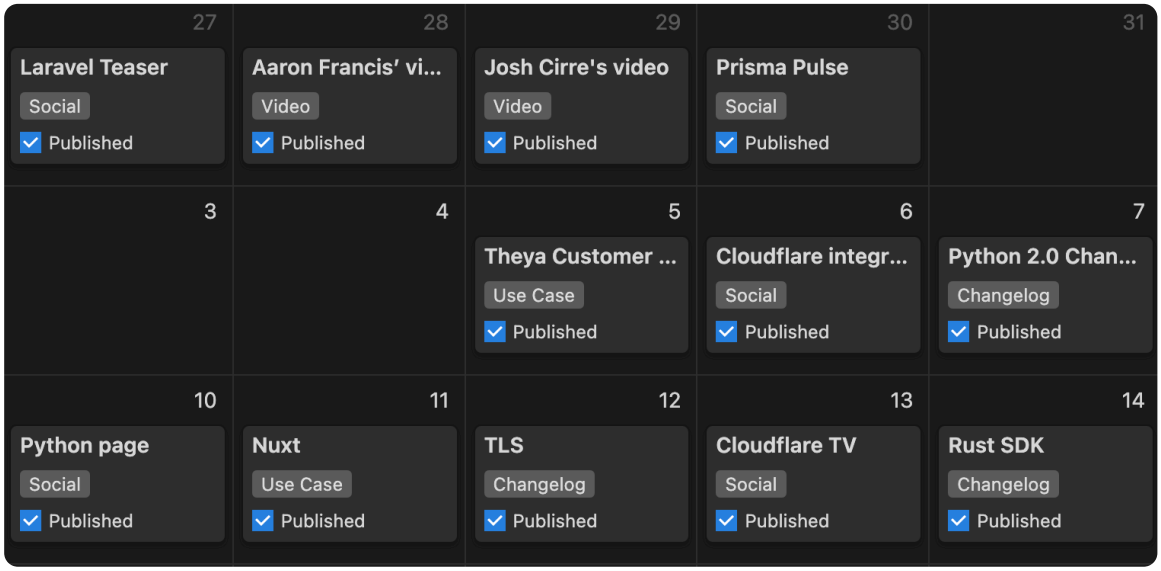
To do that, we use different communication formats and channels.

Social

We don't have the resources to be active on every social media platform like TikTok, Instagram, *<insert new trend here>*.

Instead we try to be **very active in only a few places**. We found that X and LinkedIn are the most effective places for us to be at this moment.

Our goal is to post every weekday, and we have a content calendar to help us with that.



Resend Content Calendar

We don't plan too much in advance, we mostly decide what are the things we want to post this week, and that's it. The **calendar is only a tool** to push us to meet that goal, but we don't let that get in the way.

The reason we post every day is because we want Resend to be top of mind for folks. Developers only **adopt a tool when they feel a pain**. They might not need Resend today, but when they have email problems two weeks from now, we want to be the first choice that pops on their minds.

Email

We recognize that not all people use social media, and even when they do, the **algorithm might not distribute our posts** to them.

That's why every month or so we try to send a newsletter to all of our users.

This is typically a compilation of the most recent product updates including new changelogs, blog posts, customer stories, and open source launches.

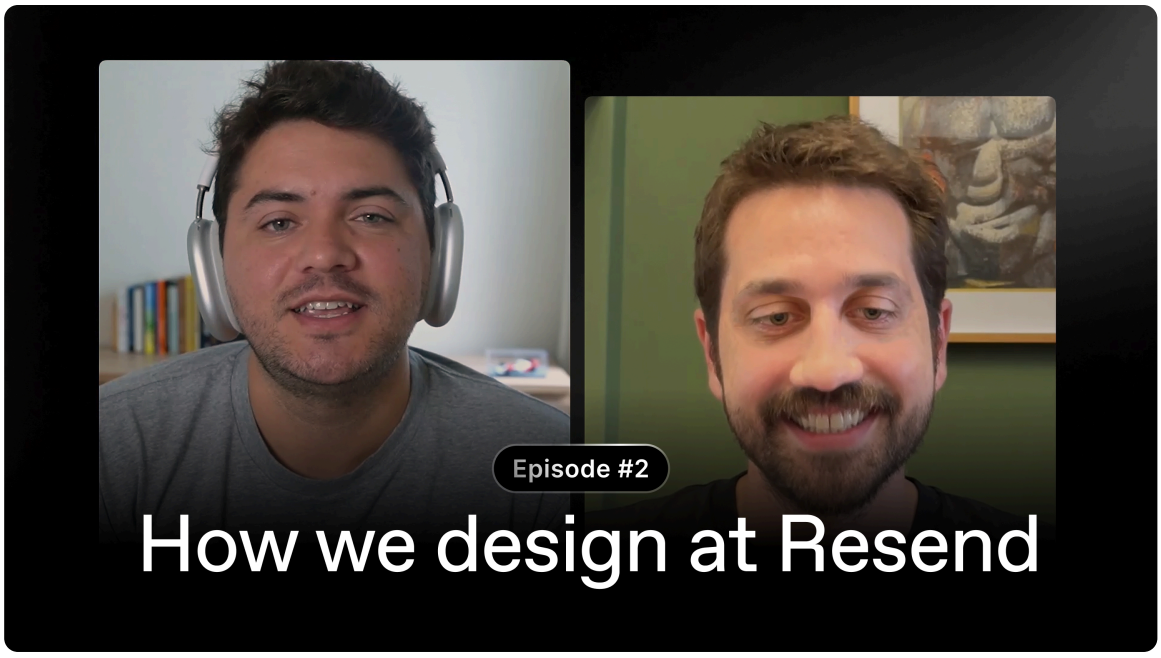
When we organize special events like launch weeks, we create unique audiences just for that specific event. When we do that, we can increase the frequency of messages such as daily emails during a launch week.

Dogfooding is a huge part of our culture, so we use Resend Broadcasts for that.

Video

Video is an experimental format for us. We know that YouTube is a different game, and to be relevant there you need to be extremely active and a little bit clickbait-y as well.

Sometimes we record how-to videos and publish a video series called Founder Q&A where we invite team members to answer questions about Resend.



Founder Q&A video series

We don't invest in video because we want to maximize the number of views. We do that because we believe video is a **great medium to tell engaging stories**.

Overall, we prefer having 120 views on a video that **engages with the right audience**, than 20,000 views on a video that reaches random people that dont't fit our ICP.

Blog

We haven't invested heavily in SEO yet. That's an area we want to explore, but haven't been able to find a consistent framework and build a sustainable machine to approach this.

Still, we try to publish something new on our blog at least once a month. These are typically company related news like new hires or fundraising.

What is our philosophy for anti-abuse

Assume positive intent

Most of our competitors require users to fill up a long form before you can start sending emails. Once you submit the form, you have to wait a couple days to receive a response if your account was approved or not.

This type of behavior communicates that they don't trust users, since you first need to prove that you're legit before you can experience the value of the product.

At Resend, we default to trust. We have a simple onboarding flow where users can start sending emails immediately, and add domains without waiting days for approval. We then monitor for any suspicious activity and take action only when necessary.

Questions to ask ourselves:

- Is this product friction going to affect all users or only abusers?
- Is this the right place to impose friction?
- How expensive it is for an abuser to overcome this friction? For example, getting a new IP is super easy and cheap by using a VPN service. However, getting a new domain is not as much.

Accept the reality that abuse will always happen

Stripe will always have to deal with fraudulent payments. X/Twitter will always have to deal with fake accounts. That's the nature of their business.

The same is true for Resend - the email industry has been a target for phishing and spamming since the beginning of time, and will continue to be like that.

We'll never have the luxury to invest 100% of our engineering and operations time combatting abusers, so we need to be strategic about how we approach this problem.

When brainstorming solutions to prevent abusers, make sure to think about the Pareto Principle.

Questions to ask ourselves:

- How big is this vulnerability?
- Is this solution going to prevent 10 phishing emails or 10k phishing emails?
- What is a good low hanging fruit that will make the biggest impact?

Understand that abusers will get more sophisticated

As we get more popular, more abusers will pay attention to Resend.

As our product grows, there will be more surface area for them to exploit.

The fact is that abusers will get more sophisticated over time, so it's important that we evolve our controls, tooling, and visibility at the same pace.

That's why it's crucial to provide weekly reports and visualize trends over time. Some anti-abuse projects will require company-wide efforts, so providing visibility to the entire team is very important.

Questions to ask ourselves:

- What are new patterns that abusers are using?
- Which TLDs are being used more often?
- Which geolocation are most abusers coming from?

Don't lock people out of their accounts

The only thing worse than blocking a legit user is to also lock them out of their account. It's okay to block specific features and slow down sending, but it's not ideal to automatically remove access from people's accounts.

Questions to ask ourselves:

- What are the alerts we can send to users when their account is marked as risky?
- What are the features that should be blocked?
- What error messages should users see when they interact with blocked features?

Blocking abuse is actually protecting safe senders

It's easy not to see the purpose of anti-abuse. Why spend so much time and resources on something that doesn't drive revenue? Similar to the police or any security detail, although most of the attention is on bad actors, the only reason they exist is because there is something to be protected. The goal is to create a safe space where senders don't have to “watch their shoulder” for risk or danger.

Questions to ask ourselves:

- Is this an abuser or just a sender that needs help?
- Are we seeing attacks and getting numb to their risk to good senders?
- How can we reuse anti-abuse mechanisms to also help good senders improve deliverability?

How we receive feedback

Receiving and acting on user feedback is key to our development process. Here's how we make sure that every piece of feedback is valued and utilized to improve our product.

Logging Feedback

We make sure that all user feedback is logged using Linear.

Every piece of feedback provided by users is logged into Linear and linked back to the original ticket. Tagging each feedback request helps us track how many people are asking for features in the product and also helps us follow up with the user for further questions on their use case and need for that feature. This process allows us to track the progress of each feedback item from submission to resolution.

Internal Advocacy

Feedback is a critical part of our team syncs.

During our weekly all-hands meetings, we report on the feedback received from the previous week, paying special attention to issues with a high volume of tickets. This transparency ensures that the entire team is aware of what users are saying and which areas require our immediate attention. By integrating user feedback into our engineering cycle, we prioritize improvements that matter most to our users.

Closing the Loop

Users are personally updated once their feedback is implemented.

Once changes and features are implemented, we ensure that we update the users who provided that feedback, let them know about the updates in detail, and invite feedback. This helps us let our users know that their feedback is invaluable and maintain a continuous dialogue on how to evolve Resend further to be better for their needs.

Our approach to receiving feedback is built around the belief that user insights are crucial to our product development process. By logging, discussing, and acting on user feedback, we foster a culture of continuous improvement, ensuring that our products and services evolve in ways that truly meet our users' needs and expectations.

How we evolve our knowledge base

Our knowledge base is a collection of answers to frequently asked questions, and solutions to commonly occurring issues. The goal for our knowledge base is to keep Resend as self-service as possible, so customers can get answers without reaching out to the Support team.

Documentation vs. knowledge base

Documentation is your dictionary; the knowledge base is your cheat sheet.

Simply put, the documentation tells you how to use the product. It's structured, comprehensive, and covers everything from set up, to advanced configurations. These include topics like creating a domain, creating an audience, or deleting a contact.

The knowledge base on the other hand is a lot more diverse, and contains articles and guides about frequently asked questions, recommendations on topics like deliverability, or if you should add an unsubscribe link to your emails.

Self-service

Skip the queue, solve it solo.

The goal for our knowledge base is to help customers get answers to questions without needing to reach out to support.

This means that the knowledge base should be easy to search for, and the articles should be easy to understand.

Up-to-date

Old answers lead to new problems.

It's crucial for knowledge base articles to be up-to-date, especially if the goal is to solve issues as fast as possible. Having outdated information increases the time-to-resolution for our customers, and decreases our efficiency as a team.

We use our support tickets as a guide to creating knowledge base articles. In a perfect world, we wouldn't have any of these, and they would all be solved in the product. However, having a knowledge base is a great way to bridge the gap while our team works on making the product answer the questions for you.

How we scale support

When support demand increases, the goal shouldn't be to immediately hire more people. This can become a temporary fix, leading to an ever-growing need for more hires as support volume rises.

The focus should be on making each hire as efficient as possible. This means listening to customers and building a product that meets their needs, reducing the necessity for them to reach out in the first place.

We think about support as a `try/catch` block.

```
try {
  product()
  documentation()
  knowledgeBase()
  codeExamples()
}
catch() {
  support()
```

```
}
```

There are all these things you do inside the `try` block, like documentation, knowledge base, and code examples. However, if the user still can't help themselves, it moves to the `catch` block.

Great support is thinking about how to improve the `try` block so it doesn't go to the `catch` block as often.

Making the product better

In a perfect world, the product should answer any question the customer has.

- How do I change my billing email? It should be easy to do this in the product.
- How can I remove a team member? It should be easy to do this in the product.
- How can I delete my account permanently? It should be easy to do this in the product.

But we don't live in a perfect world and sometimes the product doesn't fulfill all needs, so we need to add extra resources to help users.

Still, the goal should always be to make the product as self-explanatory as possible.

Create documentation dynamically

One of our favorite ways to resolve each ticket is by asking ourselves, *"How could I prevent the next customer from asking this same question?"*

Each support ticket should be actionable, whether that means creating a feature request, writing/updating a handbook article, or updating the product.

A tip that helped us achieve this is to block ourselves from moving on to the next ticket, until we write a knowledge base article and put it live.

This way, we now have a link to share not only with the customer who asked the question but also with future customers who might have the same question.

Support our support team

Customer Support is the face of your organization. Whenever someone has a question, complaint, or frustration, they're the first line of contact they have with your brand.

When customer support flags a concern, we take it with high priority.

This could mean adding tooling to make processes more efficient, or escalating a feature request because 3 people have been asking for it in the past week.

Remember, customer support pain = customer pain.

Track and log ticket frequency

This one is as simple as it reads.

The more often a feature request or bug comes up, the higher it should be prioritized. Logging frequency helps identify which features are the most asked and requested, which makes it easier to bring into current sprints.

We really like the way that Plain and Linear integrate with each other. It's extremely easy to link a support ticket with a Linear bug ticket or feature request.

How we help users

Our approach to helping users is based on three fundamental principles: understanding, responsiveness, and empowerment.

Understanding

We strive to deeply understand our users' needs and challenges.

Just as we put ourselves in each other's shoes when giving feedback, we make sure to take the time to understand our users' use cases and perspectives.

This means actively listening to their feedback, recognizing patterns in usage, and being empathetic towards their experiences.

By doing so, we can tailor our responses and solutions to their unique needs, ensuring that our help isn't just useful but truly resonates with them individually.

Responsiveness

We aim to be swift and efficient in addressing users' questions and issues.

Responsiveness is key to maintaining trust among our users.

Whether through customer support, social media, or communities like Discord, we want to ensure that users' voices are heard and their issues are addressed promptly.

Being accessible and eager to solve problems, we can fix immediate issues and show our commitment to their success.

Empowerment

We give users every resource they need to build and fix without opening a ticket.

Empowering our customer base means providing them with an extensive knowledge base, detailed API documentation, and various code examples.

Our goal is to make these materials easily accessible, ensuring our users can leverage them effectively to achieve their goals with email sending.

Additionally, we offer an easy way for users to submit feedback and feature requests, allowing us to improve the product continuously based on their input. We prioritize transparency and communication by updating users once their feedback has been implemented so they know their voices are heard and valued.

What are our brand guidelines

Principles

Simple
Help navigate complexity and remove friction

Modern
Has a look and feel ahead of the present days

Memorable
A brand that stays and earns trust

Name

“Resend” is the brand name and always spelled with a capital “R”.

It is a single word and should not be spelled as “ReSend”, "resend", or any other variation.

Typeface

Here's an overview of the typefaces used in the Resend brand:

- Display: ABC Favorit by Dinamo

- Product: Inter by Rasmus Andersson
- Editorial: Domaine Display by Klim Type Foundry

Assets

You can download the assets at resend.com/brand

How we think about design

We are a team that appreciates quality and understands how design resonates through the entire lifecycle of the product, from the infrastructure to the final pixel that reaches the user. Here are the essential values for crafting design at Resend.

Be close to customers

> *"To find ideas, find problems. To find problems, talk to people."*
>
> *— Julie Zhou*

Our mission is to **untangle the complexities of customer needs**, crafting solutions not with embellishment but with precision and empathy. It's a dance with simplicity, where every step, every feedback loop is an opportunity to listen and improve the product.

Key points to keep in mind:

- Use the product
- Do customer interviews regularly
- Work closely with the customer support team

Design is one

> *“In design, be logical, search for truth, be clear.”*
>
> *— Massimo Vignelli*

Borrowing the words from Vignelli, that define design as a thinking discipline above any style. We believe design is **not just a step or something to be applied in the end**; It's a holistic approach that acknowledges the important dialog between form and function, where every line of code contributes to the user's experience.

Key points to keep in mind

- The codebase is the source of truth of design, not Figma

Craft is care

> *"Have nothing in your houses that you do not know to be useful or believe to be beautiful."*
>
> *— William Morris*

This is our ode to craftsmanship—where every line, every shade, is a testament to our dedication. Like the venerable artisans of the Arts and Crafts movement, we imbue our work with a spirit that seeks **not just to exist but to endure**, to stand time.

Key points to keep in mind

- Work as close as possible to the final media
- Every detail is an opportunity to delight the user

Solve the hard problems

> *"Recognizing the need is the primary condition for design."*
>
> *— Charles Eames*

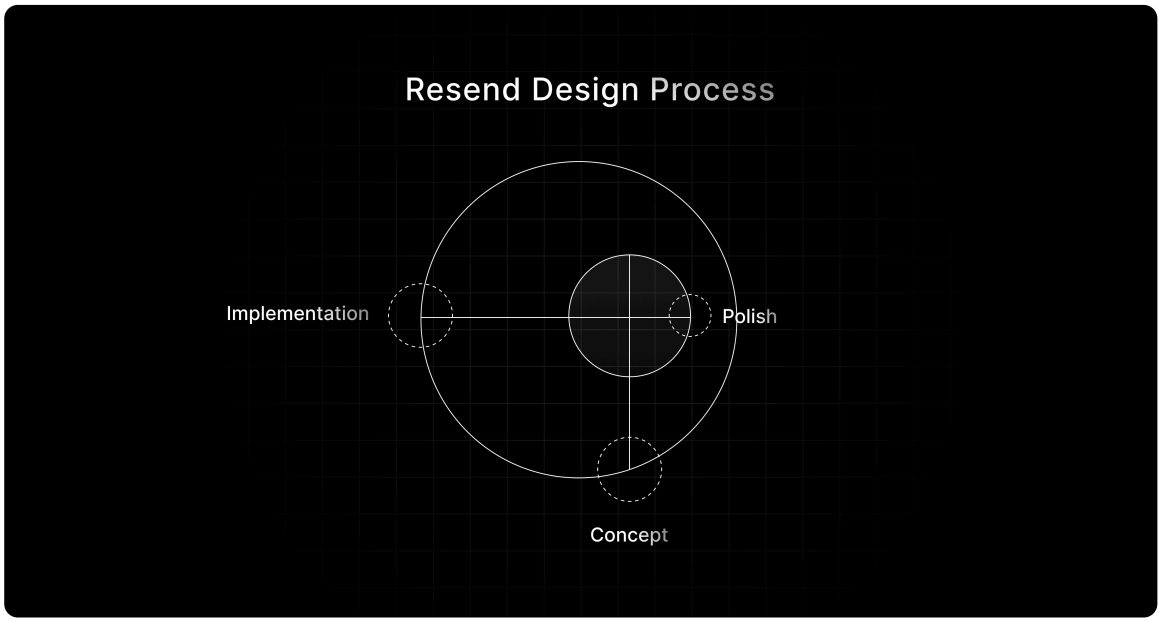
Charles Eames once envisioned the **designer as thoughtful hosts**, anticipating the unseen and unspoken. We share this philosophy as we navigate design challenges, looking for profound solutions instead of superficial ones. Our focus is provide the right insight, in the right place, at the right time, rather than just adding another feature. It's a commitment to **confront complexity with courage**, armed with empathy and insight.

Key points to keep in mind

- Explore the problem while designing

- Cycle between different hats: the designer, the user, and the stakeholder

What is our design process



One of Resend's goals is to ship high-quality software quickly. This is crucial for the current stage of the company, so our design process is built based on these constraints.

We believe design is a discipline that can be **exercised by anyone in the company**. It is a way of translating user needs and business requirements into real solutions. This process can start with customer support, operations, or engineering.

Sharing taste and the **meaning of quality** is crucial for cultivating an open culture about design, and the key to achieving that is collaboration. It is very common for us to design solutions together, booking an hour to go through a problem and generate high-fidelity mockups in Figma together.

Here are the three phases the team can follow to design high-quality solutions:

Concept

The first phase is focused on **navigating the complexity of the problem**. The goal is to frame the problem, how we can solve it, what resources we need, and how we want the user to feel. It's a **combination of research and play** with the problem and the tools you have available: writing, coding, sketching, Figma, Retool, etc.

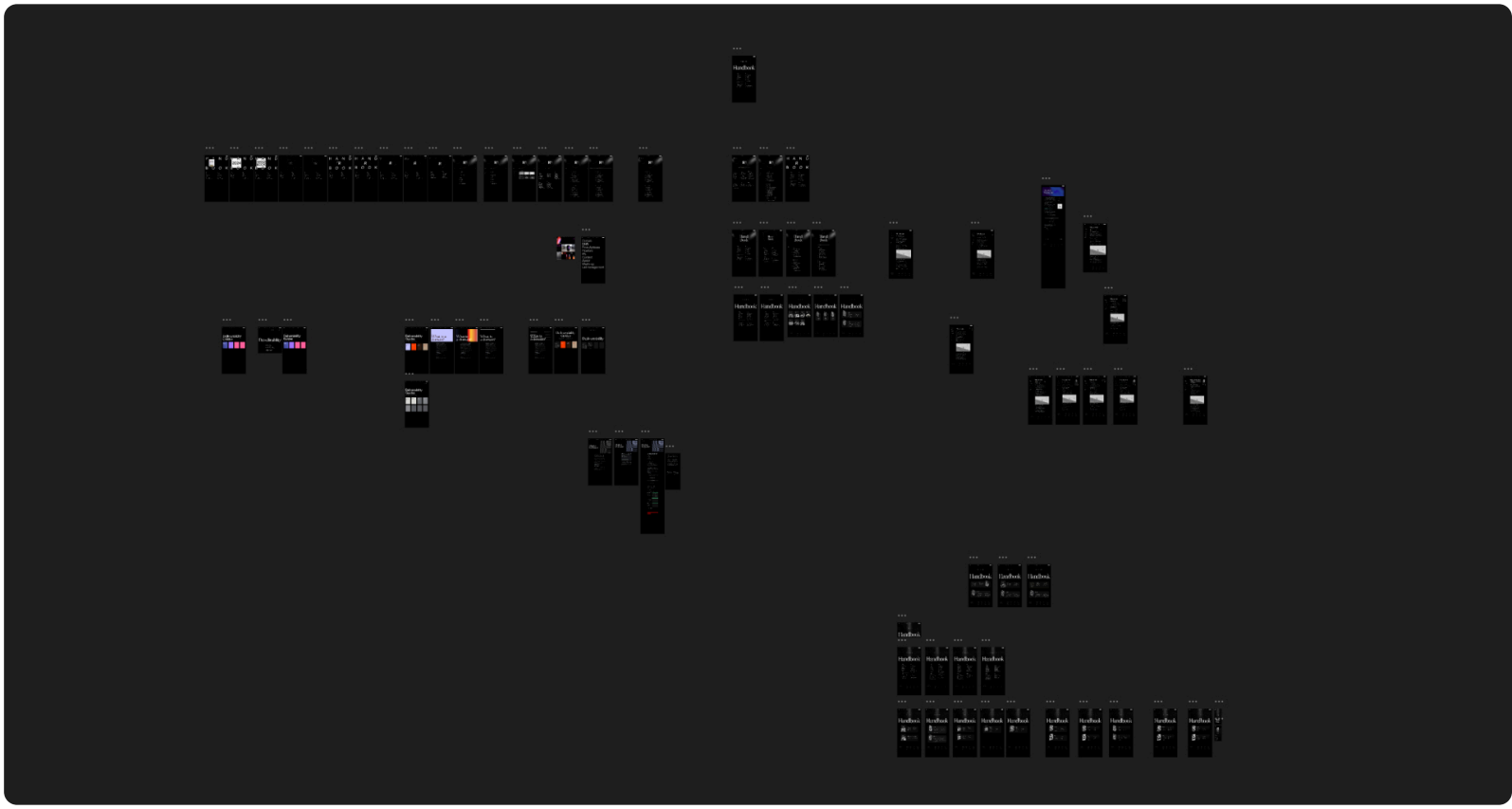
While we play with possible solutions, we increase our understanding of the problem, realize what we don't know, and search for answers.

The progress is **shared publicly with the entire team**. This transparency ensures that everyone, from developers to customer support, can provide feedback or follow the progress. It encourages a collaborative environment where diverse perspectives are integrated into the final solution.

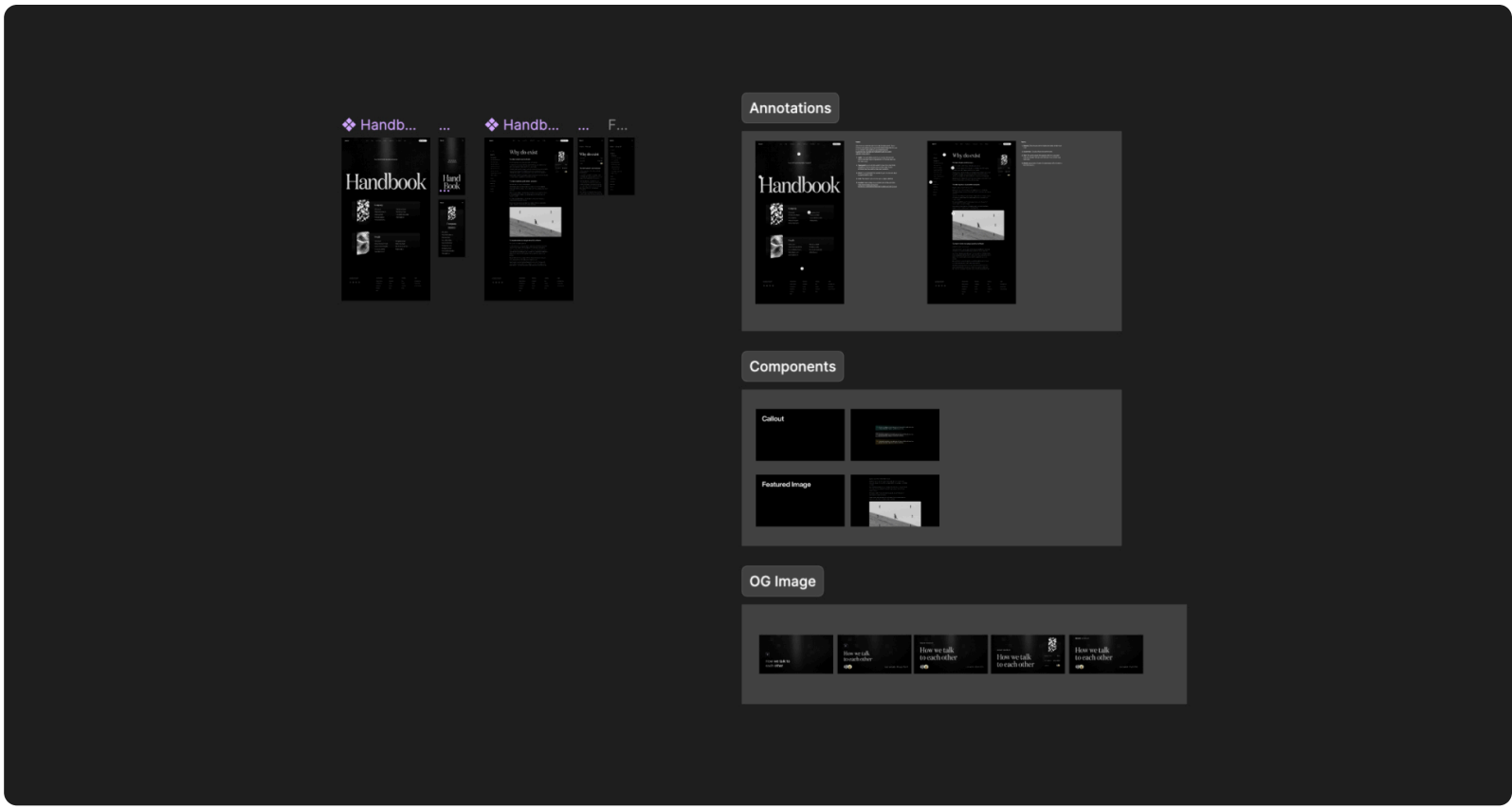
The output is not the final solution but an overall idea of how to solve the core problem.

Key points to keep in mind:

- Learn and discover the problem by doing
- Collaborate to get multiple perspectives
- Prototypes standout



Exploring the design problem



Overall idea to go through implementation

Implementation

The software industry has the privilege of rebuilding things, which not all industries have the luxury of doing. So, we will start implementing it as soon as possible to learn fast.

Many details and edge cases are addressed during implementation. We revisit the design as needed without stopping the main implementation, allowing us to iterate with real data and **real scenarios instead of imaginable ones** that sometimes people get attached to in discussions.

The software industry lacks good integrated tooling where the entire team can work in the same environment and see exactly what the user will see. Quickly moving to implementation **helps design leave the dreaming Figma world**. We prefer taking screenshots of the product and iterating on them rather than having the perfect design library.

Key points to keep in mind:

- Code can be discarded and rewritten as needed
- Address edge cases as they arise

Polish

This is where the magic of Resend happens. Once the solution is implemented, the team and some beta users can start using and experiencing it. What were previously separate pieces or hypotheses can now be felt as a whole solution, helping us identify gaps and test the entire user journey. Here, we are very committed to ensuring the solution truly solves the problem in the **simplest way possible** and how the user feels during the entire journey.

Based on feedback, we begin refining the details. After the engineering team finishes the foundation, the design team goes directly to the code to improve animation curves, fix misalignments, and **add quality-of-life improvements to the experience**. The operations team can tweak parameters exposed by the developers for fine-tuning and fast iteration, and customer support can help improve labels and documentation. The **code is the source of the truth** and where the entire team can work together.

Key points to keep in mind:

- Trust in your taste
- Make sure to solve the problem
- Test the experience as a whole

The Resend design process in action.

This is the Resend design process. It's as lean and collaborative as possible, eliminating handoffs and fragmentation in software development.

How we approach CI/CD

CI/CD (Continuous Integration and Continuous Delivery) pipelines are the core of our development and deployment process. Pipelines automate the build, testing, and deployment of our applications and services.

GitHub

We use GitHub as our primary version control system. GitHub provides a centralized platform for versioning control, enabling cross-collaboration, code reviews, and tracking changes across teams and projects.

Our preference is for smaller and more frequent changes. This approach promotes agility and makes us iterate fast, shortening the feedback loop. Breaking down tasks into smaller components helps us track progress, review code changes, and identify bugs earlier on.

Build & Deployments

Our build process starts by pushing changes to a repository on GitHub. When code is pushed to a repository through a **pull request**, it triggers a job to build the changes made to the branch and deploy them in isolation. This workflow happens interactively within the pull request, where the author has visibility of all steps.

Some tools we use offer preview URL deployments that contain the latest changes for each PR. Every new commit in the branch triggers a new build, and this isolated environment always contains the latest changes.

This practice enables the engineers to preview and validate the changes before anything is pushed to production.

Going Live

After the PR review, a squash merge is performed into the `main` branch. Once we are ready to trigger a release, the `main` branch is rebased into `production`, triggering the pipelines to deploy the latest updates.

How we use RFCs

An RFC, or “Request For Comments” is a detailed technical document we use to start new projects.

Sharing ideas, issues, and improvements is great, but documenting the problem, researching how to solve it, and describing the best way to release it is better.

Types of RFCs

- **Idea proposal:** We always have great ideas from time to time, but just sharing is not enough in a fast-paced company. We document, prioritize, and ship it.

- **Complex bug:** It usually comes from the user's feedback. We log the user's bugs and track them on Linear. When we notice an increase in the bug happening, we document to understand and share context with everyone and come up with better solutions to solve it.
- **New feature:** The RFC helps us understand and do research about the problem and solutions. It also helps us understand the scope and complexity of the project, map technical details and their impact, and how the release plan.

Why do we write RFCs

Having the RFC document written down is really helpful for a remote team to collaborate and share context and ideas async. It's been extremely important to us to understand the entire scope and size of the project. It allows us to do deeper research on the problem we're trying to solve.

It facilitates feedback and drives consensus. It is not a tool for approving or committing to ideas but more so a collaborative practice to shape an idea and to find serious flaws early.

How do we write RFCs

Here's a summary of the template we use.

```
# Purpose

Describe the specific problem or opportunity for the problem.

# Background

Context and detail the current situation and how we've acted on the problem.

# Proposal

Outline the proposed solution or change.

# Technical Details

Dive into the technical aspects of the proposal. Map the changes we must make i

# Implementation Plan

We break down into phases, each phase should have a goal for us to release in p
```

```
# General Questions

Answered questions that people may have about the proposal suggestion.

# Open Questions

Open questions to everyone to contribute and help evolve the RFC.
```

Discussion

After the RFC is written, we share it on Slack so everyone can see and collaborate on the discussion and decisions regarding the proposal.

We use Notion to write and document the RFCs, so people leave comments on certain parts of the proposal with suggestions or questions to be followed up by the writer or others.

When everyone is happy with the decision, we move the RFC to `Approved`.

How we review pull requests

Give context

As the author of a pull request, it is crucial to provide context since you have a deeper understanding of the changes, while the reviewer does not.

Although not all pull requests require extensive context, it can be beneficial. Sometimes a simple screenshot or short video can go a long way in helping the reviewer understand the changes.

We ensure the pull request title is clear and provides a good description of what the pull request accomplishes. Highlight specific areas where we need feedback and include links to related documentation, issues, or other pull requests for additional context.

One change at a time

Each pull request should be focused on a small and specific task.

This approach minimizes context switching, making it easier for the reviewer to scan and review the changes.

If multiple changes are necessary for different improvements, we separate them into individual pull requests. This reduces confusion and the risk of introducing bugs or regressions, as the reviewer can concentrate on one thing at a time.

Responsibility

Taking responsibility for the code we approve is essential. If a change causes a bug after deployment, it is partly the reviewer's responsibility.

Code reviews are a core part of our job, not a distraction. When we provide feedback, we don't just point out issues; we explain why an alternative solution might be better. For instance, instead of saying, “We shouldn't use this approach” we suggest, “Here's another way that could be better because of X and Y.”

Understanding the code being changed is vital; if we don't understand it, we can't effectively review it.

To avoid excessive back-and-forth in pull request comments, we usually use Slack DMs or huddle to have better discussions when needed.

Tools

Our CI/CD integration ensures that dependency checks, linting, formatting, building, and testing are automated. Additionally, deploying preview apps can provide better context and facilitate more effective testing.

How we ship new features

We ship early

We constantly review the scope of a project and work toward a v0, not a v1.

That means identifying the **most crucial part of a functionality**, and pushing the smallest pull request possible to make that happen. In fact, a PR should always be optimized for the speed with which we can merge it in.

It's vital that we put new things **in the hands of users as soon as possible**. Building in a silo is not a great idea, and we can only learn when real people are using a feature in a real-world scenario.

We ship often

Shipping early is not enough. We also need to keep shipping tiny iterations daily instead of waiting months to ship something big.

Having a **strong sense of urgency**, and making hard decisions fast is how we enable that process. Our sprints only last 1 week and are quite short when compared to the rest of the industry - this is by design.

Engineers respect teams that are “always shipping”, and we believe that great things are built one brick at a time.

We ship together

It's very common for a feature to be **started by one person and finished by another**.

Sometimes, someone will work on the first 80%, and then someone else will push the last 20% to the finish line.

This is a great way to share knowledge and make sure that everyone is familiar with the codebase.

Using preview deployments

For our dashboard and public website, we use Preview Deployments from Vercel.

Having a unique URL for each PR we open is vital. They are great not only for getting feedback internally but also for **gathering opinions externally** from peers who work at different companies. Fun fact: the preview URL of this handbook itself was shared externally with many of our friends before it was merged into production.

For our API, we also get an unique URL pointing to our Staging environment that allows us to perform extra tests even before merging the code.

Using feature flags

It's always a good idea to put new features behind a feature flag.

We use Posthog for that, which contributes to a rollout strategy that can surface user feedback in a much more contained and safe environment.

When thinking about how to rollout a new feature to users, first think about your peers. Many of the Resend team members also use Resend on their own side projects. So **asking feedback internally is an amazing way to improve the quality** of a feature before a wide rollout.

Before releasing to all users, **reach out to those who asked for that particular feature** in the past. Every time a new feature request comes in, we register who asked for it. Closing that feedback loop is one of our favorite things.

Reminder: feature flags are great, but not every feature needs it too.

Shipping to production

Before shipping to production, we think about all the different artifacts that might be affected by a new feature.

Here's a non-exhaustive list of things we ask ourselves:

- Does the API need to be updated?
- Is the OpenAPI spec up to date now?
- Have we released all the official SDKs?
- Do we need to write a Changelog post to announce this?
- Is there any existing Documentation that needs to be updated?
- Do we need to write a new Knowledge Base article to go along with this?
- How does this change affect our Support team?

Every new feature is a chance to make the life of our users better. We take that very seriously.

How we plan our cycles

Preparation

Before each week begins, we identify potential projects and set high-level goals for the upcoming week. This includes meetings with support and operations teams to discuss their initiatives, blockers, and any issues requiring our assistance.

Our cycle is typically divided into two main fronts: “Foundation” and “New Projects.”

Foundation

These projects focus on stability, bug fixes, technical debt, and tasks to improve our internal developer experience (DX).

New Projects

These initiatives focus on new features, improvements, and anything that will help us grow.

Planning

We hold an engineering meeting to share and discuss all proposed projects to see if they make sense or if we're missing something important. This ensures everyone is aligned and aware of the goal at hand.

We map everything on Linear and utilize their cycles for tracking.

Review

Before every engineering planning, we review the last week and update our cycle document projects to understand what went well and what didn't go well.

Execution

For new projects, we usually start with RFCs, which are shared with the team to develop solutions collaboratively.

We hold daily stand-up meetings to share progress and updates on all projects.

What is our tech stack

Back-end

We use TypeScript and Node.js across our back-end applications. They are divided into serverless functions using Next.js API Routes and long running Express servers.

Our applications generally try to follow a single concern, for example, our public API is served from an Express server, our SMTP from another one, but we have a few others

used internally for operations or background jobs.

Front-end

We use React with Next.js for our dashboard and public website.

For styling we build custom components using Tailwind CSS, Radix Primitives, and Radix Colors.

We currently use SWR and API routes to make requests to the back-end to fetch the necessary information, as well as Zod to parse responses into strongly-typed schemas.

Deployments

We have workloads spread across different providers, such as Vercel, Railway, and Fly.

Depending on the problem we're trying to solve, we choose the best provider for the job. For example, we use Vercel for our Next.js apps, which allows us to ship features quickly using Preview Deployments.

Storage

We use Postgres as our main storage system, the applications consume data using Drizzle as the ORM/query-builder.

We also have Redis, used for caching some data for fast access.

Observability

We use a combination of tools to monitor our applications:

- Datadog
- Grafana/Prometheus
- AWS CloudWatch

Background jobs

We have many background jobs that are triggered from different flows; the goal is to make requests faster by moving non-critical code out of the main path or to perform an action continuously based on some criteria.

Our main tool is Inngest, which provides a great developer experience and helps us visualize and debug background jobs.

Documentation

Our documentation is written using Markdown and it's powered by Mintifly, which let us focus more in writing the best content we can.

Webhooks

We use Svix for processing webhooks to our users. It's a great tool that helps us monitor and scale webhooks with automatic retries.

Testing

We use Vitest as our testing framework for unit testing. We don't enforce a specific test coverage, and we try to be reasonable in balancing the amount of tests we write.

We don't have UI tests or E2E tests in our dashboard, but we know we need them sooner rather than later.

Code styling

Recently, we switched from ESLint and Prettier to Biome and noticed huge performance improvements. We have been happy about it, even though some features from our previous setup are missing.

Miscellaneous

- pnpm as a package manager
- TypeScript is the default. We try to be very strict about its rules
- Pino is our logger library across the board
- MDX is how we write content

How we recognize peers

It's not easy to do great work. It requires humility, service, empathy, focus, diligence, clear communication, and much more. One way to encourage each other to keep doing our best work is to compliment them.

Just saying “Good job” isn't nearly enough. Let's break down how to give exceptional compliments.

Be authentic

> Ask yourself: Why am I recognizing this person?

We give compliments to promote greatness and instill courage, not to get promoted or affirm poor performance. Double checking your intentions before giving compliments is important to maintain a pure compliment culture. Ultimately, if you are inauthentic in your recognition, people may think you are inauthentic in other areas as well.

A good rule to follow is: Don't compliment someone because you feel you should; compliment them because you feel compelled to let them know how they impact you or others.

Be specific

> Ask yourself: What did I experience or observe?

Clarity is kindness. This is true for all feedback, including compliments. Leaving no room for doubt reinforces the authenticity of your compliment, showing that you really appreciate what you are complimenting.

Imagine telling your best friend that you love them, but when they ask you “What do you love about me?”, you can't give them a clear answer? The lack of clarity will refute the entire compliment, and even make it feel like more of an insult.

Providing details and examples to help the person comprehend the context of our remarks can be a great way to be clear.

- **Good compliment**: “Thanks for taking notes in the meeting, you're amazing!”
- **Amazing compliment**: “Bethany, I know it is your job to take notes in the meeting, but because you do it so well, I know I can relax and focus on doing my job. Thank you.”

Process over results

> Ask yourself: What did it take for them to do what they did?

It's easy to see compliments as a way to simply recognize someone for their achievements. Instead, showing appreciation for the process and effort that went into producing the results can be much more impactful. Compliments that only focus on the result often trigger a concern in the receiver of not being able to produce the same result in the future.

When recognizing someone, show them that you appreciate the time, sacrifice, creativity, or care that went into their work.

- **Good compliment**: “Brady, I'm amazed that you always go above and beyond for customers. Every month, you are showing high NPS scores. Keep it up!”
- **Amazing compliment**: “Brady, I am amazed by the deck you put together for the client. I can't even imagine all the hours, work, and creativity that went into making that presentation happen. Thank you for everything you did behind the scenes on to make this happen.”

Share the impact

> Ask yourself: How did their actions impact me or the team?

It's tempting to think that compliment is only about the receiver, but there is a reason why you are the one giving the compliment. Including the way you experienced or were impacted by their work, gives the compliment extra gravity. None of us care to do work that stays in a vacuum. We want to know that our work as meaning and impact.

Consider sharing how their ownership impacts the team, their work changes the company's results, or how their attitude influences the team culture.

- **Good compliment**: “Trinity, you do a great job at focusing on your work at not getting distracted with unimportant things.”
- **Amazing compliment**: “Trinity, the whole team is benefiting from your ability to focus on what's important. Personally, I felt myself getting lost in the details but you were so helpful to clearly align the next steps so we could keep the project moving! So grateful for your influence on our team.”

The Compliment Checklist

The next time you feel compelled to give a compliment, before you share, pause for a minute and answer the following questions:

- **Authentic:** Why am I recognizing this person?
- **Specific:** What did I experience or observe?
- **Process:** What did it take for them to do what they did?
- **Impact:** How did their actions impact me or the team?

How to give feedback

We all want to be our best. One of the ways we keep striving for that is by giving and receiving feedback. This is often done in the context of manager and direct report relationships, but it should be done across between team members, regardless of status or tenure.

We've all received poor feedback that is not helpful or constructive. Though the receiver should aim to squeeze the maximum value out of any input, regardless of how poorly it is delivered, the goal of the feedback giver is to be helpful, encouraging, and immediate.

Immediate

> Feedback should immediately follow the behavior.

What did you have for lunch last Tuesday? Don't remember? Neither do I!

The more time that passes, the more cloudy our memory becomes. This is why waiting to provide feedback can prevent some of the opportunities for change. The only way to change is to know what needs to change. Waiting weeks or months to share feedback means neither party has clarity on what happened.

Waiting also leaves room for feelings to grow and fester. It's hard to be helpful when you're mad; it's hard to be encouraging when you're bitter.

It can be hard to find the time and energy to share feedback in the moment, but it is the most effective way to help your team move forward and iterate quickly. Ultimately, immediate feedback is about fueling progress.

> If something seems off, jump on a quick huddle to talk it out right away.

> Don't go so fast that you don't gain full context. Ask questions before sharing your thoughts, and use “I felt/noticed” language over “you did/are” language.

Helpful

> Feedback should be beneficial for the listener.

This starts with empathy. You cannot help someone unless you know what they need. And you can't know what they need unless you are really understanding their context. This can mean spending time “putting yourself in their shoes” or asking questions prior to giving feedback to make sure you fully understand where they are coming from.

Examples:

1. A team member is always showing up late for a regular meeting. Instead of telling them that you don't want them to be late anymore, check the meeting time in their timezone and check their other meetings in their calendar to see if it's ideal for them.
2. A direct report is showing frustration towards a change in priorities. Instead of telling them that they need to just be ok with things changing, ask them how the last change impacted their existing work or ask them how you could share news of changes better.

> Put yourselves in their shoes before sharing your feedback.

> Identify the root cause and then ask yourself: what would I need if I was experiencing this?

> Prepare 1-2 clear examples to share along with your feedback to bring clarity.

Encouraging

> Feedback should instill bravery.

It's scary to change, especially when it requires doing something you aren't good at. **Feedback should have a “heroing” effect**, where it makes someone feel strong and brave enough to tackle the changes head-on. They should feel like the superhero of their story not the scared recruit in the trenches.

A helpful way to think about this would be to ask: **how am I affirming their ability to change?** This can be done while the feedback is being given by sharing ways that they have shown their ability to achieve similar triumphs, but often people can feel guarded in that moment and don't receive this encouragement fully. Ideally, this affirmation is done

before and after the feedback. This means **being proactive with this relationship beyond just when you want them to fix something.**

- Share what you saw them do well alongside what could be improved.

- Affirm positive change as soon as you see it before and after the feedback

What is the travel policy

Flight Booking

- No preference on airline or booking method
- Book directly with airline using your Brex Card
- Reduce layovers and connections to prevent long flights
- Try to keep the price within 1.5x the cheapest option

Add-ons

Approved add-ons

- Checked bags
- Minor seat upgrades (when meaningful)
- In-flight WIFI
- In-flight meals
- One massage per travel (great for a layover to returning refreshed)
- Transportation to/from the airport

Unapproved add-ons

- First/Business class upgrades
- In-flight alcohol
- Travel insurance (Brex provides this if booking through Brex Trips)

- Rental cars

Meals

- Don't go hungry; buy food, snacks, and liquids to stay nourished
- One alcoholic drink per day
- Prices vary by location; be reasonable but enjoy yourself

Generally aim for:

- $20 for breakfast
- $30 for lunch
- $50 for dinner

Receipts

Take a picture of receipts and upload to Brex via:

- Brex Mobile App
- Brex website
- Email (receipts@brex.com)

Reimbursement

Using your Brex card is preferred, but if the vendor does not accept it or you forget it, simply keep the receipt and request a reimbursement.

If you don't have the receipt, please submit a picture of the transaction on your bank statement.

How to spend money

Guiding Principles

Spend conservatively

We want to spend less than we make. This means not spending just because we can, but making sure every expenditure is bringing value and that it makes sense in light of

the health and stage of the company, especially revenue.

Spend transparently

We want to share our costs as much as we can with the whole team so that the whole team can challenge and push us to use our money better.

Spend autonomously

We want to limit approvals and bureaucracy. For smaller transactions, their approval is mostly dictated by a policy; no review is needed. Larger and recurring expenses should be discussed as a team. If a decision is hanging, a leader can make the final call.

If you ever have a question about an expense, you can ask in #all-expenses.

Policies

Existing budgets

- **Onboarding budget**: this is a stipend provided to all new employees to purchase office supplies that will make them happy and productive.
- **Travel budget**: Pre-approved expenses can be found in What is the travel policy.
- **Productivity software budget**: Every employee has a budget of $500 per year for tools and systems that make them more productive. No questions asked.

Hiring Approval

One of the biggest expenses is payroll, which makes sense why they can fall under the most scrutiny. In order to post a job description, we must decide on a salary budget for the role. Every salary budget must be approved by the executive team.

Reimbursements

Using your Brex card is preferred, but if the vendor does not accept it or you forget it, simply keep the receipt and request a reimbursement. Either way, make sure to upload your receipt to Brex.

If you don't have the receipt, please submit a picture of the transaction on your bank statement.

You can also download the Brex App on your phone to easily see transactions, check card details, and even book travel.

How to take time off

Types of Paid Time Off

There are a few types of paid time off (PTO): Sick Days, Vacation, and Family Leave.

Sick Days

- **Purpose**: PTO due to a physical ailment.
- **Request Timing**: It is usually last minute and is not coordinated ahead of time.
- **Approval**: Sick time can be taken at any point without approval. If more than 2 consecutive sick days are required, it must be discussed with your manager.
- **Amount**: Unlimited.

Vacation

- **Purpose**: Vacation is chosen by the employee for any reason they desire. Purpose does not have to be disclosed.
- **Request Timing**: General rule of thumb for request timing is to request 2x the amount of days prior as you are taking, with the minimum being 1 week in advance. For example, if you are taking 3 days off, request it 7 days in advance. If you are taking 2 weeks off, request it 4 weeks in advance. The more time in advance you can request, the more time you have to work out any conflicts.
- **Approval**: Vacation is part of your compensation and you don't need approval to use it. With that said, coordinating with your manager to make sure all duties and responsibilities can be delayed or covered is essential to keeping operations smooth regardless of PTO. In some cases, the manager may have feedback about the timing due to specific projects or deadlines and will discuss options with you.
- **Amount**: 20 days a year.

Family Leave

- **Purpose**: Family Leave is intended for families that are introducing biological or adopted children into their family.
- **Request Timing**: Family leave should be coordinated 3-6 months in advance.
- **Approval**: Family Leave is part of your compensation and you don't need approval to use it. You also cannot be penalized because of taking Family Leave in the form of

termination or pay reduction. With that said, it is a big chunk of time and major changes may be required to make it work, including hiring to fill the gap.

- **Amount**: 3 months. This can be broken up into chunks across a 1-year period.

How to Request PTO

- Ping your manager with the details (written is better than verbal)
- Describe dates and a plan on how duties can be covered if applicable
- Add time off request in Rippling
- Add time off to the Team OOO Calendar

Note: you are not obligated to share why you are taking time off or what you will be spending it on. You are welcome to share those details if you would like to for your own interest or if you feel it would be helpful to others.

How to Leave for PTO

This is a generic list that can apply to all employees and roles. There will likely be specific actions you can take before you leave for PTO that apply to your role.

- Set autoresponder for your email
- Set Slack status to OOO message
- Document any tasks that other people need to take and are not trained on
- Share project/task updates with your manager or stakeholders
- Share a second message in #all-ooo when you are signing off (people forget)

Working While on PTO

You are not expected to work while on PTO unless a special arrangement has been made with your manager for specific tasks. It is tempting to keep tabs and complete high-value tasks while on PTO, but there are many benefits to *not* doing that:

- **You come back more restored** (improves your quality of life and makes you more productive)
- **We discover if there are tasks that no one else can do but you** (something we can consider fixing when you return)
- **We invest in the long-term health of you and the business** (if we build a business that cannot let anyone take PTO, that business will not last)

What are the benefits

Home Office Allowance (Global)

Spend $2,000 to get your home office primed for optimal flow.

This is in addition to the macbook provided by Resend.

Unlimited Sick Days (Global)

Take the time you need to rest, recover, and get back on track.

Annual In-Person Retreat (Global)

Travel and meet up with the entire team around the world.

12 Week Family Leave (Global)

Bond with your new biological or adopted child.

This can be taken all at once or spread out over the first year.

20 PTO Days per Year (Global)

Take time off for whatever reason you want. No work allowed.

10 Holidays per Year (Global)

Choose the 10 US holidays or swap in local ones.

2024 Holiday Schedule:

- Jan 1st: New Year's Day
- May 27th: Memorial Day
- June 19th: Juneteenth
- July 4th: Independence Day
- Sept 2nd: Labor Day
- Nov 28th: Thanksgiving
- Nov 29th: Black Friday

- Dec 24th: Christmas Eve
- Dec 25th: Christmas
- Dec 31st: New Year's Eve

401K (US)

401K account provided to invest in your future (no matching).

Health, Dental, Vision (US)

100% Employee, 90% Family principle covered by Resend.

Life Insurance (US)

Care for your family and loved ones, even after a life crisis.

How we think about remote work

Messaging

We use Slack as the primary Operation System for our messaging. If some information should be remembered, we transition it to Notion. It's easy to shove all communication into Slack, it's much harder to work efficiently and communicate well.

Here are a few ways we achieve that:

Channel purpose should be in the name

We add prefixes to all channels to indicate its purpose. Every team member should be able to see a channel with unread messages and be able to triage how urgent it is or what type of work is involved with it by just looking at the name. Here are some of the prefixes we use:

- `bot-` system notifications that usually **don't** require action
- `alert-` system alerts that usually require action
- `ext-`, `cust-` discussions with customers
- `collab-` discussions with vendors/partners

- `proj-` temporary working groups about a project
- `all-` permanent channels to discuss topics
- `inc-` incident working groups

When to thread, post, or DM

There are many ways to utilize threads, channel posts, and DM's to get work done. Here is a general matrix for when to use each:

Threading

- Good for staying focused on a topic, having multiple conversations in one channel
- Bad at getting quick responses, giving visibility to members not in the thread

Posting in Channels

- Good for informing all members in that channel, preparing for a thread
- Bad at targeting a message to certain members, asking for individual collaboration

DMing an individual

- Good for asking for collaboration, keeping a discussion private
- Bad at informing all members, being searchable for all members

Tasks

We use Linear to manage all tasks. If you are asking for another team to help with a task, you should create a Linear ticket for their team. It's easy to create a ticket from a Slack message by clicking into the menu and selecting “Create Linear Issue”. This also syncs the Slack thread of that message to the issue, depending on where the discussion happens.

The anatomy of the perfect ticket

- Title is short and succinct
- Description contains every details that the person implementing would need
- Think about questions they would ask, and already answer edge cases
- Break down bigger issues into sub-issues
- Include severity so the other team knows how to incorporate into their existing work

Working Hours

Although we are remote, we believe that centralizing our working hours is helpful to improving collaboration by increasing the amount of back and forth that can happen between members.

We currently hire within the America's to keep the whole team's working hours naturally aligned. Generally 15:00-19:00 GMT is when we are online and more accessible to each other. Specific teams may align expectations of working hours outside of that window.

We don't mandate work on evenings or weekends. Those are your personal times, so you can choose to do whatever you want. If you feel like you have to consistently work during those times just to keep up with goals or demands, please raise this with your manager so we can think about reducing work or hiring help.

Meetings

There are many good reasons to book a meeting. Here are some helpful guidelines for doing that well:

- Try to solve it async, if that fails, then book a meeting
- If you are demoing something, record a video and share on Slack instead
- For pairing, jump into a Slack Huddle rather than a meeting room
- Meetings default to camera on, Slack Huddle default to camera off
- Cancel recurring meetings if they aren't useful and consider setting an expiration

Handoffs

The way plans and requests are passed between team members can make or break a small, remote team. We must get feedback from each other as soon as possible to validate ideas, and we must remove potential blockers from each other before work is started to not break up the flow.

RFCs play a huge role in this. This is where we put ideas onto paper and get feedback. It is also where we identify roadblocks and hurdles to completing the task. The deeper the RFC goes, the less surprises and scope creep we have later during the implementation. This is also an important way we expand our “group brain” and don't lose an idea after thinking of it (Slack is not good for this).

Another way to accomplish this is by creating really good tickets (see tickets section above). Most notably, by anticipating questions and roadblocks that the implementer

might have and solving before the work is started, it means that someone can grab the ticket, understand everything, and build everything in one stream. No need to stop because of missing information.

How we work

Remote

Resend is a remote company. There are no plans for this, but even if there are some local offices in the future, the core operations of the company will always facilitate remote work because we believe in it as a way of working.

Being remote means we will **default to async** forms of communication like writing RFC's instead of booking a meeting, using Slack channels instead of waiting for synchronous stand-ups, and logging tickets in Linear instead of an office white board.

Being remote **does not mean we only work independently**. Especially at our stage, it's important to be deeply in sync and iterating constantly. To do this, we're careful to keep working hours inside of the America's timezone, plan 2 offsites every year for in-person time together, and often switch to synchronous work when its appropriate (incidents, beginning complex problems, etc.).

Curiosity over ego

> *"Egotism is the anesthetic which nature gives us to deaden the pain of being a fool."*
>
> *— Dr. Herbert Shofield*

We see many benefits to confidence, conviction, and bravery, but no value in ego. It hinders collaboration, squashes innovation, and destroys moral. It has no place at Resend.

Instead, we **encourage a confident curiosity** approach to work. With confident curiosity, we're brave enough to share our values, views, and convictions, but open enough to have empathy for other perspectives and to learn from other ways.

Two very simple ways to put this in practice is to:

- Ask before you tell
- Be the last to speak

Challenge the status quo

Resend is a challenger brand. We aren't inventing email, we aren't inventing developer experience, we aren't inventing dark mode. In many ways, we're simply taking known patterns and applying them to a space that has not traditionally received that treatment.

This requires us to be constantly challenging the “normal” approach in the little things. We must **see every detail as a way to delight and wow**, which won't happen if we're sheep blindly following a leader.

This requires asking hard questions like “why,” sometimes over and over and over again, until we get down to the most primitive conclusion and then build from that primitive.

Show when possible

> *"Don't tell me the moon is shining; show me the glint of light on broken glass."*
>
> *— Anton Chekhov*

Life is more than pixels, and communication is more than text. Both for users and team members, we aim to engage multiple senses when delivering an idea.

We do this with customers by including screenshots, screen recordings, and code examples in the docs. **Just a wall of text is not adequate** to helping users solve their issues. Let me see, hear, and feel it.

We do this with other team members by reinforcing feedback, ideas, and suggestions with references or examples. Instead of just saying *“This dropdown is hard to use”*, try describing your experience using the dropdown and share references of dropdowns you like or a screen recording of you using the dropdown to show why it may be hard to use.

Use the resources available

We need each other for many things, but one of those is not answering questions with answers already available. One example of that would be asking a team member a question that is already documented. Another is asking for them to find something for you that is publicly available. Another is for a team member to train you on a technology or system just because they know it.

In almost all of these cases, there should be a **best effort given to helping yourself** with the available resources before going to a coworker.

Some helpful tips of ways to solve your own problem before asking:

- Search Slack and Notion to see if someone has already answered it
- Search the code and try to read it before asking how a feature works
- Use AI for writing help, generic technical questions, or translations

Everything requires iteration

We would be foolish to think that we can get things right the first time. If we all accept that our **first attempt may be completely wrong**, then there is grace and understanding for shipping or sharing half-baked work, especially internally, and receiving feedback so the iteration can begin.

If we wait too long to share with others, we can often get stuck in our heads and waste time. Many times, the questions we labor over would be answered in 10 seconds if a few more eyes were on it.

This is also why performance is measured more by how well feedback is requested and incorporated, than by how good our work is on the first attempt.

Get feedback quickly and be ready to make your work better with it.

How we talk to each other

Radically intentional

Without constant and intentional communication, we cannot stay aligned with our goals. This is especially true for a remote company.

If we have questions, we should ask them. **If something isn't clear, we should ask** for clarification. If something doesn't seem to be headed toward our goals or values, we should challenge it.

Ideally, this is done async on Slack, so the whole team can stay in the loop and collaborate, but there are times that we need to talk it out face to face. A Slack Huddle or Google Meet is a great solution (make sure to ask before calling).

It is better to break async and **take a moment to synchronize** than continue with misalignment.

In the open

When companies grow, there's a natural tendency for people to start having conversations in DMs instead of group channels. This typically happens because they don't want to bother others or be too noisy. The problem is that it leads to silos and missed opportunities for collaboration.

We have to try extra hard to capture the benefits traditionally taken for granted in a physical office like spontaneous knowledge sharing over lunch or coffee breaks.

By having conversations in the open, we can **make information searchable** and future posts can be linked to old posts. We create space for serendipitous discussions and teamwork. This is especially important when thinking about folks who just joined the team.

There are obviously exceptions - when talking about compensation, performance, or other sensitive HR matters it's important to do this in private, but other than that we should **default to talking in the open**.

Active listening

Conversations are typically bi-directional, but that doesn't mean you should speak all the time just for the sake of speaking.

One misconception many people have is that we need to **listen to reply**, whereas in reality, we need to **listen to understand**. Understanding someone is far more important than replying to someone. Anyone can reply, few can understand.

Regardless of the conversation that we're having, we must practice the habit of active listening. Not just listening in order to reply, but truly trying to understand what point the other person is trying to convey.

Surgically clear

As humans, we all have a need to be liked. Sometimes we put the need to be liked in front of our honest opinions in order to please others.

We care about creating a **safe space where anyone can speak up** and share their personal opinions about a product, a feature, or an executive decision.

We prefer to be **intellectually honest instead of sugarcoating** the truth to avoid potential conflict.

Note: this doesn't mean you can be rude to others.

Be kind

Helping others is part of the job.

It's okay to admit you don't know something and ask for help - it's **a sign of strength, not weakness**.

Remember the words of one of the greatest comedians of all time:

> *"Everyone you meet is fighting a battle you know nothing about.*
>
> *Be kind. Always."*
>
> *— Robin Williams*

We always **assume positive intent** and we're kind to each other.

What are our rituals

Our team defaults to async forms of communication.

Although we avoid having meetings to prevent interruptions and keep the flow, we believe it's essential to have some face time with the team.

These are the current company-wide rituals:

Offsite

Every six months we travel to a different city for a week to talk about our progress, think about the future, and reflect on how we can improve.

Most importantly, this is a time to have fun, eat good food, play board games, and get to know each other. Our past offsites have been in:

- Cancun, Mexico (May, 2024)
- Rio de Janeiro, Brazil (December, 2023)
- Lisbon, Portugal (April, 2023)

Offsite in Cancun, Mexico (May, 2024)

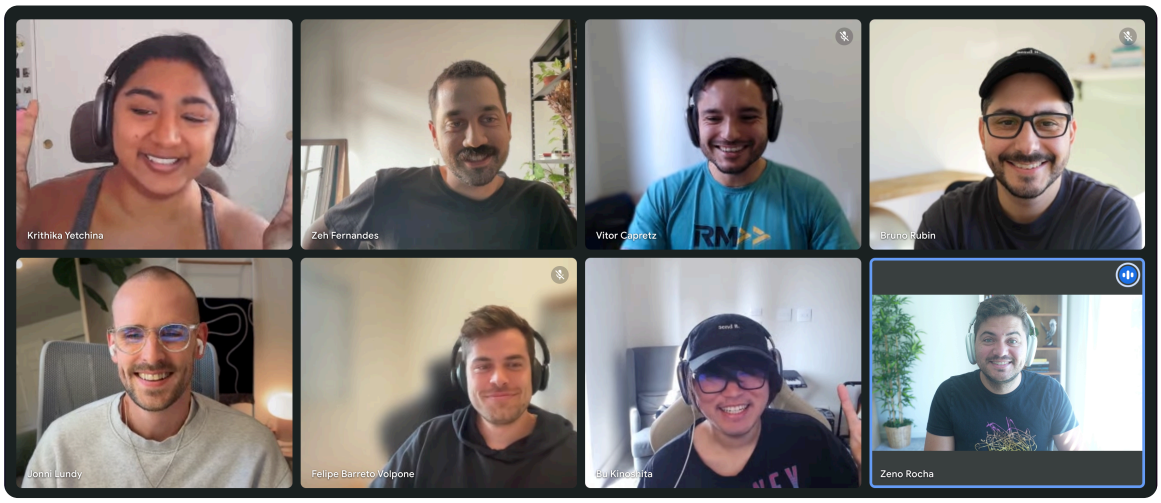
All-Hands

Every Monday morning the whole team gets together to share key metrics from each area, how we spent our time last week, and what we plan to tackle this week.

Daily Syncs

From Tuesday to Friday, we reserve 30 minutes to see each other's faces and talk about what we're working on.

It's a relaxed format, and we have a safe space to voice any blockers and share how we're feeling energy-wise.



Daily Sync (June, 2024)

Demo Day

Every Thursday, someone on the team shares their screen and talks about something interesting they're exploring or working on.

This is an excellent opportunity to showcase a new tool or process that can affect others.

How we make money

By giving first

As developers ourselves, we started our careers by using products that were either open source or had a free tier.

We believe it's important to **let people try the product** before they make a purchase decision, and that's why we have a generous free tier.

This is not only a way to capture new leads with a low customer acquisition cost, but also **a way for us to give back**.

Of course, we still need to pay our bills, so once a user likes our product and gains value from it, we should not be afraid to charge for it.

By providing value

Resend is a multi-product company, and we believe in providing individual subscriptions that reflect what people are getting value out of it, **instead of a one-size-fits-all** pricing strategy.

Some users only need transactional emails, others only need marketing emails, and there are many that need both.

Our pricing reflects that reality, and we structure it so that you can add new subscriptions or **cancel anytime** in case you no longer have a need for it.

By growing together

If a solo developer is sending a few emails for their side project, they shouldn't be charged a ton of money for it.

If a company is growing and dealing with very complex use cases, they should be **charged in a scalable way based on their usage**.

We believe in having progressive pricing tiers in which the **cost per unit gets cheaper** as the volume grows.

That way, solo developers and scaling companies can benefit the same way.

What inspires us

We are inspired by a variety of things, they represent the quality that surrounds us and that we want to bring to our work. Here is a non-exhaustive list of places, people, documentaries, objects, and games that inspire us.

Places

- Aoraki for its landscapes, provoking awe and a profound connection with nature
- Brasilia for the visionary architecture with profound urbanism lessons
- Gärdet for being a calm and peaceful place to go on long walks
- Griffith Observatory for the breathtaking views and iconic architecture
- Zereno for the amazing craftsmanship applied to the art of making coffee

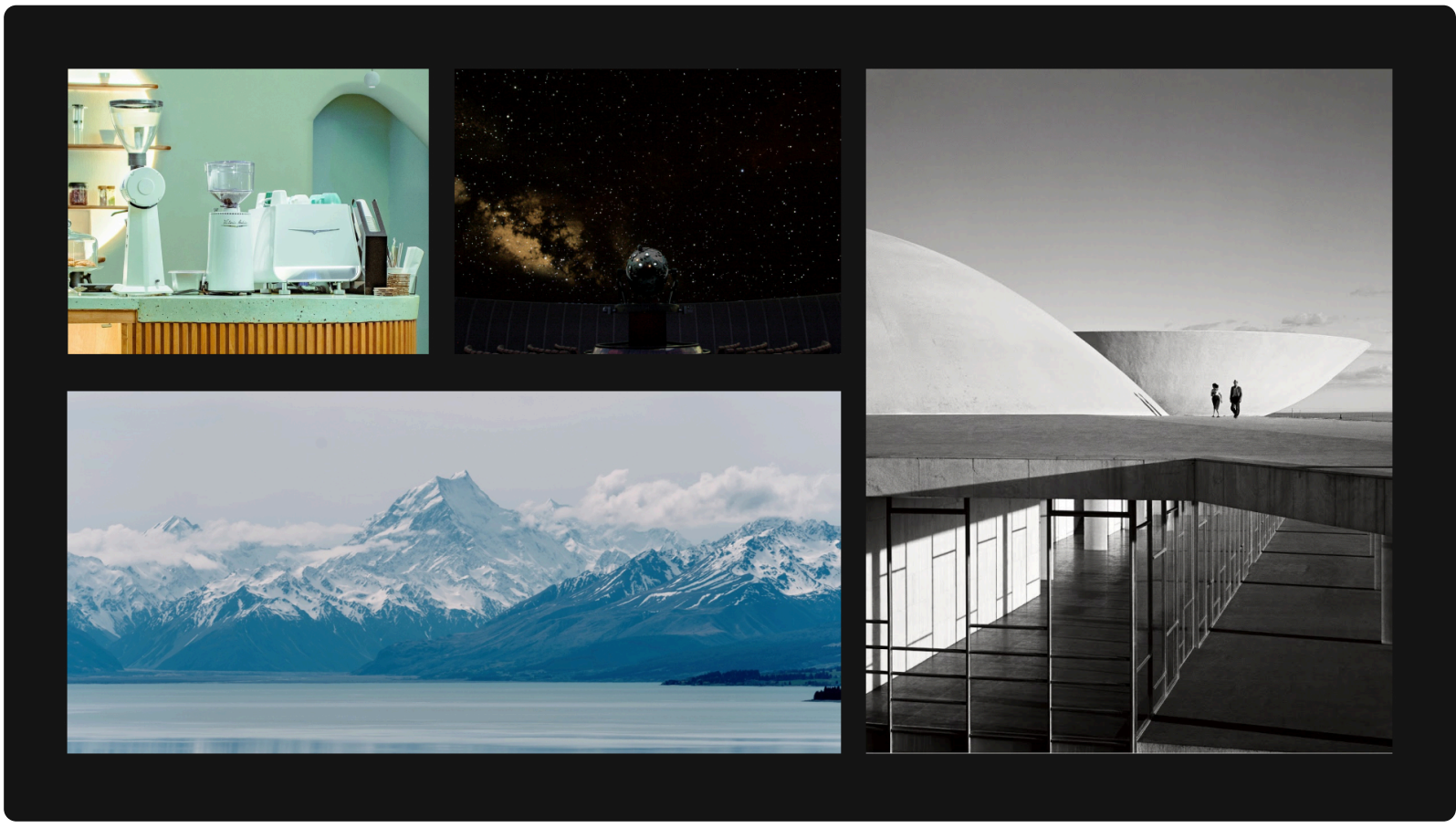


Photo collection of places that inspires us

People

- Ayrton Senna for unparalleled work ethic and pursuit of the truth
- Clarice Lispector for the craft of translating the questions of the soul in words
- Dave Grohl for being a resilient musician and a model of perseverance and creativity
- Lex Fridman for his love of humanity and curiosity in exploring new topics
- Steve Jobs for the relentless pursuit of perfection and intuitive design that combined functionality with elegance

Photo collection of people that inspires us

Documentaries

- F for Fake for the storytelling masterpiece, that blurs reality and illusion
- Jiro Dreams of Sushi for striving to elevate his craft even after decades of experience
- The Last Dance for showing what a team can achieve when motivations are aligned
- The Startup Kids for shaping and sharing the stories of young entrepreneurs

Photo collection of documentaries that inspires us

Objects

- Herman Miller for ensuring longevity without sacrificing design and comfort
- iPhone for being the tool that changed how we interact with the world
- Le Creuset for their timeless craft that meets culinary inspiration
- Mouse for the bridge between human and the virtual world
- Polaroid for its blend of tech and art, and its ability to instantly capture moments

Photo collection of objects that inspires us

Games

- Ghost of Tsushima for the incredible attention to detail
- Horizon for the stunning visuals, rich storytelling and the immersive experience
- Metal Gear Solid for being one of the best stealth games of our generation
- The Legend of Zelda: A Link to the Past for teaching how to design around a feeling: the childhood exploration

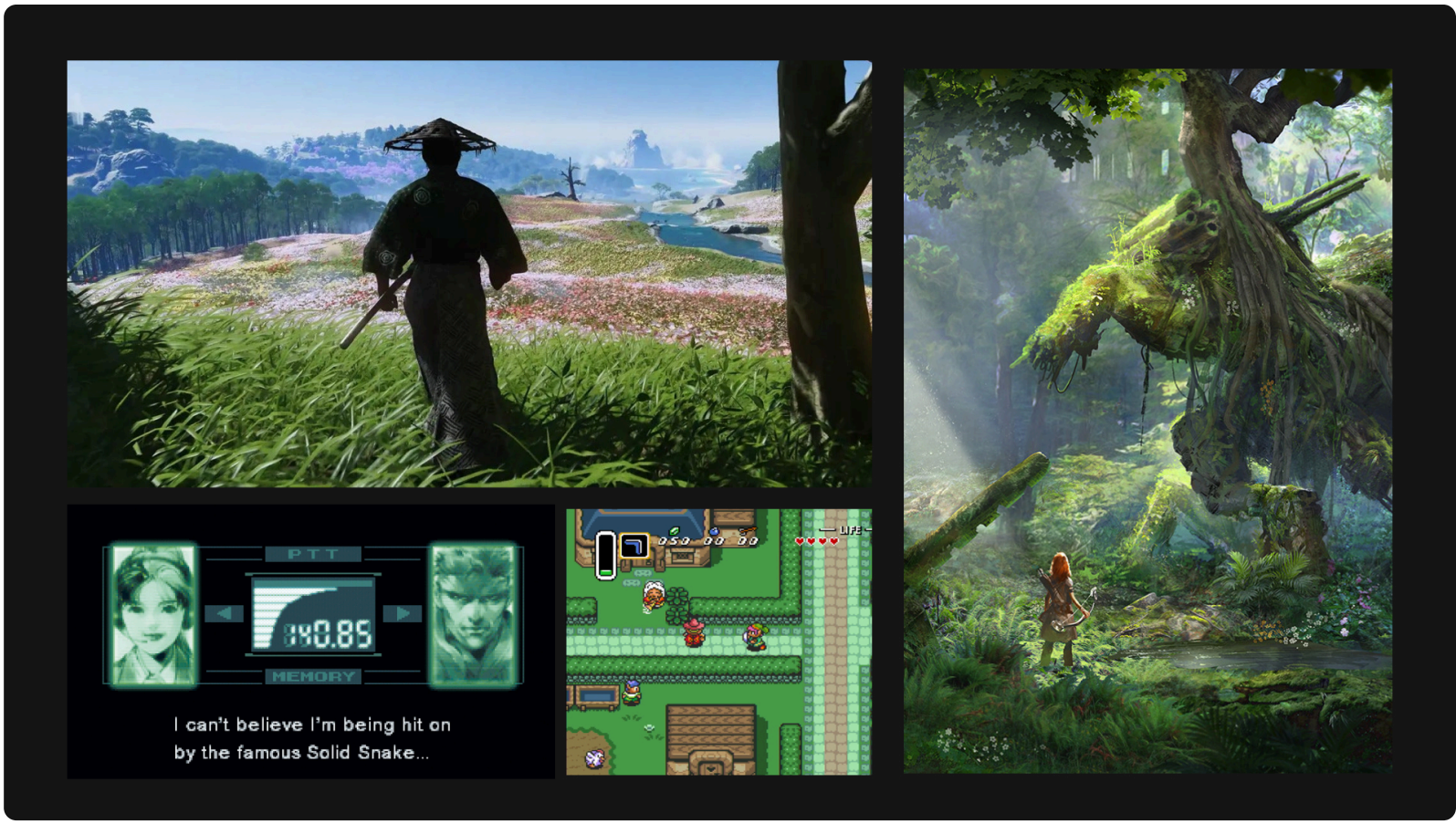


Photo collection of games that inspires us

How we got here

Aug 2022: Reimagining email

Bu and I were using multiple email tools, but they all felt outdated, slow, and built for marketers, not developers.

We also had to deal with emails landing in the spam folder and felt the pain of building a beautiful email template that would work everywhere.

So, we started playing with the idea of building the “Stripe of Email” as a side project.

Zeno Rocha (left) and Bu Kinoshita (right) writing the first lines of code for Resend

Sep 2022: First customer

Once we had a working MVP, we shared it with a friend who was working at another startup. He liked it and started using it in production. We sent him a $10 payment link as a way to validate if the product was really good or not.

To our surprise, he paid (!), which gave us the confidence to apply to Y Combinator.

Nov 2022: Getting into YC

After pitching the idea to Gustaf, Harj, and Diana, they finally accepted our application!

We immediately quit our jobs and moved to San Francisco to join the YC Winter 2023 batch.

Dec 2022: Open source launch

We launched react.email because we were frustrated by how difficult it was to build modern email templates that worked well across all email clients.

This project caught the attention of a lot of people, and it became clear that this was just the tip of the iceberg.

Jan 2023: Hello world

Resend was incorporated in January 2023, and we shared the news online.

The response was overwhelmingly positive, and the announcement tweet alone had more than 1M views.

This was also the month Jonni Lundy joined our founding team as Operations Manager.

Introducing Resend

Why are we doing this? What problem are we solving? Wh...

resend.com/blog/introducing-resend

Mar 2023: First launch week

We organized the first Resend launch week and shipped seven new features in seven days.

Launch Week 1: Wrap Up

All about the seven new features we launched in the last s...

resend.com/blog/launch-week-1-wrap-up

Apr 2023: $3m seed round

After finishing Y Combinator, we raised a $3m seed round (see full list of investors).

We also had our first team gathering in Lisbon, Portugal.

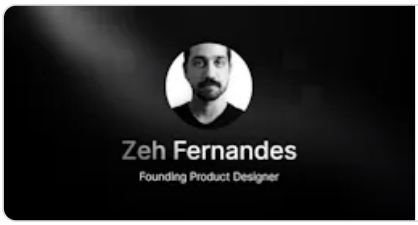
Resend raises $3M

Who invested on us. Where we're going. Why are we doin...

resend.com/blog/resend-raises-3m-seed-round

May 2023: Welcoming Zeh

We announced that Zeh Fernandes joined our founding team as Product Designer.



Welcoming Zeh Fernandes, our new Product Desig...

We're thrilled to share the news that Zeh Fernandes is join...

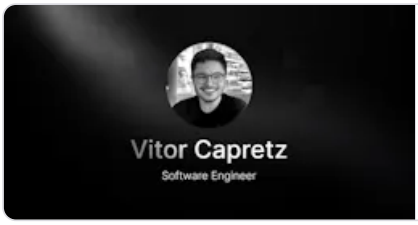
resend.com/blog/welcoming-zeh-fernandes

Jun 2023: Public launch

After operating behind a waitlist, we finally opened the product to everyone.

Sep 2023: Welcoming Vitor

We announced that Vitor Capretz joined the team as a Software Engineer.



Welcoming Vitor Capretz, our new Software Engin...

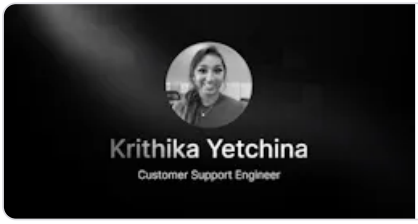
We're thrilled to share the news that Vitor Capretz is joinin...

resend.com/blog/welcoming-vitor-capretz

Nov 2023: Welcoming Krithika

We shared the news that Krithika Yetchina joined the team as Customer Support Engineer.

React Email also reached 10,000 GitHub stars that month.



Welcoming Krithika Yetchina, our new Customer S...

We're thrilled to share the news that Krithika Yetchina is jo...

resend.com/blog/welcoming-krithika-yetchina

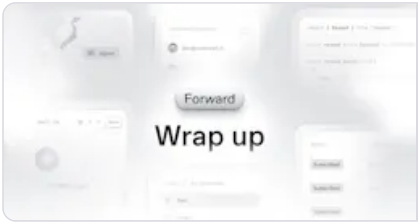
Dec 2023: 1,000 paying customers

We celebrated 1,000 paying customers as we closed the first year of Resend as a company.

Jan 2024: Looking “Forward”

We organized our second launch week, and this time, we called it Forward.

This marked the launch of marketing emails and turned Resend into a multi-product company.



Resend Forward: Wrap Up

All about the new features we launched last week.

resend.com/blog/resend-forward-wrap-up

Mar 2024: Investing more in security

After months of work, Resend became SOC 2 Type II compliant.

Resend is SOC 2 Type II compliant

Our journey to becoming compliant and what it means for ...

resend.com/blog/soc-2

Apr 2024: First 100,000 users

After 15 months since officially starting Resend, we reached 100,000 users.

What's next after 100,000 users

How we got here and where we're going next.

resend.com/blog/what-is-next-after-100-000-users

Jun 2024: Opening our strategy to the world

After months of work, we launched our handbook and shared 35 articles on how we approach engineering, design, support, and marketing.

We also announced that Felipe Volpone joined the team as a Software Engineer.

Handbook

The process behind Resend.

resend.com/handbook

Jul 2024: Improving data privacy

Resend became GDPR compliant and we wrote about the journey to get there.

We also shared that Brian Kerr joined the team as a Customer Success Engineer.

Resend's journey to GDPR compliance

Our path to complying with GDPR and continuing our com...

resend.com/blog/gdpr

What we believe

Do work that inspires others

You can't inspire people by doing an “okay” job.

To inspire, you need to do **phenomenal work**.

Quality is not a plus; it's a **must-have**. We know nothing will ever be perfect, but that doesn't stop us from pushing the limits.

We don't want to just meet the bar of our peers. We want to **raise the bar**. The sum of all the **small details** is what makes something special.

No ego

Helping others is part of the job.

It's okay to admit you don't know something and **ask for help** - it's a sign of strength, not weakness. We work together as an async team and **assume positive intent**.

We communicate in a kind, direct, and **transparent way**.

Keep shipping

Speed is key.

We prioritize ruthlessly, have a strong **sense of urgency**, and **make decisions fast**.

We constantly review the scope of a project and work toward a v0, not a v1.

We **ship early**, and we **ship often**.

Default to action

Regardless of what you do, we are all makers, and **makers don't wait**.

When we see a problem, we don't expect someone to fix it or tell us what to do. Instead, **we take initiative**, find creative ways around it, and send a pull request to solve it.

Remember - **no problem is too small**.

Why we exist

To help humans communicate

Humans are great at many things, especially communication.

If it weren't for **our ability to tell stories**, share knowledge, and collaborate with each other, we wouldn't have made it this far as a species.

Communication is not only the key to our survival, but also the **key to our progress** and growth. It's through communicating ideas that we've been able to build societies, technologies, and achieve things that were previously unimaginable.

Resend exists to **make human communication easier**. We do that through the internet and the protocols that surround it, like email.

To help developers build better products

Developers are at the **center of every product**. And you don't need to have "engineer" in your job title to be one.

Without them, there would be no product to speak of. They are the builders, the problem solvers, the creators of our digital world. They are all makers, and **makers don't wait**.

When they see a problem, they **don't expect someone to fix it**. Instead, they take initiative, and send a pull request to solve it. By creating a platform tailored to these people, we can scale the quality of communication across the entire industry.

Resend exists to **help developers build better products**. We want to provide the modern tools they need to navigate complex use cases.

To inspire others to design beautiful software

The world is full of crappy products.

Products that do not work as expected. Products that are difficult to understand and slow to use. Products that are **simply not good**. We want to change that.

When we say “design beautiful software”, we do not mean only building products that are aesthetically pleasing. For us, a software that is **easy to use is beautiful**. A product that is **fast is beautiful**. A product that does what it is meant to do is beautiful.

Resend exists to **design beautiful software that inspires** others. And you can't inspire people by doing an “okay” job. To inspire, you need to do phenomenal work.