Concise

New Employee Onboarding

v.1.0.0

2025

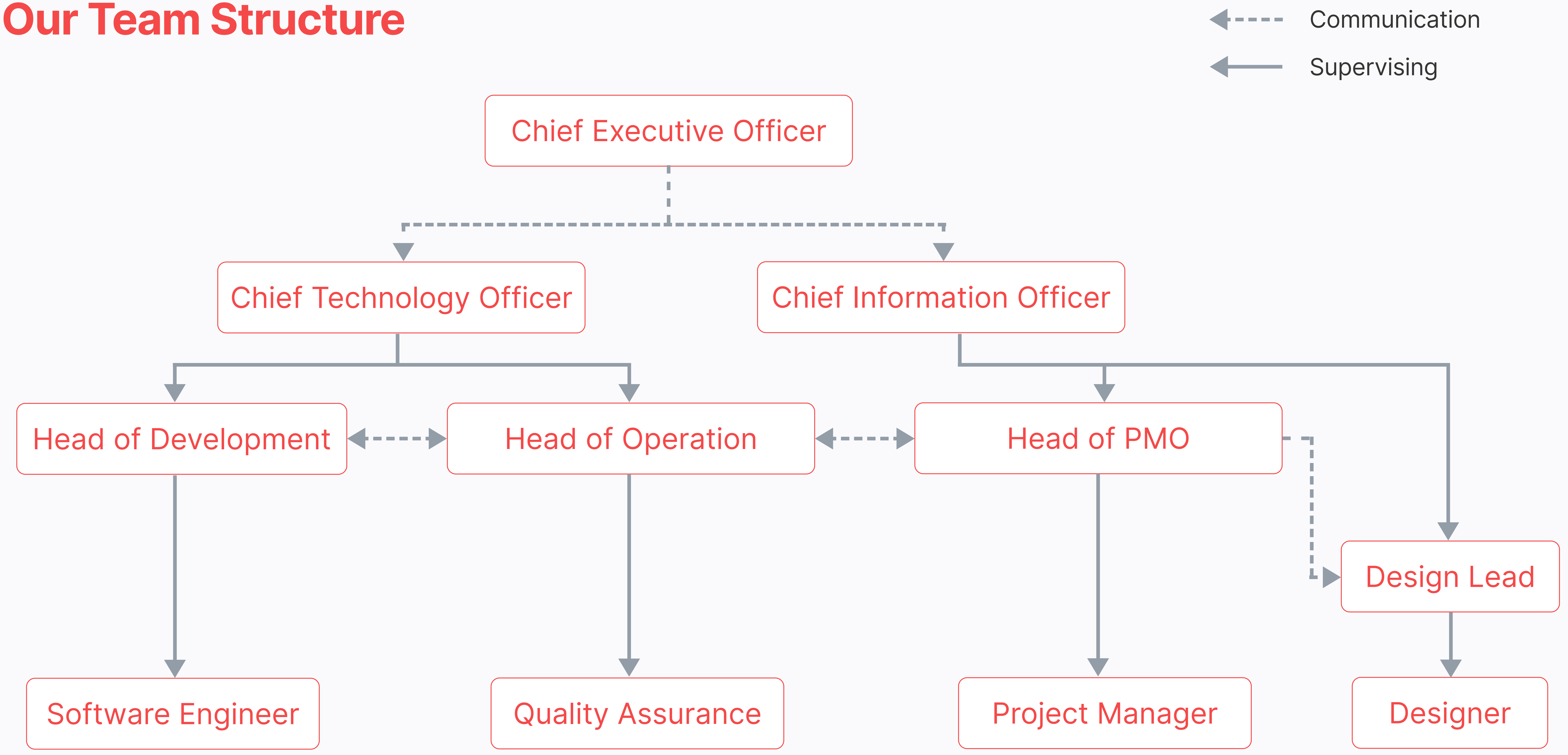
A couple of points

1. Introduction
2. Our Development Workflow (SDLC)
3. Front End & Back End Boilerplate

Introduction

...

Our Team Structure



Try Pitch

Our Development Workflow (SDLC)

...

Overview of SDLC (Software Development Life Cycle)

Planning

Understanding what the client or business needs and translating it into features.

Design

Planning the architecture, creating mockups, and setting up workflows.

Development

Writing clean, efficient, and well-documented code.

Testing and Deployment

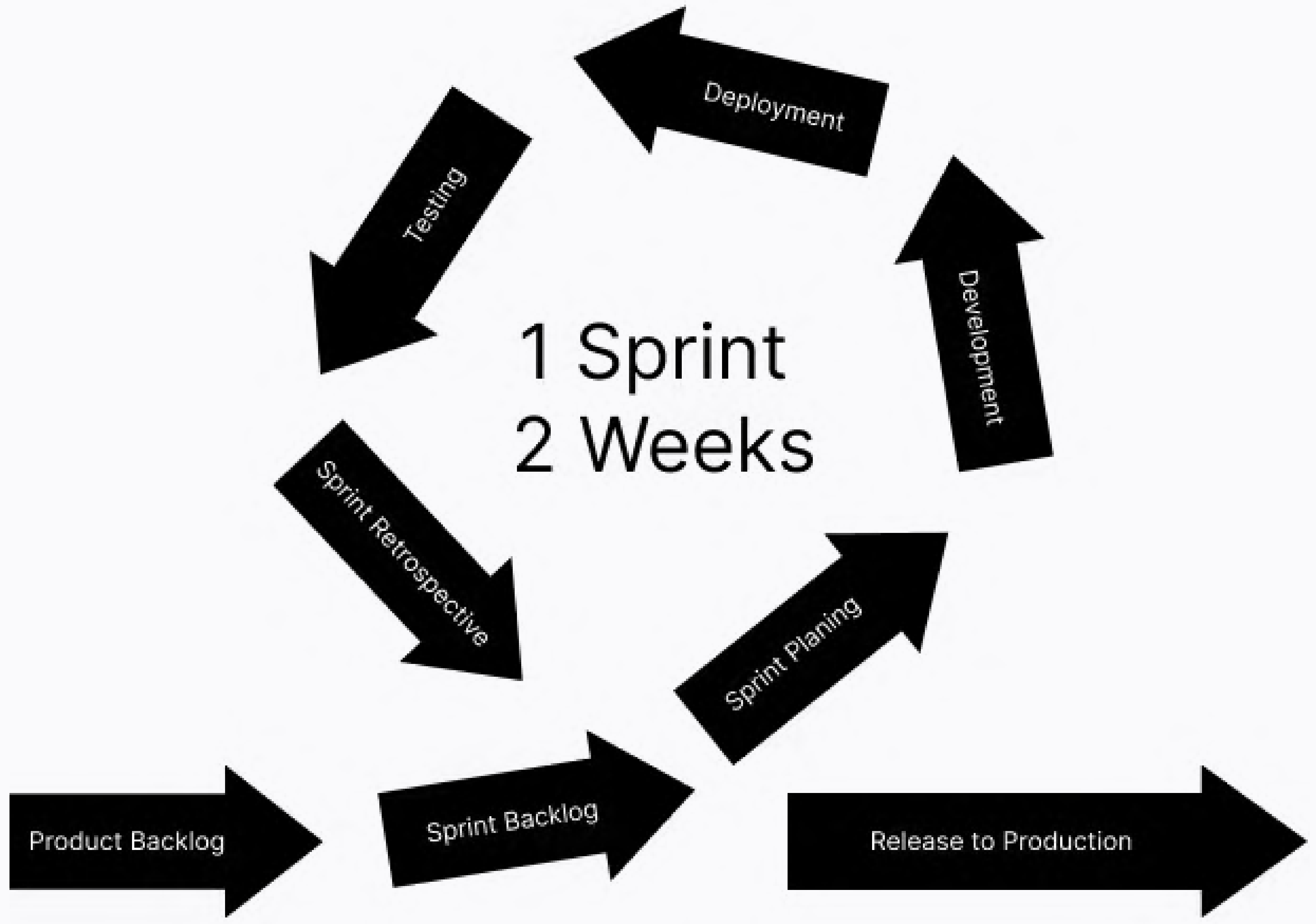
Ensuring quality through unit tests, integration tests, and UAT (User Acceptance Testing).

The process of releasing finalized features to the production environment, ensuring minimal impact on users and system availability.

Maintenance

Ensure smooth operation of live projects and performing regular backups to safeguard data.

Overview of TDLC (Technical Development Life Cycle)



Overview of TDLC (Technical Development Life Cycle)

1. Sprint Backlog
An activity where the PM and Tech Lead discuss what tasks will be worked on for the next two sprints. This is typically done one day before sprint closing (retro & planning).

2. Sprint Planning
An activity where all team members gather to discuss and explain the tasks that will be worked on during the sprint.

3. Development
An activity where designers and developers work on their respective tasks. During this phase:

- Designers create designs for features to be worked on in the next sprint.
- Front-end and back-end developers work on feature development and integration within the LAN environment.
- Developers are required to do the Pull Request to branch development, triggering an automatic deployed to LAN

Overview of TDLC (Technical Development Life Cycle)

4. Deployment

Once front-end and back-end integration is complete:

- Both front-end and back-end developers submit PRs to branch stg.
- The Tech Lead reviews the code to ensure it adheres to **concise** standards.
- Upon approval, the code is merged and deployed to the staging environment (STG).

5. Testing

After the deployment phase is completed, QA then conducts testing, including regression testing (both automated and manual) and give the report before the sprint ends.

6. Sprint Retrospective

The sprint ends with a retrospective, where all team members gather to review the work completed during the sprint. Usually, after the sprint retrospective, sprint planning for the next sprint is conducted on the same day.

Testing & Deployment

...

Roles

- Dev Ops
- Tech Lead
- Programmer
- QA

Procedures

1. Dev Ops will initialize or prepare 2 environments:
   a. Development
      This Environment will be frequently updated by tech lead & programmers, where version update is obligatory.
   b. Staging
      This Environment will be primarily used to test any versions of deployment.
2. Tech Lead initialize or prepare code repo for the project, including initializing 3 main branches:
   a. main
   b. staging
   c. development
3. Programmer do tasks assigned to them and push to branches that branched out from development branch
4. When tasks are already in the state of "In Review", Programmer may show result to Tech Lead or Tech Lead may test it out first before Tech Lead marking the task as "Done"

Procedures

5. When all tasks in that branch is done, Programmer have to make a pull request to initiate merging into development branch
6. Tech Lead will do PR Review
   a. if there are merge conflict, Tech Lead have to fix it
   b. if there are no issue, may proceed on merging the PR
7. After merging branches, Tech Lead is required to create version tag to the last commit in the development branch as a checkpoint
8. Then Tech Lead have to create releases from that version tag, so that we can keep track on the change logs from each version
9. For now to deploy to corresponding server, need the help of Dev Ops

Testing

According to Concise's QA SOP, each project must have a document to list out every possible test cases for each corresponding versions which agreed upon in the beginning or design phase of the project

Then QA may create manual testing or automated testing for those test cases.

QA will only test it out in Staging Environment with version information of frontend and backend provided

If by any chance QA found one or more bugs, they will create bug ticket, set the criticality, then assign it to the project's Tech Lead or Project Manager

Then Tech Lead or Project Manager will assess the bug ticket, if it is valid then Tech Lead or Project Manager have to create a task linked to the bug ticket and assign it to a programmer

Bug Criticality Level Measurement

Critical

- Definition: Bugs that cause system-wide failures, severe data loss, or complete application inoperability. These bugs often affect a large number of users and must be fixed immediately.
- Examples: System crashes, security vulnerabilities, inability to access critical features.

High

- Definition: Bugs that significantly impair functionality, cause performance issues, or affect important features. These bugs need to be addressed promptly, but the system remains operational.
- Examples: Major feature malfunctions, significant performance degradation, issues affecting important business processes.

Medium

- Definition: Bugs that affect non-critical functionality or cause minor inconveniences to users. These bugs are less urgent but should be fixed in the regular development cycle.
- Examples: Minor feature malfunctions, cosmetic issues, small performance hits.

Bug Criticality Level Measurement

Low

- Definition: Bugs that have minimal impact on the system's functionality or user experience. These bugs are often addressed as part of routine maintenance or future development cycles.

Examples: Minor UI glitches, typos, rare edge cases with minimal impact.

Our Tools

...

Our Tools

1. **Discord:** Used for team communication, enabling quick discussions and updates.
2. **Jira:** Helps organize tasks, track progress, and maintain project workflows.
3. **Git Concise:** Managing code repositories, enabling version control, collaboration, and maintaining a structured development workflow. (you can access it here)
4. **Figma:** For designing and prototyping user interfaces, facilitating collaboration between designers and developers.
5. **Apidog:** For API design, testing, and documentation, streamlining backend and frontend integration.

Notes: Please directly send your discord username to Edo

Git Flow Introduction

Helping you to GIT gud

What is GIT?

Source Code Management Tool

Platforms:

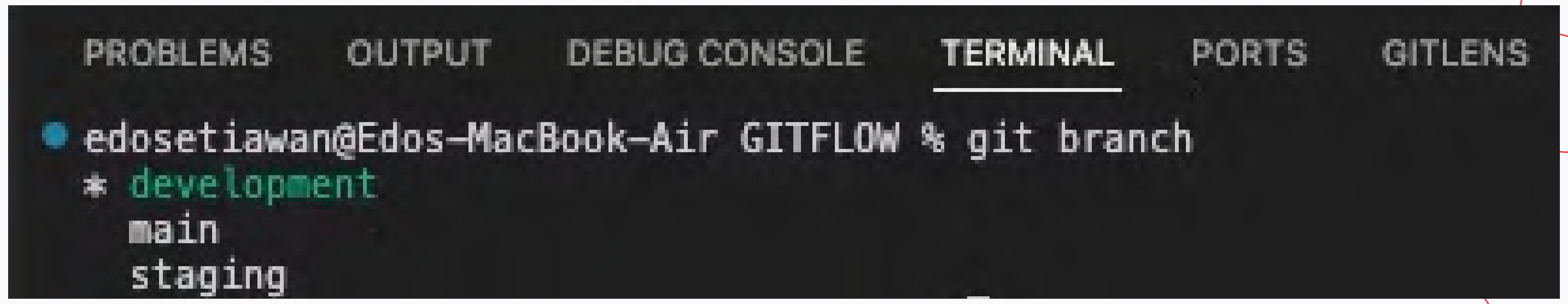
- github
- gitlab
- gitea
- and so on

GIT Commands?

- git init
- git add
- git commit
- git branch
- git push
- git pull
- and so on

Initialization

1. Prepare Code
   a. *git init*
2. Prepare Repo
   a. *git remote add origin PATH*
   b. *git push -u origin main*
3. Prepare Core Branches
   a. main
      i. automatically created after 2b
   b. staging
      i. *git checkout -b staging*
      ii. *git push --set-upstream origin staging*
   c. development
      i. *git checkout -b development*
      ii. *git push --set-upstream origin development*



```
edosetiawan@Edos-MacBook-Air GITFLOW % git branch
* development
  main
  staging
```

What is Git Flow ?

Git flow is a popular Git branching strategy aimed at simplifying release management.

For more references:

- http://danielkummer.github.io/git-flow-cheatsheet/
- https://gist.github.com/JamesMGreene/cdd0ac49f90c987e45ac

Git Flow Proper Procedure

Other than 3 Core branches, You can fork other branches from development branch with name that specify the purpose of that branch, for example:

1. Features

   feature/foo
2. Bugfixes

   bugfix/bar
3. Hotfixes

   hotfix/vX.X.X
4. ...

   ...

Git Flow Proper Procedure

Assuming you start your project with version 0.0.0
then you are given task to do feature User
which is supposed to be deployed in version 0.1.0
and may consist of multiple subtasks such as:

- Backend
  - Create User API
  - List User API
  - ...
- Frontend
  - User Table Page
  - List User API Integration
  - ...

Git Flow Proper Procedure

1. Create branch

   git checkout -b feature/users

2. Convert Coffee into Code
3. Add Changes

   git add .

4. Commit Changes

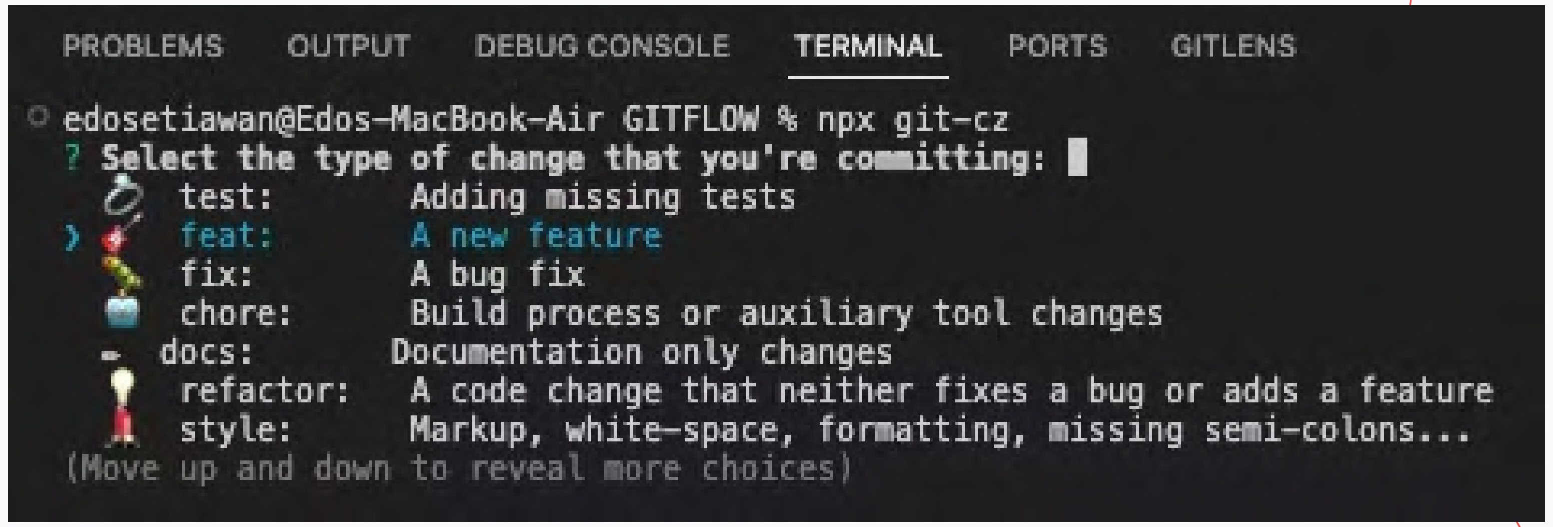
   ~~*git commit -m 'MESSAGE HERE'*~~

   npx git-cz

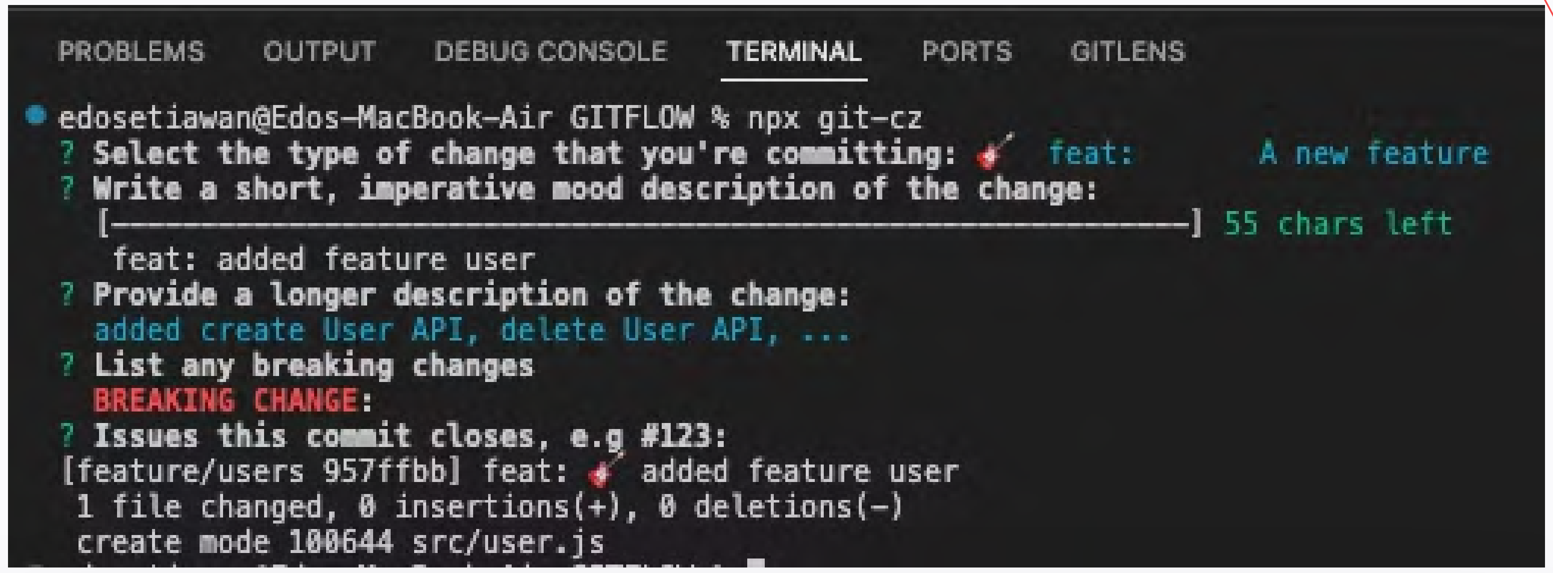
5. Push Changes

   git push

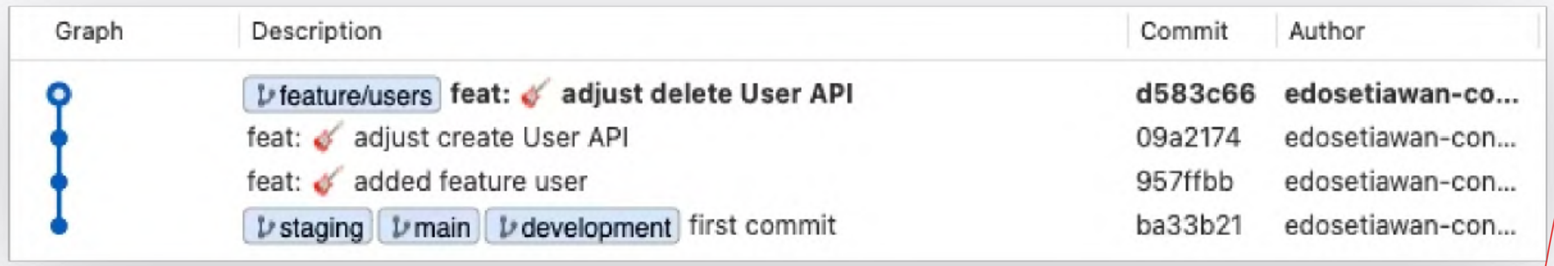
6. Still have tasks related to this branch ? repeat #2 : go to #7
7. ~~Procrastinate~~ Do other branch's tasks



```
edosetiawan@Edos-MacBook-Air GITFLOW % npx git-cz
? Select the type of change that you're committing:
    test:       Adding missing tests
  › feat:       A new feature
    fix:        A bug fix
    chore:      Build process or auxiliary tool changes
    docs:       Documentation only changes
    refactor:   A code change that neither fixes a bug or adds a feature
    style:      Markup, white-space, formatting, missing semi-colons...
(Move up and down to reveal more choices)
```



```
edosetiawan@Edos-MacBook-Air GITFLOW % npx git-cz
? Select the type of change that you're committing:   feat:       A new feature
? Write a short, imperative mood description of the change:
  [-----------------------------------------------------------] 55 chars left
   feat: added feature user
? Provide a longer description of the change:
  added create User API, delete User API, ...
? List any breaking changes
  BREAKING CHANGE:
? Issues this commit closes, e.g #123:
[feature/users 957ffbb] feat:  added feature user
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 src/user.js
```

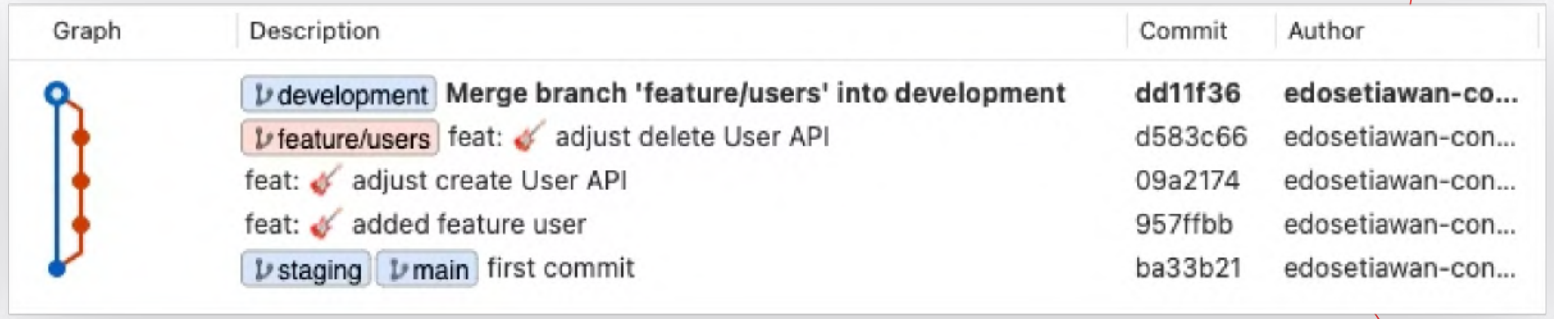


Graph	Description	Commit	Author
	feature/users **feat: adjust delete User API**	**d583c66**	**edosetiawan-co...**
	feat: adjust create User API	09a2174	edosetiawan-con...
	feat: added feature user	957ffbb	edosetiawan-con...
	staging main development first commit	ba33b21	edosetiawan-con...

Git Flow Proper Procedure

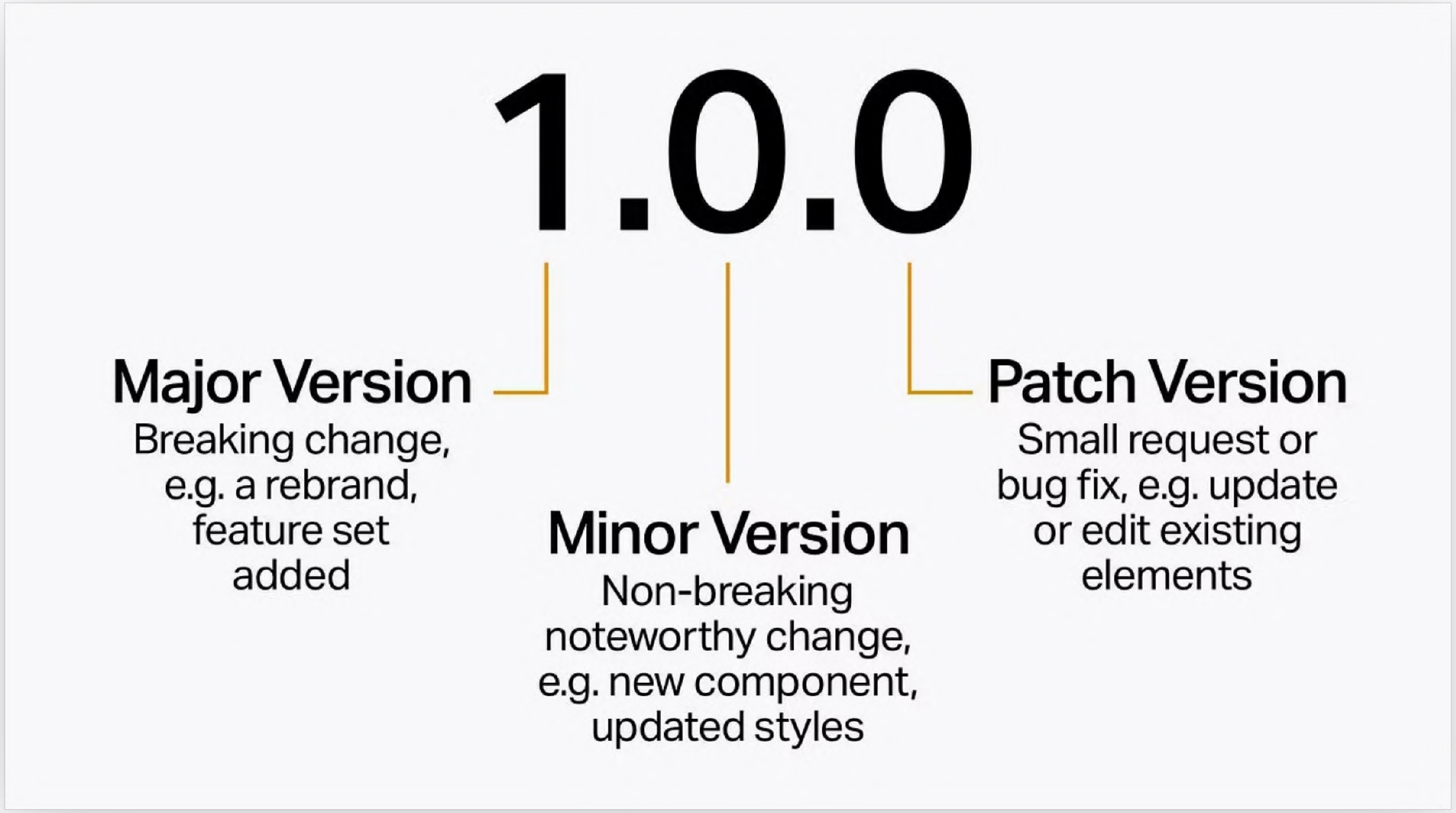
What to do when you have done all task in a branch?

- Create merge request to development branch



Graph	Description	Commit	Author
	development **Merge branch 'feature/users' into development**	**dd11f36**	**edosetiawan-co...**
	feature/users feat: 🎸 adjust delete User API	d583c66	edosetiawan-con...
	feat: 🎸 adjust create User API	09a2174	edosetiawan-con...
	feat: 🎸 added feature user	957ffbb	edosetiawan-con...
	staging main first commit	ba33b21	edosetiawan-con...

Semantic Versioning


1.0.0
Major Version
Breaking change, e.g. a rebrand, feature set added
Minor Version
Non-breaking noteworthy change, e.g. new component, updated styles
Patch Version
Small request or bug fix, e.g. update or edit existing elements

Git Tagging

How to Git Tag ?

- git tag -a VERSION

How to delete Git Tag ?

- git tag -d VERSION

Remember to push Tag to Repo

- git push origin tag VERSION

Semantic Versioning

X.X.X-devel

X.X.X-rc.X

X.X.X

Development	Staging	Production
0.1.0-devel	0.1.0-rc0	0.1.0
0.1.1-devel	0.1.0-rc1	0.1.1
0.1.2-devel	0.1.1-rc0	0.1.2
0.2.0-devel	0.1.2-rc0	0.2.0
0.2.1-devel	0.2.0-rc0	0.2.1
...	...	...

Front End & Back End Boilerplate and SOP

...

Front End Boilerplate

Our frontend boilerplate is designed to get you started quickly with a consistent structure and pre-configured tools.

- **React**: We use React for building dynamic, server-side-rendered web applications.

Resources

1. **Git - Front end - Web:** boilerplate-frontend
2. **Git - Front end - Mobile:** boilerplate-react-native
3. **Figma - Design Kit:** concise design kit

Back End Boilerplate

Our backend boilerplate is built to provide a solid foundation for building scalable APIs.

- **Node.js**: The backend is based on Node.js
- **Authentication**: Pre-configured JWT or OAuth2 authentication mechanisms.
- **Database**: We use ORMs like TypeORM or Sequelize for managing database interactions.

Resources

1. **Git - Back end - Node.js:** concise-boilerplate-backend
2. **Git - Back end - Java:** boilerplate-java

SOP Concise

SOP Concise (https://concise.co.id/sop)

- username: concise
- password: go2sop

Expectations from You

...

Please follow our company's regulation

Working Hour

Monday to Friday

Start from :

09.00 (max **09.30**)

Stand up :

09.30

Please don't be late...

also, We want You to

Collaborate & Communicate

Good communication is key. Join stand-ups, share updates regularly, and ask for help when needed.

Code Quality

Deliver clean, maintainable, and scalable code. Make sure your work is well-tested and properly documented.

Be Proactive in Learning

Keep up with new technologies and help improve our development practices.

Thank You

Concise

in case you feel lost, you can contact

keshia: keshia.tiffany@concise.co.id

fauzi: fauziats@concise.co.id | +62 851 7957 6176

edo: edo.setiawan@concise.co.id | +62 878 1823 9408