

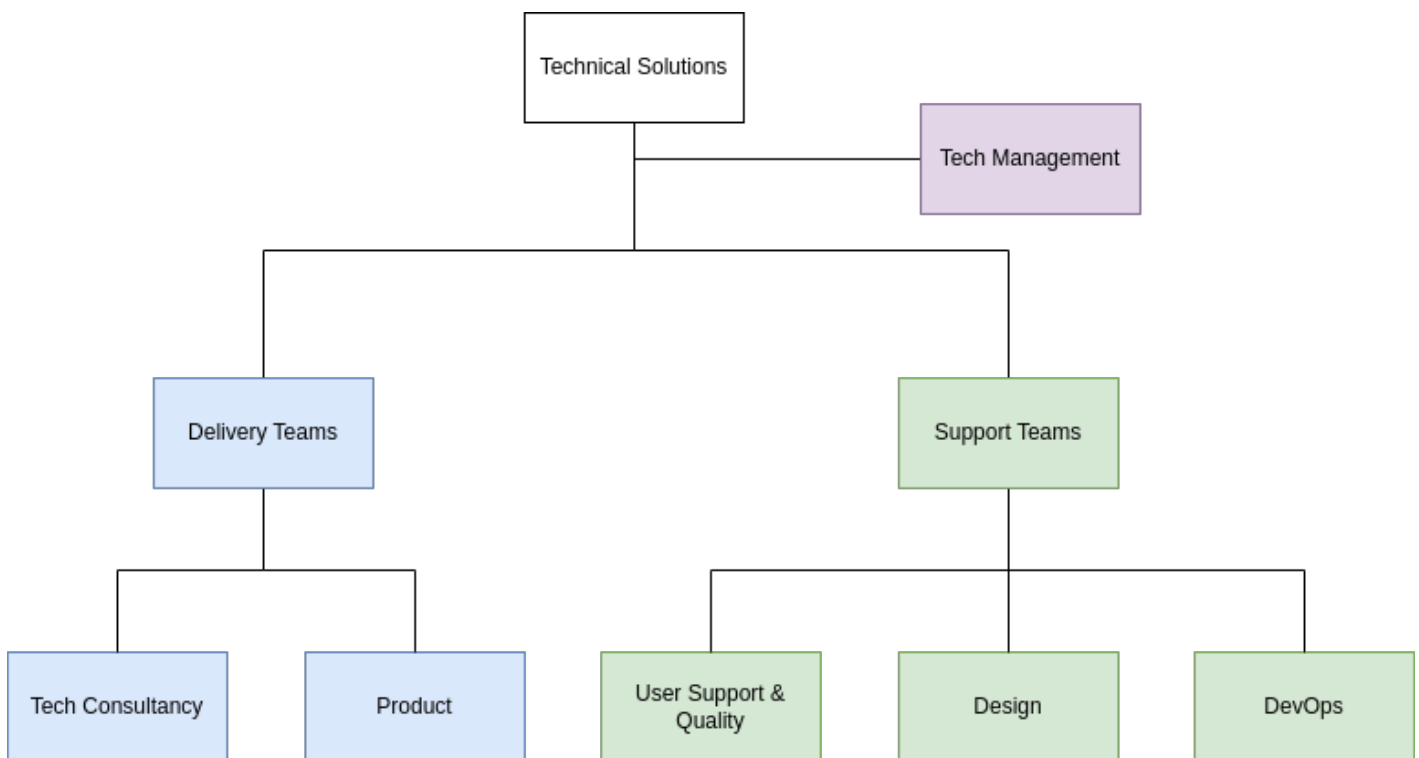
# Tech Team Operations

- Operating Model
- Documentation Guidelines
- Password Management Guidelines [TBC]
- Asana
  - Asana For Internal Team Coordination
  - Asana For Project Execution [TBC]
  - Objective & Key Results
- Leave Request Process
- Developer Guidelines

# Operating Model

## Team Structure

The Technical Solutions team is unique in the sense that it federates a very wide range of skills and expertise (inherent complexity of building software). Due to our team's size, some domains of expertise are led by a single person, resulting in one-person teams. We see this as a temporary situation but find it necessary to formalize these teams as the services they provide are essential to our collective success.



The Technical Solutions team is composed of 2 types of teams:

- **Mission Teams** - They execute the core operational goals of Akvo, i.e. build software for impact.
- **Support Teams** - They provide centralized domain expertise that is required for the Mission teams to achieve their goals.
- **Hybrid Teams** - They execute the mission and provides support capacity to the organization

Each team is described in a table with the following attributes:

- **Mandate** - A summary of what the team does at/for Akvo

- Offers - The services that the team offers
- Needs - The information, tools and services needed for that team to deliver its function
- KPIs - The KPIs that will be used to measure the performance and success of the team. In some instances, we've also described a number of metrics that reflect the volume of work.
- Lead - The person accountable for the team
- Coordination - The frequency with which the coordination meeting is held.
- Task Tracking - The place where work is tracked and coordinated (Asana).

# Mission Teams

## TC Team

Tech Consultancy Team	
Mandate	<ul style="list-style-type: none"> <li>• Support the BD team in developing and closing deals</li> <li>• Deliver high quality and on budget custom software solutions</li> </ul>
Offers	<div>BD</div> <ul style="list-style-type: none"> <li>• Explore partnership collaboration</li> <li>• ToR evaluation (Go, No Go)</li> <li>• Technical proposal writing (Budgets, Timelines)</li> <li>• Tech PoC for concepts provided by sector leads</li> </ul> <div>Delivery</div> <ul style="list-style-type: none"> <li>• Custom software solutions</li> <li>• Software handover activities</li> <li>• Maintenance and support</li> </ul>

## Tech Consultancy Team

### Needs

#### Information

- BD - ToR, context
- Delivery - ToR, context, user requirements

#### Tools

- Capacity Planner
- PMT
- Task Manager
- Code hosting: Github
- CI/CD Pipeline: Semaphore, CircleCI
- Unit testing: coveralls

#### Services

- Hosting
- Design
- Testing
- User support

Tech Consultancy Team	
KPIs	<p>Execution Efficiency</p> <ul style="list-style-type: none"> <li>• Definition: Ability to execute the scope within the allocated budget</li> <li>• Source: PMT</li> <li>• Target: 100%</li> </ul> <p>Team Utilization</p> <ul style="list-style-type: none"> <li>• Definition: Ratio between allocated work days and available days to work</li> <li>• Source: Capacity Planner</li> <li>• Target: 90%</li> </ul> <p>Team Billability</p> <ul style="list-style-type: none"> <li>• Definition: Ratio between allocated work days on client projects and available days to work</li> <li>• Source: Capacity Planner</li> <li>• Target: 75%</li> </ul> <p>Unit Test Coverage</p> <ul style="list-style-type: none"> <li>• Definition: % of the code covered by unit tests</li> <li>• Source: <u>Coveralls</u></li> <li>• Target: 80%</li> </ul> <p>Integration Test Coverage</p> <ul style="list-style-type: none"> <li>• Definition:</li> <li>• Source: Integration testing software</li> <li>• Target:</li> </ul> <p>User reported bugs</p> <ul style="list-style-type: none"> <li>• Definition: number of bugs reported by users after the software went into productions (by severity)</li> <li>• Source: Freshdesk</li> <li>• Target: less than 5% <u>critical bugs</u></li> </ul>
Lead	<u>Joy Ghosh</u>
Coordination	<ul style="list-style-type: none"> <li>• Project issue board on GH</li> <li>• 3 times a week tasks status / adhoc as required</li> <li>• Bi-weekly team status check</li> </ul>
Task Tracking	Each project has its own Asana Team or Project

# Product Teams

Product Teams	
Mandate	<ul style="list-style-type: none"><li>• Build, maintain and provide user support of current Akvo products (FLOW, RSR).</li><li>• Develop new products based on market needs.</li></ul>
Offers	<ul style="list-style-type: none"><li>• Product development</li><li>• Product management</li></ul>
Needs	<p>Information</p> <ul style="list-style-type: none"><li>• Market data</li><li>• User feedback</li></ul> <p>Tools</p> <ul style="list-style-type: none"><li>• Capacity Planner</li><li>• PMT</li><li>• Task Manager</li><li>• Code hosting: Github</li><li>• CI/CD Pipeline: Semaphore, CircleCI</li><li>• Unit testing: coveralls</li></ul> <p>Services</p> <ul style="list-style-type: none"><li>• Hosting</li><li>• Design</li><li>• Testing</li><li>• User support</li></ul>

Product Teams	
KPIs	<p>Bug Reports</p> <ul style="list-style-type: none"> <li>• Definition: number of bugs reported by clients</li> <li>• Source: Freshdesk</li> <li>• Target: less than 5% <u>critical bugs</u></li> </ul> <p>User Engagement</p> <ul style="list-style-type: none"> <li>• Definition: This metric measures how often users are using the product and how engaged they are with it.</li> <li>• Source: TDB</li> <li>• Target: TBD</li> </ul> <p>Revenue</p> <ul style="list-style-type: none"> <li>• Definition: This metric tracks the amount of money generated by the product</li> <li>• Source: Contracts</li> <li>• Target:</li> </ul> <p>Customer Satisfaction</p> <ul style="list-style-type: none"> <li>• Definition: This metric measures how satisfied users are with the product.</li> <li>• Source: Satisfaction questionnaires</li> <li>• Target: 4/5 in average</li> </ul>
Lead	<u>Guillaume Deflaux</u>
Coordination	Every 2 weeks - <u>Guillaume Deflaux</u> <u>Zuhdil Kurnia</u>
Task Tracking	<u>RSR Project in Asana</u> <u>FLOW Project in Asana</u>

# Support Teams

## DevOps Team

DevOps Team	
Mandate	<ul style="list-style-type: none"> <li>• Secure internal and external systems</li> <li>• Provide best-in-class infrastructure for the services Akvo offers</li> </ul>
Offers	<p>Security for internal and external systems (Company-wide)</p> <ul style="list-style-type: none"> <li>• Implement security best practices for our internal tools (email, password management, etc...) and server deployments.</li> </ul> <p>Infrastructure (TC Platforms / Products)</p> <ul style="list-style-type: none"> <li>• Design</li> <li>• Deployment</li> <li>• Support &amp; Monitoring</li> <li>• Costing</li> </ul> <p>Knowledge Management &amp; Company Culture</p> <ul style="list-style-type: none"> <li>• Updated technical documentation (wiki) covering the above topics (SOPs, best practices, ...)</li> <li>• Provide DevOps &amp; Security trainings internally</li> <li>• Promote a DevOps &amp; Security culture</li> <li>• Domain technical watch (best practices, tools, ...)</li> </ul>



**DevOps Team**

**Needs**

Information

- Product/project software architecture plan (before development started)
- Product/project software architecture documentation
- Product/project system requirements and quickstart documentation
- Infrastructure budget allocation plan from product, project or BD team

Tools

- Hosting provider (Google Cloud, Contabo, Siteground)
- Code Version Control (Github)
- CI/CD Tools (SemaphoreCI, CircleCI, Github Action)
- Documentation tools
- Infrastructure orchestration and configuration management tools (Terraform, Ansible)
- Secret management tools (Hashicorp Vault, Vaultwarden)
- Mailing tools (Mailjet)
- Workflow automation (N8N)

Services

- Testing

DevOps Team	
KPIs	<p>Average Service Availability</p> <ul style="list-style-type: none"> <li>• Definition: The client facing service are available</li> <li>• Target: 99.9% over a quarter</li> <li>• Source: Uptime checks</li> </ul> <p>Change Success Rate</p> <ul style="list-style-type: none"> <li>• Definition: Changes made by the DevOps team do not result in any downtime or other issues.</li> <li>• Target: 90% over a quarter</li> <li>• Source: uptime checks, CI/CD</li> <li>• Note: downtime caused by deployments are not taken into account</li> </ul> <p>Cost</p> <ul style="list-style-type: none"> <li>• Definition: Amount that we spend on cloud services (mostly GCP)</li> <li>• Target: Yearly targets based usage of services ( <u>COGS</u>, cost per project...). Still TBD</li> </ul> <p>Team Satisfaction</p> <ul style="list-style-type: none"> <li>• Definition: The DevOps team should aim for a high level of satisfaction among internal teams who rely on DevOps support.</li> <li>• Target: Score at least 8 out of 10 on average</li> <li>• Source: Quarterly survey</li> </ul>
Lead	<u>Anjar Fiandriato</u>
Coordination	Every 2 weeks - <u>Guillaume Deflaux</u> <u>Anjar Fiandriato</u>
Task Tracking	<u>DevOps Project in Asana</u>

## Design Team

Design Team	
Mandate	<ul style="list-style-type: none"> <li>• Support TC and product teams in delivering designed-for-purpose applications</li> </ul>

Design Team	
Offers	<p>Requirements Gathering</p> <ul style="list-style-type: none"> <li>• Conduct user research</li> <li>• Collaborate with product managers, developers, and subject matter experts to define and prioritise user stories and product requirements</li> </ul> <p>Wireframing &amp; UI Design</p> <ul style="list-style-type: none"> <li>• Create wireframes, prototypes, and high-fidelity mockups that effectively communicate design concepts and interactions</li> <li>• Create and maintain design style guides and design systems that support consistent and efficient design</li> <li>• Provide corresponding HTML/CSS templates</li> </ul> <p>User Feedback</p> <ul style="list-style-type: none"> <li>• Continuously iterate on designs based on user feedback, stakeholder input, and usability testing</li> </ul> <p>Knowledge Management &amp; Company Culture</p> <ul style="list-style-type: none"> <li>• Updated technical documentation (wiki) covering the above topics (best practices, tools, ...)</li> <li>• Domain technical watch (best practices, tools, design trends, ...)</li> </ul>
Needs	<ul style="list-style-type: none"> <li>• User Research Tools</li> <li>• Design and Prototyping Tools</li> <li>• Collaboration Tools</li> <li>• Design Systems and Style Guides</li> <li>• Feedback and Iteration Tools</li> <li>• Training and Professional Development</li> </ul>

Design Team	
<b>KPIs</b>	<p><b>Process Indicators</b></p> <p>Projects use Design Process Checklist</p> <ul style="list-style-type: none"> <li>• Target: 80%</li> <li>• Source: Design Process Checklists</li> </ul> <p><b>Output Indicators</b></p> <p>% Completion of Design Process Checklist</p> <ul style="list-style-type: none"> <li>• Target: 100%</li> <li>• Source: Design Process Checklists</li> </ul> <p>% Successful Task Completion (prototypes)</p> <ul style="list-style-type: none"> <li>• Target: 80%</li> <li>• Source: Maze</li> </ul> <p>User Perceived Experience (prototypes)</p> <ul style="list-style-type: none"> <li>• Target: 4/5</li> <li>• Source: Maze</li> </ul>
<b>Lead</b>	<u>Ouma Odhiambo</u>
<b>Coordination</b>	Every 2 weeks - <u>Guillaume Deflaux</u> <u>Ouma Odhiambo</u>
<b>Task Tracking</b>	<u>Design Project in Asana</u>

## User Support & Quality Team

User Support & Quality Team	
<b>Mandate</b>	<ul style="list-style-type: none"> <li>• Be the first point of contact of our end-users and help them make the most of our platforms and products.</li> <li>• Support the TC and products teams in delivering well tested software applications.</li> </ul>

## User Support & Quality Team

### Offers

All the services are provided to Products and TC Platforms alike.

#### L1 Support

- First point of contact with users
- Triage issues and escalate if necessary
- Resolve issues that are related to the usage of the software (anything that is not a bug or a server issue)

#### User Trainings

- Create training curriculums
- Deliver trainings to end-users (remote/onsite)

#### Functional Testing

- Write test plans
- Execute test plans (automated or not)
- Document issues and create test reports

#### User Documentation

- Create and update user documentation inline with software release cycles.

#### Knowledge Management & Company Culture

- Updated technical documentation (wiki) covering the above topics (best practices, tools, ...)
- Domain technical watch (best practices, tools, ...)

User Support & Quality Team	
Needs	<div>Information<ul style="list-style-type: none"><li>• Platform/Product onboarding from the development teams</li><li>• Functional requirements documents</li><li>• User stories and personas</li><li>• Release schedules</li><li>• Test data</li><li>• Test environment</li></ul></div> <div>Tools<ul style="list-style-type: none"><li>• Helpdesk (Freshdesk)</li><li>• Documentation/Wiki (Freshdesk, Bookstack)</li><li>• Automated Testing (Selenium, Appium)</li><li>• Test Management (TestRail, TestLodge, Zephyr, Zebrunner)</li><li>• Bug Tracking(Github, Asana)</li><li>• Load Testing (Gatling, Locust, LoadRunner)</li><li>• User Feedback (Piwik, Hotjar, ProductBoard)</li></ul></div>

## User Support & Quality Team

### KPIs

#### L1 Support

- KPI
  - Definition: Number and % of tickets that breached the SLAs
  - Source: Freshdesk
  - Target: 0(%)
  - Definition: Level of satisfaction of users after the handling of their tickets
  - Source: Freshdesk
  - Target: 4/5 or higher
    - SLA breaches
    - Average User satisfaction Level
- Work Volume
  - Number of tickets received (Freshdesk)
  - Number / % of escalated tickets (L2+) (Freshdesk)
  - Average first response time (Freshdesk)
  - Average resolution time (Freshdesk)

#### User Trainings

- KPI
  - Definition: Declared level of satisfaction of trainees after a training on one of our platforms or products
  - Source: Post training questionnaire
  - Target: 4/5 or higher
    - Average trainee satisfaction Level
- Work Volume
  - Number of trainings delivered
  - Number of created training curriculums
  - Number of updated training curriculums

#### Functional Testing

- KPI
  - TBD
- Work Volume
  - Number of test plans written
  - Number of test plans executed
  - Number of test reports written

#### User Documentation

- KPI
  - Definition: The % of documentation sites that have content that cover 100% of the user facing features of platforms and products
  - Source: Documentation sites + platform / product release notes
  - Target: 100%

User Support & Quality Team	
Lead	<a href="#">Jonah Kisioh</a>
Coordination	Every 2 weeks - <a href="#">Guillaume Deflaux</a> <a href="#">Jonah Kisioh</a>
Task Tracking	<a href="#">Support &amp; Quality Project in Asana</a>

# Hybrid Teams

## Technical Solutions Management

Technical Solutions Management	
Mandate	<ul style="list-style-type: none"> <li>Align objectives of the Tech Solutions Team with the objectives of the rest of the company</li> <li>Improve the efficiency of the entire Tech Solutions Team</li> </ul>
Offers	<p>Strategic and technical orientations to sub teams, including tie-breaking based on suggestions. Internal initiatives to focus on:</p> <ul style="list-style-type: none"> <li>Resourcing</li> <li>Team / HR</li> <li>Tech Stack</li> <li>Methodology</li> <li>Tooling</li> <li>Quality Standards</li> <li>OKRs</li> </ul>
Needs	<p>Information</p> <ul style="list-style-type: none"> <li>Company strategy and objectives</li> <li>Feedback from sub teams</li> </ul>
KPIs	<p>Objective alignment</p> <ul style="list-style-type: none"> <li>Definition: The fact that the Technical Solutions sub team objectives are aligned with larger company objectives.</li> </ul> <p>Team Efficiency</p> <ul style="list-style-type: none"> <li>Definition: The fact that the Technical Solutions sub team objectives create efficiencies that affect the entire team.</li> </ul>
Lead	<a href="#">Guillaume Deflaux</a>



Technical Solutions Management	
Coordination	Monthly - <a href="#">Guillaume Deflaux</a> <a href="#">Joy Ghosh</a> <a href="#">Deden Bangkit</a>
Task Tracking	<a href="#">Internal Project in Asana</a>

# Coordination Mechanisms

Team coordination plays a big part in creating the right environment for our team to deliver great work and to create a positive dynamic. The team coordination approach described below aims to foster team building while boosting our collective efficiency, i.e. provide enough context for everyone to properly execute their work while doing so in a friendly, respectful and engaging manner.

This approach is purposely built for our current team. We should expect it to change should our team size and structure evolve.

## 1-to-1s Guillaume / Team Leads

In most instances these meetings are used both for people management and team coordination purposes.

Frequency	Every week
Duration	30 min
Purpose	<ul style="list-style-type: none"> <li>• Ensure that there is a space to discuss your life at Akvo.</li> <li>• Coordinate the definition and the execution of team roadmaps</li> <li>• Ensure coherence of team specific initiatives within the larger Tech Solutions Team</li> </ul>
Structure	<p>Manager-Direct 1-to-1</p> <ul style="list-style-type: none"> <li>• Workplate &amp; Projects</li> <li>• Growth</li> <li>• Performance</li> </ul> <p>Team Specific Discussion</p> <ul style="list-style-type: none"> <li>• Services</li> <li>• Initiatives</li> <li>• Roadmap</li> <li>• Performance</li> </ul>

# Tech Solutions All Hands

<b>Frequency</b>	Every 2 weeks
<b>Duration</b>	1h
<b>Purpose</b>	<ul style="list-style-type: none"><li>• Make sure that we can have a bit of “face time” all together in order to break the silos created by our day-to-day</li><li>• Create a space for direct communications with the entire team (email remains the official channel for team wide announcements)</li><li>• Create an experience sharing space to foster ideation and innovation within the Tech Team</li></ul>
<b>Structure</b>	<p>Think Tank Thursdays (30 min)</p> <ul style="list-style-type: none"><li>• A rotating presentation by someone from the team.</li><li>• Any topic that is (vaguely) related to our work</li><li>• Any format allowed as long as there’s at least 10 min for questions. The more interactive the better.</li></ul> <p>All Hands (30 min)</p> <ul style="list-style-type: none"><li>• Team wide announcements</li><li>• Project / Product updates</li><li>• Team Specific Updates</li><li>• Updates from MC</li></ul>

# Documentation Guidelines

Our wiki is based on the Open Source application called [BookStack](#). Check the [fundamentals](#) and [official documentation](#).

## Why Do We Need Documentation?

Reason	Description
Knowledge Management	Documentation serves as a way to capture and preserve important information and knowledge within an organization. It helps ensure that critical information is not lost due to employee turnover or changes in leadership.
Training & Onboarding	Documentation can be used as a tool to train and onboard new employees. It provides a clear and concise source of information that helps new employees understand the company's policies, procedures, and processes.
Communication	Documentation can serve as a communication tool, ensuring that all team members are on the same page and understand what is expected of them. It can also help facilitate collaboration and knowledge sharing between team members.
Continuous Improvement	Documentation can help identify areas for improvement within a company's processes, policies, and procedures. It provides a record of past practices and helps identify opportunities for optimization and innovation.
Risk Management	Documentation helps manage risks associated with the company's operations. It provides evidence of past decisions and actions, which can be used to evaluate the effectiveness of risk management strategies.

# What Makes A Good Documentation?

By applying these principles, you will create useful and enjoyable documentation. Please, keep these in mind at all times while you are writing a page.

Principles	Description
Clear and Concise	The documentation should be written in a clear and concise manner, using simple language that is easy to understand. The use of jargon and technical terms should be kept to a minimum and explained when necessary.
Well-Organized	The documentation should be well-organized and presented in a logical sequence. It should have headings, subheadings, and an index to help the reader quickly find the information they need.
Accurate	The documentation should be accurate and up-to-date, reflecting the current state of things
Comprehensive	The documentation should cover all the important aspects of the system or product being documented, including features, functionalities, and limitations.
User-Centric	The documentation should be written with the end-user in mind, providing information that is relevant to their needs and expectations.
Visual Aids	The documentation should include relevant images, diagrams, and videos to help explain complex concepts and processes.
Consistency	The documentation should maintain consistency in its style and formatting, using the same terminology and language throughout.
Easy to Update	The documentation should be easy to update, so that it can be kept current things evolve.

Always use BookStack's Headings to structure the content of your pages. This automatically generates the table of content of your pages in the left pane.

## Content Organization

# The PARA Method

The content of our wiki is organized following the [PARA Method](#). It is based on 4 simple categories of content:

Category	Description
Projects / Products	<p><b>Definition:</b> Time-bound efforts that we are working on now.</p> <p><b>Akvo:</b> This is where we centralize the documentation about our current active projects and products.</p>
Areas	<p><b>Definition:</b> Long term knowledge areas that we want proactively manage.</p> <p><b>Akvo:</b> This is where we document our internal processes, SOP and knowledge about our areas of expertise.</p>
Resources	<p><b>Definition:</b> Topics or interests that might useful in the future.</p> <p><b>Akvo:</b> This is where we compile useful links and documents, reading lists, list of conferences, etc...</p>
Archive	<p><b>Definition:</b> Inactive items from the other 3 categories.</p>

## Content Nesting

BookStack offers 4 levels of content nesting:

- **Shelves** - Used for content categories (PARA).
  - **Books** - Used for top level content grouping.
  - **Chapters** - Used to organize low level content elements.
  - **Pages** - Used to organize low level content elements. Note that pages can also be directly below a book.

## Default Organization

### Projects / Products

Each project has a dedicated book, with the following structure:

- Project Sheet (page) - A table providing essential information about the project. Use the Project Sheet template.

- Technical documentation (chapter)??

Each product has a dedicated book with the following elements:

- Product Sheet (page)
- Technical documentation (chapter)??

## Areas

Each support team has a dedicated book.

# Archiving Content

To archive content you simply need to move it to the Archive shelf. Make sure to add the `[archive]` prefix to the title of every archived content (book, chapter, page).

# Password Management Guidelines [TBC]

# Asana

How we use Asana at Akvo to get things done.



# Asana For Internal Team Coordination

New to Asana at Akvo? Check out the [Asana - Fundamentals](#) wiki.

In addition to using Asana for coordinating and managing the execution of client work, we also use it to coordinate the internal work specific to the Tech Team. We use a number of Asana Projects grouped in Asana Teams.

## Naming Convention

All Asana Teams used to group the Asana Projects are use the following naming convention: `Team - Tech - <project group name>`.

## Project Organization

### Team - Tech - General

☐ **Asana Team Privacy Setting:** Public to organization

☐ **Asana Project Privacy Setting:** Shared with Team - Tech - General

This is the Asana Team that contains the main internal projects for the Tech Team.

Project	Description
Leave Requests	The project for leave request validation.

Project	Description
Professional Development	The list of courses that everyone is taking as part of the professional development goals. This is public so that everyone can see what others are learning and eventually create small cohorts.
DevOps Roadmap	The DevOps roadmap, <i>backlog</i> section is where we keep any big ideas for upcoming quarters.
DevOps - Daily Task	Ad hoc tasks that ensure our internal hosting capabilities are top notch. Any task related to our operation and supporting team will be logged here.
Support & Quality	Tasks related to the development of support & quality skills and services.
Design	Tasks related to the development of design skills and services.
Internal	Tasks related to short-term internal projects.

# Asana For Project Execution

## [TBC]

New to Asana at Akvo? Check out the [Asana - Fundamentals](#) wiki.

Estimated + Actual Hours => To create a feedback loop

1 Asana Team per client project

Each Asana Team should at least have 2 Asana Projects:

- Timeline
- Sprint

For small client projects

- Describe the structure of the sprint board. Create a template for this in Asana.

For big client projects

- Describe the structure of the sprint board. Create a template for this in Asana
- Add a Backlog Asana Project. Describe the structure (sections) and to use it. Create a template for this in Asana
- Add a Feedback project with a form. Describe the structure (sections) and to use it. Create a template for this in Asana.

Boom!

# Objective & Key Results

## Introduction

Objectives & Key Results (OKR, alternatively OKRs) is a goal-setting framework used by individuals, teams, and organizations to define measurable goals and track their outcomes. We will use this framework to increase our focus and intentionality.

We use Goals feature in Asana to manage our team OKRs.

## Basic Principles

We stick to a 2-level goals hierarchy:

- The top level goals represent the Objectives.
- The sub-goals represent the Key Results.

Objectives are defined and tracked yearly.

Key Results are defined and tracked quarterly. KRs can be repeated (these KRs are generally important things that we need to track all year long, but for which the analysis makes sense at a higher frequency).

OKRs are public to the entire company.

The OKR management process is a collaborative process to which everyone is expected to contribute.

## Management Process

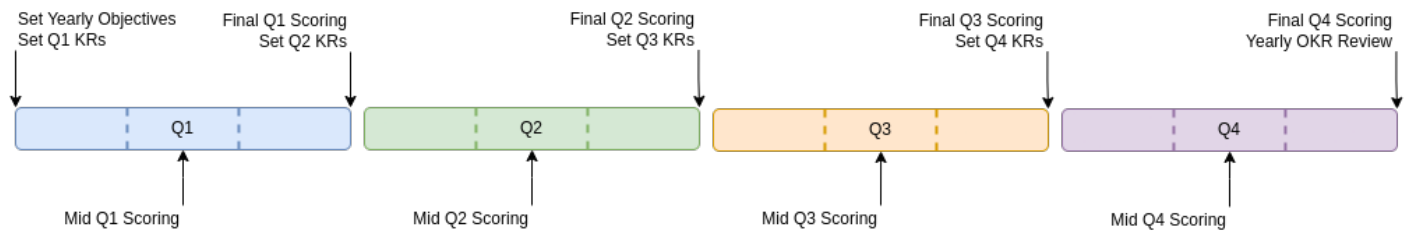
## Accountability

The CTO is ultimately accountable for the performance of the Tech Team and therefore accountable for the OKR process. Tech Team Management is responsible for its day-to-day execution.

Objectives and Key Results are owned by a single person. That person is accountable for achieving the target. The owner can share or delegate the execution with approval of Tech Team Management. That approval is required to make sure that people have enough capacity to do the work.

# OKR Management Calendar

The OKRs are managed following the calendar below.



Activity	Description
Set Yearly Objectives	<ul style="list-style-type: none"><li>• Tech Team Management sets the objectives based on the past year and the already identified goals and challenges for the upcoming year.</li></ul>
Set Q# KRs	<ul style="list-style-type: none"><li>• Tech Team Management sets the goals for the quarter, factoring the inputs from team members.</li></ul>
Mid Q# Scoring	<ul style="list-style-type: none"><li>• Each goal owner updates the scores that best reflects their current progress.</li><li>• The scores are reviewed and discussed at the following Tech Team Management Meeting.</li></ul>
Final Q# Scoring	<ul style="list-style-type: none"><li>• Each goal owner provides the final scores for the quarter.</li><li>• The scores are reviewed and discussed at the following Tech Team Management Meeting.</li><li>• A team-wide review takes place at the following All Hands.</li></ul>

Activity	Description
Yearly OKR Review	<ul style="list-style-type: none"> <li>• Tech Team Management compiles the data on OKRs and shares an analysis with the entire team.</li> </ul>

## Collaborative Nature Of The Process

The member of Tech Team Management directly manage the entire Tech Team. As such they are expected to socialize and proactively collect feedback on KR: suggestions to achieve current targets, suggestions for future KRs, etc...

All team members are expected to proactively participate in the OKR process by either providing direct feedback to members of the Tech Team Management during their 1-to-1 and during the planned team-wide discussion blocks such as the All Hands. Feedback and suggestions are welcome at any other time.

You can suggest KRs at **anytime** by adding a task in the [OKR Suggestions](#) project in Asana.

It's super simple:

- Create a task in the **Suggestions** section using the Key Results Template.
- Assign it to yourself so we know who made the suggestion.
- Select the objective. If you are not sure, you can pass.
- Fill in the description as explained in the task template.

## Managing OKRs With Asana

### Create An Objectives

Fill in the form as follows:

- **Title:** Short and clear statement
- **Goal Owner:** CTO
- **Accountable Team:** Team - Tech - General
- **Time Period:** FY##
- **Privacy:** Public
- **Members:** Team - Tech - General [Can edit] (automatically set once the Accountable Team is selected)

- **Parent Goal:** None
- **Update Method:** Automatic
- **Progress Source:** Sub-goals
- **Measurement:** Percent

The result should look something like below.


Add goal

What is a goal or key result you want to accomplish?


Goal title \*

Put A Human Being On The Moon

Goal owner

 Guillaume Deflaux


Company or accountable team

 Team - Tech - General


Time period ⓘ

FY24 Jan 1, 2024 – Dec 31, 2024

Privacy

 Public

Members ⓘ

 10 Team - Tech - General

Can edit

Parent goal

Connect a parent goal


How will you measure progress toward success?


[Learn to set measurable targets](#)


Update method \*

Progress source \*

Measurement \*

 Automatic

 Sub-goals

 Percent

Cancel

Save goal

Feel free to set a description for the objective once it is created.

# Create A Key Result

Fill in the form as follows:

- **Title:** Short and clear statement
- **Goal Owner:** Anyone in theory, Tech Team Management member in most cases
- **Accountable Team:** Team - Tech - General
- **Time Period:** Q# FY##
- **Privacy:** Public
- **Members:** Team - Tech - General [Can edit] (automatically set once the Accountable Team is selected)
- **Parent Goal:** Select the corresponding parent goal
- **Update Method:** Manuel
- **Measurement:** Choose the appropriate option
- **Current Value:** Input the starting value
- **Target Value:** Input the target value that we intend to achieve by the end of the quarter

The result should look something like below.



Add goal


×

What is a goal or key result you want to accomplish?

Goal title \*


Create A Bad A\*\* Rocket

Goal owner

 Guillaume Deflaux

×

Company or accountable team


 Team - Tech - General

×

Time period ⓘ


Q1 FY24 Jan 1, 2024 – Mar 31, 2024

Privacy

 Public

▼


Members ⓘ

 10 Team - Tech - General

×

Can edit ▼

Parent goal

 Put A Human Being On The Moon

×

How will you measure progress toward success?  
[Learn to set measurable targets](#)

Update method \*

Manual

Measurement \*

% Percent

Current value \*

0%

Target value \*

100%

Cancel

Save goal

## Scoring Scale

Once the goal is created, the description **must** be updated to specify the scoring scale of the KR.

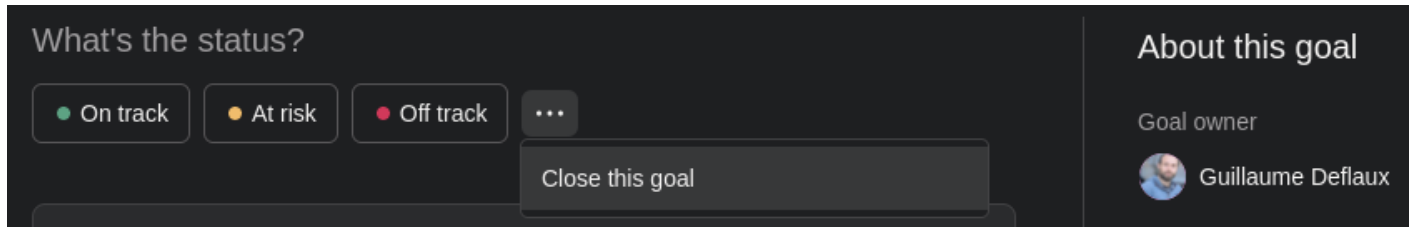
For the scoring scale we use a % scale, even for discrete targets, since in most cases some work is done even if the goal is not achieved. Each step builds on the previous one.

- 0% - Nothing done / dropped
- 30% - Did something, but not enough for change to materialize
- 50% - Goal partially achieved

- 70% - Goal achieved (actual target)
- 100% - Goal surpassed (good to think about what surpassing the goal is)

# Score A Key Result

Scoring KRs is done using the Status Update feature.



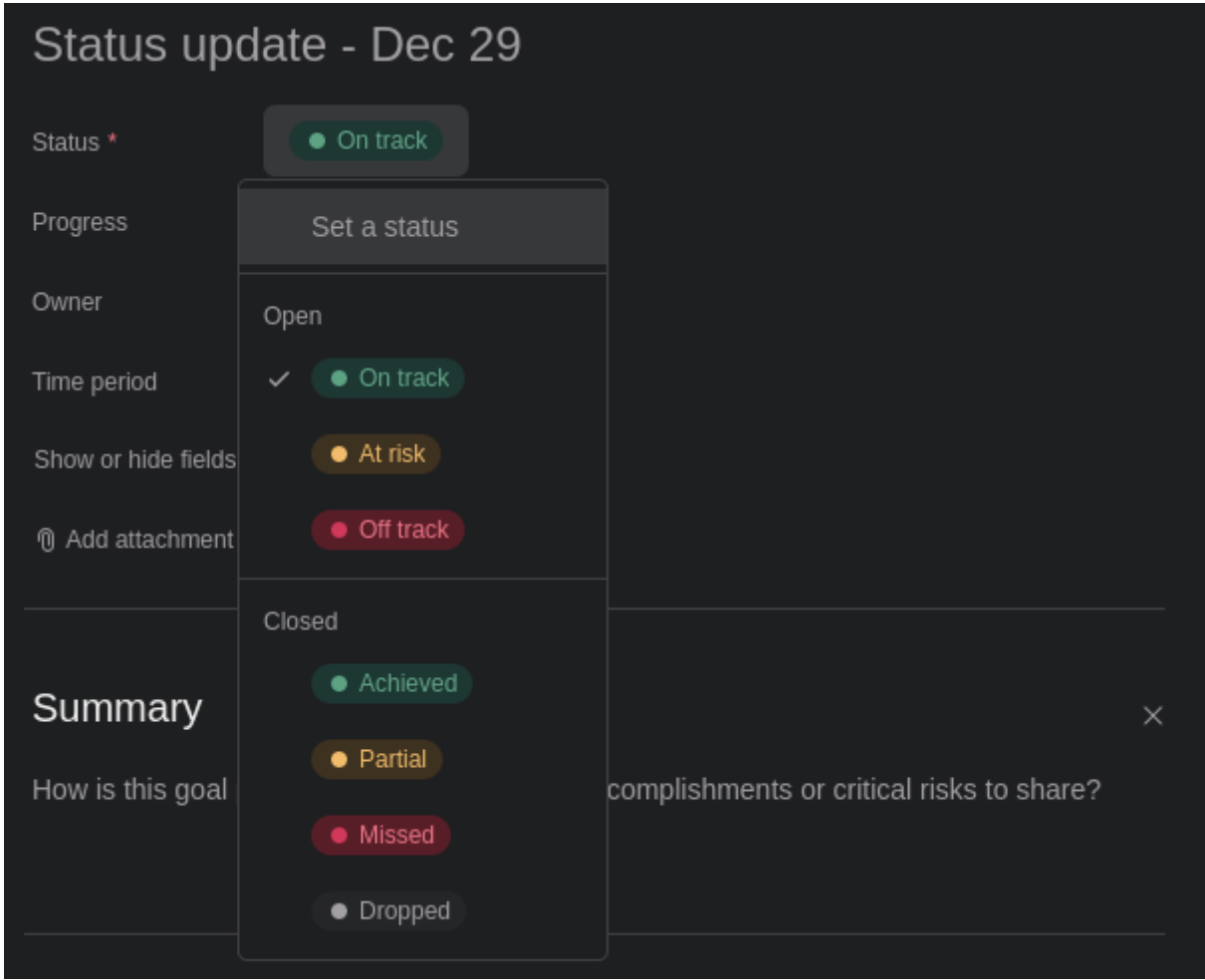
If you are performing the Mid Quarter Scoring, click on one of the highlighted statuses.

If you are performing the Final Scoring, click Close this goal.

Scoring a Key Result implies the following 3 steps.

## 1. Setting The Status

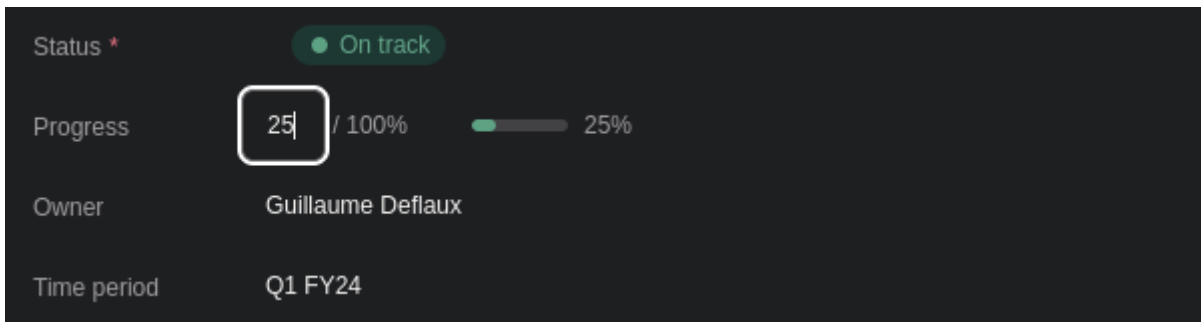
Select the appropriate status. See description below.



Status	Description
<b>Open Statuses</b> - Relevant for <u>mid quarter</u> scoring	
<input type="checkbox"/> On Track	You expect to achieve the target.
<input type="checkbox"/> At Risk	You think it's going to be difficult to achieve the target, but that it's still doable.
<input type="checkbox"/> Off Track	You don't think you will be able to achieve the target.
<b>Closed Statuses</b> - Relevant for <u>final</u> scoring	
<input type="checkbox"/> Achieved	All things considered, you think that the target was achieved.
<input type="checkbox"/> Partial	You did not achieve the target but did reasonable progress.
<input type="checkbox"/> Missed	You did not do substantive progress.
<input type="radio"/> Dropped	You decided to drop the target during the quarter.

## 2. Updating The Progress (Score)

Score the KR according to the on the scoring scale

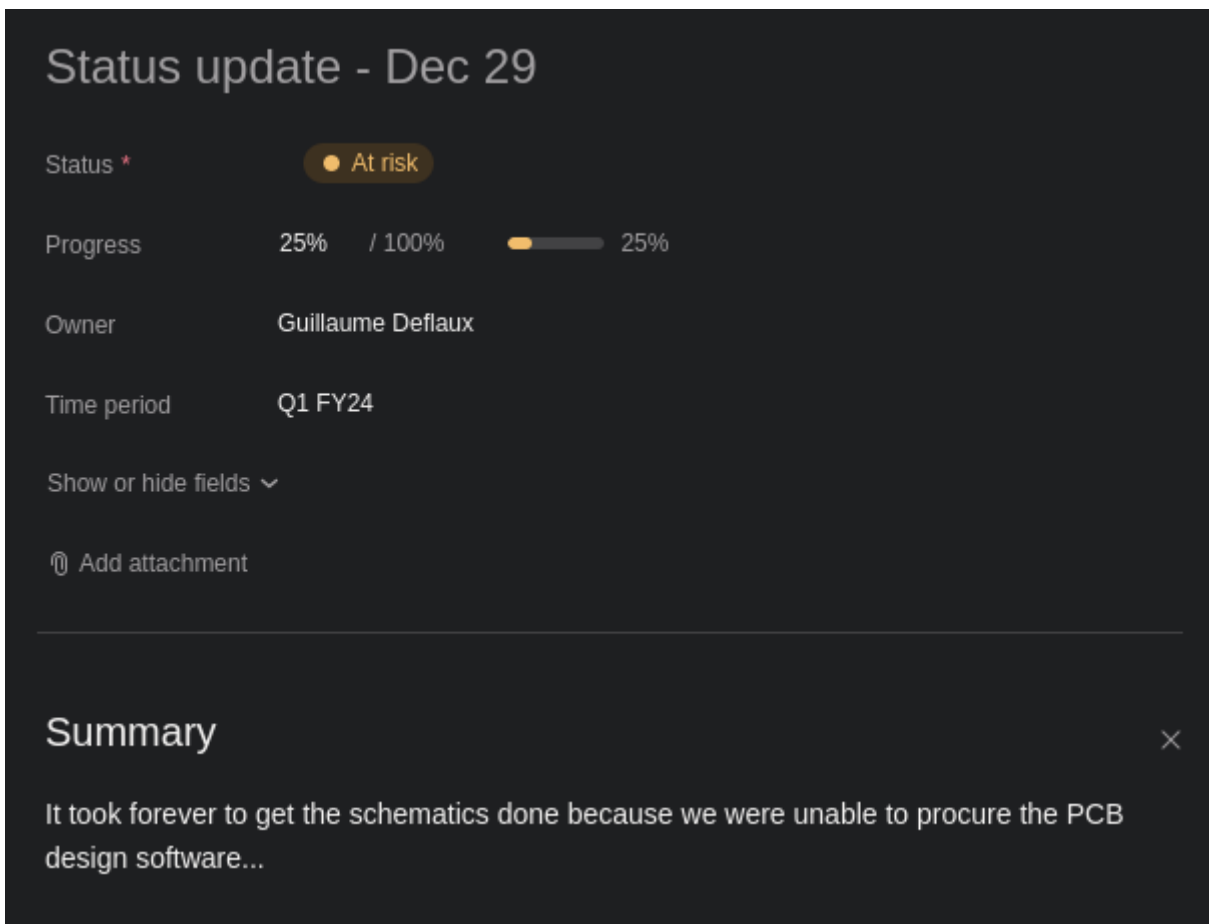


A screenshot of a progress update form. The status is 'On track' with a green dot. The progress bar is at 25% of 100%. The owner is Guillaume Deflaux and the time period is Q1 FY24. The progress value '25' is highlighted with a white box.

Status *	On track
Progress	25 / 100% 25%
Owner	Guillaume Deflaux
Time period	Q1 FY24

## 3. Provide Context

Fill in at least the Summary section.



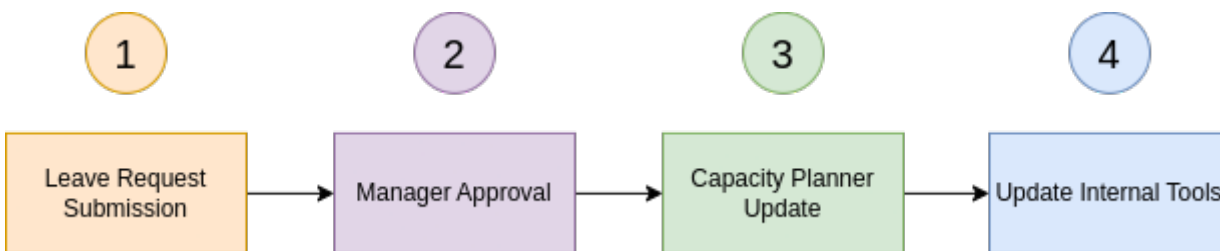
A screenshot of a progress update form. The status is 'At risk' with an orange dot. The progress bar is at 25% of 100%. The owner is Guillaume Deflaux and the time period is Q1 FY24. There is a 'Show or hide fields' dropdown and an 'Add attachment' button. A 'Summary' section is expanded, showing the text: 'It took forever to get the schematics done because we were unable to procure the PCB design software...'

Status update - Dec 29	
Status *	At risk
Progress	25% / 100% 25%
Owner	Guillaume Deflaux
Time period	Q1 FY24
Show or hide fields ▾	
📎 Add attachment	
Summary ✕	
It took forever to get the schematics done because we were unable to procure the PCB design software...	

# Leave Request Process

## Process Overview

All requests are managed in the [Leave Requests](#) Asana project.



## Who starts the process?

The person requesting leave.

## When should the request be sent?

As soon as this person knows s/he wants to take leave. Depending on the duration of your leave you are expected to send the request a in advance. This is important to make the team has enough to get organized.

Duration	Notice Period
$\leq 2$ days	1 week before the first day of leave
$3 < \text{days} \leq 5$	2 weeks before the first day of leave
$1 < \text{week} \leq 2 \text{ weeks}$	1 month before the first day of leave
$> 2 \text{ weeks}$	2 months before the first day of leave

## When is a request approved definitively?

When it is reflected in the Capacity Planner.

# Detailed Process

## 0. Talk To Your Project Managers (Asana)

Before submitting a Leave Request, we recommend you first talk to the Project Managers of the projects you are working in order to:

1. Inform them
2. Confirm that it will be possible from the perspective of projects.

PM validation is not an absolute requirement depending on the reason why you need leave.

## 1. Leave Request Submission (Asana)

Create a task in the Leave Requests Asana project in the *Manager to Approve* column. All fields are mandatory.

1. **Title:** The name of the requester
2. **Start Date:** First day the request will be on leave
3. **End Date:** Last day the request will be on leave
4. **# Working Days:** The number of working days included in the period of leave
5. **Assignee:** The requester's manager
6. **Description (optional):** Any additional context related to the request. Remember that the content of this task is public.

## 2. Manager Approval (Asana)

The manager approves the request based on the following criteria:

- PMs are aware of the request
- PMs have a plan in place to manage project deliverable during the leave period
- The leave request does not create issues with internal projects

The manager decision can be one of the following:

Decision	Action In Asana
Approve the request	Move the request to the <i>Capacity Planner To Update</i> column and assign the task to the Capacity Manager.
Ask for more information	Write a comment mentioning the requester and asking for additional information. Sensitive conversations should happen in direct Slack messages or during 1:1 meetings.
Deny the request	Add “Not approved” in the comment section. The request should be moved to the <i>Processed</i> column.

### 3. Capacity Planner Update (Asana)

The Capacity Manager attempts to update the Capacity Planner accordingly. The following scenarios are possible:

Scenario	Action in Asana
No issues updating the Capacity Planner	<p>The Capacity Manager:</p> <ul style="list-style-type: none"> <li>• Moves the request to the <i>Internal Tools To Update</i> column</li> <li>• Leaves a comment mentioning the requester indicating that the request has been taken into account.</li> <li>• Assigns the task to the requester</li> </ul> <p>⚠ <b>At this stage only is the request considered as accepted.</b></p>
A conflict appears in the Capacity Planner	The Capacity Manager leaves a comment mentioning the Manager. Discussions happen in Asana or on Slack.

### 4. Update Internal Tools

As the requester you are expected to update the internal tools to make sure other colleagues are aware / can find out that you have leave planned.

1. Announce your leave period in `#team-tech-leaves`.
2. Update the company Holiday calendar
3. Update your personal calendar. Make sure to use an Out Of Office block so that meetings are rejected automatically. This inform meeting organizers immediately.
4. Log your leave in Clockwise (employees only, not applicable to contractors).

Details for steps 2 and 4 are available [here](#).





# Developer Guidelines

## Version Manager

**Akvo** is using **GitHub** as main version control system. By using GitHub, we can ensure that our code-base is well-managed and that changes are thoroughly reviewed and approved before they are added to the code-base, ultimately improving the quality and stability of our software. Here are the most important rules of that have to be consider:

## Branch Protection Rules

- **Require pull request reviews before merging:** This rule requires that all changes to a branch be submitted as a pull request, and that at least one other team member approves the changes before they can be merged into the main branch.
- **Require status checks to pass before merging:** This rule requires that certain conditions be met before a pull request can be merged. For example, we require that all tests pass in the CI and certain code quality coverage metrics are met.
- **Require a minimum number of reviewers (at least one reviewers):** This helps to ensure that changes are thoroughly reviewed before merging.
- **Restrict who can push to the branch:** This rule limits who can make changes to a branch, helping to prevent accidental or unauthorized changes.

## Branch Naming

It's important to have a clear and consistent naming convention for your branches. A good naming convention for Feature Branches is to use the following format: `feature/<issue_number>-<issue_description>`. For example, `feature/13-backend-test-setup`.

Make sure that you have the issue number is available on **GitHub**

Here's what each part of the naming convention means:

- `feature/`: This is a prefix that identifies the branch as a feature branch.
- `<issue_number>`: This is the number of the issue or task that the branch is related to.
- `<issue_description>`: This is a brief description of the issue or task. It should be short but descriptive enough to give an idea of what the branch is about.

Using this naming convention makes it easy to identify which branches are related to which issues or tasks. It also helps to keep our branches organized and easy to manage.

# Pre-Commit Config

Pre-commit config is a configuration file used by the pre-commit framework to define a set of code checks, also known as "hooks," that are run before code is committed to a version control system. The pre-commit framework is a tool that allows developers to define and manage these hooks locally, providing a way to catch errors early and enforce coding standards.

Example: [\[#363\] Pre-commit initial config](#)

It is true that Continuous Integration/Continuous Delivery (CI/CD) pipelines can also catch errors and enforce coding standards. However, pre-commit hooks have some distinct benefits that make them complementary to CI/CD pipelines.

## Pull Request

In our projects, each team is relatively small, and developers often work independently. This makes PR (Pull Request) reviews critical, as they allow other developers to understand the project better, which is essential for cross-team knowledge and backup in case of absence. Although Deden (Lead Developer) is the primary person responsible for reviewing code, it is not mandatory that he handles every review.

### Benefits of Code Reviews

- They help ensure code quality, maintainability, and alignment with the project's goals.
- They allow the team to share knowledge about features and code structures, reducing bottlenecks when someone is unavailable.
- They encourage best practices, such as adhering to DRY, KISS, and YAGNI principles.
- They provide a learning opportunity for both the code author and the reviewer.

## As a Requester

### 1. Move the Task to the PR Review Section

Move the task related to the feature you're asking to be reviewed into the **PR Review** section on the Asana's current Sprint board (e.g. Sprint #2).

### 2. Provide the PR Link

Ensure that the GitHub PR link is added to the "GitHub" field in the Asana task.

GitHub  Add GitHub pull request

### 3. Assign a Reviewer

Assign the reviewer for your code in Asana. Talk to the Lead Developer on your project to confirm who to assign the PR too. Usually one person is appointed for a project, but we remain flexible based on the workload of said person and the timeliness needed for the review. Escalate to the Delivery Manager or CTO if you are blocked.

### 4. Assign the Reviewer in GitHub

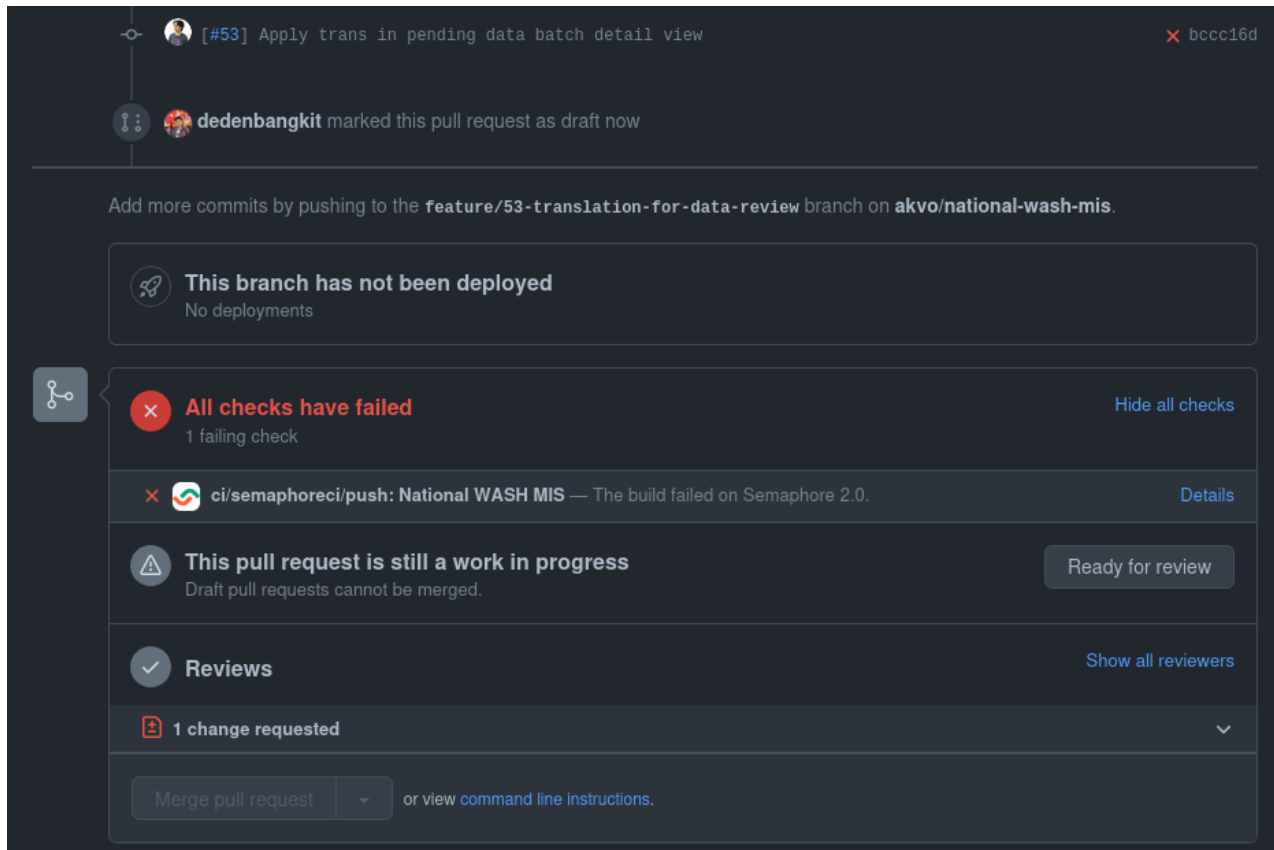
Don't forget to also assign the same reviewer in the GitHub PR.

## 5. Follow Up on Delayed Reviews

If the assigned reviewer hasn't been able to review your code after 2 days, report it in the relevant Slack channel, mentioning the reviewer. You can use **#team-tech-general** or the specific project's channel **#proj-<project-name>-tech**. Escalate to the Delivery Manager or CTO if you remain blocked.

## 6. Ensure CI Passing and Test Coverage

Make sure your code passes all CI checks. Additionally, unit tests are mandatory, or at the very least, integration tests (e.g., testing API endpoints) should be included if it's a back-end feature



## As a Reviewer

### 1. Check PR Readiness

Confirm that the code is marked as "Ready to Review" in both GitHub and Asana. If the PR is still in draft or missing the GitHub PR link in Asana, notify the requester to update it.

### 2. Ensure CI Status

If the PR hasn't passed the CI checks, reassign the task back to the requester and move the task to the **In Progress** section.

### 3. Understand the Feature's Context

Carefully read the task description in Asana, paying special attention to the goals and purpose outlined in the Tech AC (Acceptance Criteria) section or Low-Level Design. Check if there's a QA plan—if so, you might want to manually test the feature as well.

### 4. Run and Inspect the Code (if necessary)

For larger or more complex PRs, run the code locally. Also, review code quality by running linters or quality checks if required. Evaluate the code's adherence to best practices like.

## 5. Log Your Review Time

Track the time you spend reviewing the code. The review ideally should not take longer than an hour, log your time in the Asana task itself and **Clockwise (or General, with notes)** to accurately reflect the effort.

## 6. Check the Target Branch

Ensure that the PR is targeting the correct branch (usually `main`) since the QA team will test the code manually after it's deployed to the main branch.

# When the Review is Completed

## 1. Reassign to the Developer

Once the review is done, reassign the task back to the developer. Keep the task in the **PR Review** section on Asana.

## 2. Move to QA if Necessary

If you know who the QA person is, move the task to the **QA** section in Asana and assign it to the appropriate person. The QA is typically the Project Manager, who may not be highly technical. If there's nothing for them to test, you can simply mark the task as "Done."

## 3. Limit Feedback Cycles

It is desirable to keep the number of iterations to a minimum as each iteration can add delays to the development process. The author of the PR should ensure that they have done thorough testing and review before submitting the PR for review, and try to incorporate feedback as much as possible in a single iteration.

As a general rule of thumb, most teams aim to keep the number of iterations between 1 and 3

# Code Format

A consistent code format is important for several reasons:

1. **Readability:** A consistent code format makes it easier for team members to read and understand the code, even if they didn't write it themselves.
2. **Maintainability:** Consistent formatting makes it easier to maintain the code over time. When code is formatted consistently, it's easier to spot errors and to make changes to the code without introducing new errors.
3. **Collaboration:** Consistent formatting makes it easier for team members to collaborate on code. If everyone is following the same format, it's easier to understand each other's code and to work together to solve problems.

## Prettier - JavaScript

Some common best practices for JavaScript code formatting include:

- Using consistent indentation, such as two or four spaces per level
- Using **camelCase** for variable and function names
- Placing opening braces on the same line as the associated statement or declaration
- Using semicolons to terminate statements
- Using single quotes for strings, unless the string contains a single quote
- Using **===** and **!==** instead of **==** and **!=** for comparisons
- Limiting line lengths to 80-120 characters to improve readability.

Example: `.prettierrc.json`

```
{
  "trailingComma": "es5",
  "tabWidth": 2,
  "useTabs": false,
  "semi": true,
  "singleQuote": true,
  "printWidth": 80,
  "singleAttributePerLine": true
}
```

## Python - Black

- Indentation: Use four spaces per indentation level. Avoid using tabs or a mix of tabs and spaces for indentation.
- Line length: Keep lines of code to a maximum of 79 characters. If a line needs to be longer, break it into multiple lines.
- Naming conventions: Use lowercase letters for variable names, and separate words with underscores. Use **Capitalized** for class names, and **lowercase\_with\_underscores** for module names.
- Imports: Import one module per line, and place imports at the top of the file. Use relative imports for **intra-package imports**.
- Function and class definitions: Use two blank lines to separate function and class definitions from other code.
- Blank lines: Use blank lines to separate logical sections of code. For example, use a single blank line to separate method definitions in a class.

Example: `setup.cfg`

```
[flake8]
max-line-length = 79
inline-quotes = single
accept-encodings = utf-8
isort-show-traceback = True
```

```
# Excluding some directories:
```

```
exclude = .git,__pycache__
```

```
[tool.black]
```

```
line-length = 79
```

# Text Editor Setup

## Visual Studio Code

To use the Flake8 in VS Code, you can install the "Python" extension from the VS Code marketplace. This extension provides a built-in linter that can use the Flake8 configuration in your **setup.cfg** file. Once you have the extension installed, open a Python file in VS Code and the linter should automatically start providing feedback on your code.

To use the Black in VS Code, you can install the "Python" extension and the "Black" extension from the VS Code marketplace. Once you have both extensions installed, you can enable Black as the default formatter by adding the following line to your VS Code settings:

```
"python.formatting.provider": "black"
```

With this setting enabled, VS Code will automatically format your code using Black whenever you save a Python file. You can also customize how VS Code reads your setup.cfg file by adding the following line to your VS Code settings:

```
"python.linting.flake8Path": "/path/to/flake8"
```

To auto-format your JavaScript code using Prettier in Visual Studio Code (VS Code), you can install the Prettier extension from the VS Code marketplace. This extension provides automatic code formatting using the Prettier code formatter.

Open your VS Code settings by pressing Ctrl+, (Windows and Linux) or Command+, (macOS). In the search bar at the top of the settings window, search for "Prettier". Under **"Prettier: Config Path"**, enter the path to your **.prettier.json** configuration file. With these settings in place, whenever you save a JavaScript file in VS Code, the Prettier extension will automatically format your code according to the options specified in your **.prettier.json** configuration file.

## Vim / Neo-Vim

You can also use the Prettier and Black in Vim by adding the following lines to your Vim configuration file:

```
" Format code using Black and Prettier when saving
augroup autoformat
  autocmd!
  autocmd BufWritePre *.py :%!black -
  autocmd BufWritePre *.js :silent! %!prettier --stdin --semi --single-quote --no-bracket-spacing --tab-width 2
augroup END
```

## GNU Emacs

For Emacs, you can use the `before-save-hook` feature to run a command before saving the file. Add following lines to your `~/.emacs.d/init.el`:

```
;; Format Python code using Black when saving
(defun format-python-code-with-black ()
  (when (eq major-mode 'python-mode)
    (progn
      (call-process-region
        (point-min) (point-max) "black" t t nil "-")
      (save-buffer))))

;; Format JavaScript code using Prettier when saving
(defun format-javascript-code-with-prettier ()
  (when (eq major-mode 'js-mode)
    (progn
      (call-process-region
        (point-min) (point-max) "prettier" t t nil "--stdin" "--single-quote" "--no-bracket-spacing" "--tab-width=2")
      (save-buffer))))

(add-hook 'before-save-hook #'format-python-code-with-black)
(add-hook 'before-save-hook #'format-javascript-code-with-prettier)
```

Please ask questions or raise concerns if you're not sure how to follow the code format

## Quality Control

These standards is defined by **Project maintainer / Team Lead**, it should follow best practices and established coding conventions. There are many aspects to code quality, but some common factors include:

1. **Readability:** The code should be easy to understand and follow, with clear and consistent formatting and naming conventions.
2. **Maintainability:** The code should be modular and easy to modify, with clear separation of concerns and a well-defined structure.
3. **Efficiency:** The code should be optimized for performance and resource usage, with efficient algorithms and data structures.
4. **Robustness:** The code should handle errors and unexpected inputs gracefully, with appropriate error handling and testing.
5. **Security:** The code should be designed with security in mind, with appropriate measures to protect against potential vulnerabilities and attacks.

Example python code that following code quality standards:

```
def calculate_average(numbers):  
    if not isinstance(numbers, list):  
        raise TypeError("Input must be a list of numbers.")  
  
    if len(numbers) == 0:  
        return 0  
  
    total = sum(numbers)  
    return total / len(numbers)
```

This code is a function that calculates the average of a list of numbers. It follows some good coding practices, such as:

- Checking the input parameter to make sure it's a list before proceeding
- Handling the case where the input list is empty
- Using descriptive variable names that make the code easier to understand
- Using Python's built-in functions like `isinstance` and `sum` to write concise and readable code
- Raising a meaningful exception when the input is not of the expected type

On the other hand, here's an example of **bad code** that doesn't follow code quality standards:

```
def avg(num1, num2, num3):  
    if not isinstance(num1, (int, float)):  
        return "num1 must be a number"  
    if not isinstance(num2, (int, float)):  
        return "num2 must be a number"  
    if not isinstance(num3, (int, float)):  
        return "num3 must be a number"
```



```
sum = num1 + num2 + num3
average = sum / 3
return average
```

This code also calculates the average of three numbers, but it's written in a way that violates good coding practices, such as:

- Hard-coding the number of input parameters, which would require changing the code if we wanted to calculate the average of more or fewer numbers
- Returning a string message instead of raising an exception when the input is not of the expected type
- Using a variable name `sum` that's the same as a built-in Python function, which can cause confusion and errors
- Not handling cases where the input values might be invalid or lead to errors

There are also several principles that are widely recognized as important for writing high-quality code. Here are some of the most important ones:

## SOLID

SOLID is an acronym for a set of principles that were developed to guide object-oriented design. The principles are:

1. Single Responsibility Principle: Each class should have a single responsibility.
2. Open-Closed Principle: Classes should be open for extension but closed for modification.
3. Liskov Substitution Principle: Sub-types should be substitutable for their base types.
4. Interface Segregation Principle: Clients should not be forced to depend on interfaces they don't use.
5. Dependency Inversion Principle: High-level modules should not depend on low-level modules; both should depend on abstractions.

Code example:

```
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * (self.radius ** 2)

class Square:
    def __init__(self, side):
        self.side = side
```

```
def area(self):  
    return self.side * self.side  
  
shapes = [Circle(5), Square(10)]  
total_area = sum(shape.area() for shape in shapes)
```

This code defines two classes, `Circle` and `Square`, each with a single responsibility of calculating its own area. This adheres to the Single Responsibility Principle of SOLID. Both classes use a common interface (the `area()` method) that makes them interchangeable, which adheres to the Liskov Substitution Principle. The code is also open to extension (adding new shapes) but closed to modification, which adheres to the Open-Closed Principle.

Example of code that does not follow SOLID:

```
class User:  
    def __init__(self, name, email):  
        self.name = name  
        self.email = email  
  
    def save(self):  
        # Code to save user to database  
        pass  
  
class UserManager:  
    def __init__(self):  
        self.users = []  
  
    def add_user(self, name, email):  
        user = User(name, email)  
        self.users.append(user)  
        user.save()
```

This code defines two classes, `User` and `UserManager`. The `User` class has the responsibility of storing information about a single user, and the `UserManager` class has the responsibility of managing a list of users and saving them to a database. However, the `UserManager` class violates the Single Responsibility Principle by having both responsibilities of creating users and saving them to the database. This makes the code harder to maintain and test.

References:

- <https://towardsdatascience.com/solid-coding-in-python-1281392a6a94>
- <https://realpython.com/solid-principles-python>

# DRY - Don't Repeat Yourself

This principle states that we should avoid duplicating code and instead aim to write code that is reusable and modular.

Code Example:

```
# DRY approach
def calculate_sum(numbers):
    return sum(numbers)

def calculate_average(numbers):
    if not numbers:
        return 0
    return calculate_sum(numbers) / len(numbers)
```

This code calculates the sum and average of a list of numbers. Instead of duplicating the code to sum and average the list of numbers, the `calculate_sum` function is defined and reused in the `calculate_average` function. This makes the code more concise, easier to maintain and less error-prone.

Example of code that does not follow DRY:

```
# Not DRY approach
def calculate_sum(numbers):
    total = 0
    for num in numbers:
        total += num
    return total

def calculate_average(numbers):
    total = 0
    for num in numbers:
        total += num
    if len(numbers) == 0:
        return 0
    return total / len(numbers)
```

This code also calculates the sum and average of a list of numbers, but it repeats the code to calculate the sum in both functions. This makes the code longer, harder to maintain and more error-prone. If a bug is found in the sum calculation, it would have to be fixed in both functions.

By refactoring the code to follow the DRY principle, we could improve the code quality and avoid duplicating code. This would lead to more maintainable and efficient code in the long run.

References:

- <https://realpython.com/lessons/zen-of-python/>
- <https://medium.com/technology-hits/dry-dont-repeat-yourself>

## KISS - Keep It Simple, Stupid

This principle suggests that we should aim for simplicity in our code and avoid unnecessary complexity.

Code Example:

```
def is_palindrome(word):  
    return word == word[::-1]
```

Above code defines a function that checks whether a given word is a palindrome (i.e. reads the same backward as forward). The function is concise and easy to understand, adhering to the KISS principle.

Example of code that does not follow KISS:

```
def calculate_fibonacci(n):  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return calculate_fibonacci(n-1) + calculate_fibonacci(n-2)
```

This code defines a function that calculates the nth number in the Fibonacci sequence recursively. While this code is functional, it can be hard to read and understand, especially for those who are not familiar with the Fibonacci sequence or recursive functions. This violates the KISS principle by introducing unnecessary complexity.

References:

- <https://code-specialist.com/code-principles/kiss>
- <https://softwareengineering.stackexchange.com/questions/178294/kiss-principle-applied-to-programming-language-design>

# YAGNI - You Aren't Gonna Need It

This principle suggests that we should avoid adding functionality to our code until we actually need it.

Code Example:

```
def multiply_numbers(num1, num2):  
    return num1 * num2
```

This code defines a function that multiplies two numbers together. It does only what is needed for the immediate task and does not add any unnecessary functionality.

Example of code that does not follow YAGNI:

```
class Question:  
    def __init__(self, name: str, options: list, type: TypeEnum):  
        self.name = name  
        self.options = options  
        self.type = type  
  
    def set_options(self, options):  
        self.options = options
```

This code defines a `Question` class with several attributes and methods, including attributes for `options`, and methods for adding options for the `options` attribute. However, it's unclear whether all of these attributes and methods will be used in the immediate project, and whether they will be needed in the future. By Adding unnecessary functionality and complexity to the code violates the YAGNI principle.

References:

- <https://dev.to/richardwynn/yagni-principle-in-100-seconds-1i6j>
- <https://solidstudio.io/blog/deep-dive-into-kiss-and-yagni>

## Clean Code

The idea of clean code is to write code that is easy to read, understand, and maintain. This involves using clear and descriptive variable and function names, following good coding conventions, and breaking code up into small, modular functions. Example:

```
def find_missing_number(numbers):  
    """Find the missing number in a list of consecutive numbers."""
```

```
n = len(numbers)
expected_sum = (n + 1) * (n + 2) // 2
actual_sum = sum(numbers)
return expected_sum - actual_sum
```

This code defines a function that finds the missing number in a list of consecutive numbers. The function is well-organized, has a clear and descriptive function name and documentation, and uses clear and concise variable names. This adheres to the Clean Code principle of writing code that is easy to read, understand, and maintain.

Example of code that does not follow Clean Code:

```
def fmm(nmbrs):
    n = len(nmbrs)
    es = (n + 1) * (n + 2) // 2
    ac = sum(nmbrs)
    return es - ac
```

This code defines the same function as the previous example, but with poorly named variables and an unclear function name. It's harder to understand what the code does and what the variables represent.

References:

- [Clean Code in Python by Mariano Anaya](#)

# Test-Driven Development

Test-Driven Development (TDD) is considered a best practice in software development for several reasons:

1. **Improved Code Quality:** TDD promotes writing high-quality code by focusing on small units of functionality at a time. Developers write tests before writing the code, which helps clarify the expected behavior and ensure that the code meets those requirements. By continually running tests during development, developers can catch bugs early, leading to cleaner and more robust code.
2. **Faster Debugging and Bug Fixing:** With TDD, bugs are often caught early in the development process since tests are executed frequently. When a test fails, it indicates the presence of a bug. Developers can then pinpoint the issue quickly and fix it before moving forward. This iterative approach saves time in the long run by reducing the debugging phase.
3. **Facilitates Refactoring:** Refactoring is the process of improving code without changing its behavior. TDD provides a safety net for refactoring by ensuring that tests act as a

safety harness. Developers can confidently make changes to the code-base, knowing that if they accidentally introduce a bug, the tests will catch it. This ability to refactor code without fear encourages cleaner and more maintainable code-bases.

4. **Regression Detection:** TDD's incremental and iterative nature ensures that changes to the code-base are made in small, manageable steps. After each step, developers run the tests to verify that the existing functionality is still intact. If a test fails, it immediately highlights a regression, indicating that the refactoring has unintentionally broken existing behavior. This immediate feedback helps pinpoint the cause of the regression, allowing developers to quickly identify and rectify the issue.
5. **Documentation and Examples:** Test cases serve as a form of documentation for the code-base. They provide concrete examples of how the code should behave and can act as living documentation for other developers. Newcomers to the code-base can understand the intended functionality by reading the tests, facilitating faster on-boarding and reducing reliance on outdated or missing documentation.
6. **Confidence and Peace of Mind:** TDD gives developers confidence in their code-base. Passing tests indicate that the code behaves as expected, reducing uncertainty and providing peace of mind. Developers can make changes to the code-base with confidence, knowing that if they accidentally break something, the tests will quickly alert them.

While **TDD** has many advantages, it may not be suitable for every situation or team. It requires discipline and initial investment in writing tests, which can slow down the development process in the short term.

## Example of TDD

Let's say we want to create a simple function that adds two numbers together. Using TDD, we would follow these steps:

1. Write a test case that defines the behavior of the function:

```
import unittest

class TestAddition(unittest.TestCase):
    def test_add_numbers(self):
        result = add_numbers(2, 3)
        self.assertEqual(result, 5)
```

2. Run the test case and observe that it fails since the `add_numbers` function does not exist yet.
3. Implement the `add_numbers` function to pass the test:

```
def add_numbers(a, b):
    return a + b
```

4. Run the test case again and verify that it passes.

ensure that your code is testable and you are only writing code that is **necessary** pass the test.

References:

- <https://www.linkedin.com/pulse/what-advantages-test-driven-development-fortegroup>
- <https://www.codica.com/blog/test-driven-development-benefits/>
- <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-test-driven-development-tdd/>

## Documentation

Documentation serves as a valuable resource for current developers, and future maintainers, and maintaining the software. Here are a minimum requirements for Documentation:

- **API Documentation:** This type of documentation focuses on explaining the usage, inputs, outputs, and behavior of application programming interfaces (APIs). It helps developers integrate and interact with the code-base effectively.
- **Installation and Configuration Guides:** These guides explain the installation process, system requirements, and configuration options for deploying and setting up the software environment.
- **Release Notes:** Release notes provide information about new features, bug fixes, known issues, and compatibility changes in each software release. They help users and stakeholders understand the changes and potential impacts of an upgrade.
- **User Manuals:** User manuals or guides are created to assist end-users in understanding how to use the software. They typically provide step-by-step instructions, explanations of features, and troubleshooting tips.

Important points about documentation:

- **Keep it Up to Date:** Regularly review and update documentation to ensure its accuracy and relevance. Outdated or incorrect documentation can be misleading and counterproductive.
- **Balance Detail and Conciseness:** Document important details without overwhelming the reader. Use clear and concise language, provide examples, and consider different audiences' needs.
- **Use Consistent Formatting:** Establish a consistent formatting style throughout the documentation to enhance readability. Use headings, bullet points, and formatting conventions to structure the content effectively.
- **Include Examples and Visuals:** Examples, diagrams, and screenshots can significantly enhance understanding, especially for complex concepts or workflows. Visual aids help illustrate relationships, dependencies, and system flows.



- **Make it Searchable:** Use proper organization and indexing techniques to make documentation easily searchable. This enables users to quickly find the information they need.
- **Collect Feedback:** Encourage users and developers to provide feedback on the documentation. Feedback can help identify areas for improvement, clarify ambiguities, and address common pain points.

References:

- <https://blog.jetbrains.com/writerside/2022/01/the-holy-grail-of-always-up-to-date-documentation/>
  - <https://medium.com/@lanceharvieruntime/code-documentation-waste-of-time-or-vital-for-success-in-c-and-c-development-c1618c0c2f7a>
- 

# Tests

It's essential to include testing in your development process to catch bugs early and ensure that your application meets the required quality standards. There are several types of tests that you should consider writing to ensure the quality and reliability of your code.

## Unit tests

These tests are written to test the smallest units of code in isolation, such as individual functions or methods. They are usually automated and should be written for every new piece of code you add to your application.

Example: [test\\_category\\_data\\_frame.py](#)

## Integration tests

These tests check how different parts of the application work together. They test how components interact and ensure that the application functions as expected when all the pieces are put together.

Example: [tests\\_form\\_approval.py](#)

## Functional tests

These tests focus on testing the application's features and functionalities. They ensure that the user's requirements are met and that the application works as intended.

Example: [geo.test.js](#)

## Performance tests

These tests are designed to check the application's ability to handle a large number of users and data without slowing down or crashing.

Example: [test\\_10\\_stress\\_and\\_timeout.py](#)

## Security tests

These tests ensure that the application is secure and that sensitive data is protected from unauthorized access.

Example: [test\\_01\\_auth.py](#)

## End-to-end tests

In an end-to-end test, the system is tested as a whole, without any isolation of its components. This means that the tests encompass the entire stack of the software system, including the user interface, application logic, and database. The objective of these tests is to verify that the software system is functioning as expected and that the user's needs are met.

Example: [00\\_login\\_with\\_verified\\_super\\_admin.side](#)

---

# Code Coverage

Code coverage is a measure of how much of a software application's source code is executed during the testing process. It is used to determine the effectiveness of software testing and the quality of the testing suite. Code coverage is expressed as a percentage, representing the percentage of code that was executed during testing. For example, if a test suite runs through 80% of the lines of code in a software application, then the code coverage is said to be 80%.

## Coveralls

Tech consultancy team uses [Coveralls](#) because it provides a simple and efficient way to measure code coverage in our development process. Coveralls can be integrated with GitHub repositories to provide code coverage reports for each pull request, allowing developers to quickly and easily identify code changes that have impacted test coverage. With this integration, we can ensure that all changes to the code-base are properly tested, and we can catch any regressions before they make it into the main code-base.

Example code coverage summary by Coveralls:

SOURCE FILES ON MAIN

TREE

LIST 57

CHANGED 6

SOURCE CHANGED 0

COVERAGE CHANGED 6

SEARCH:

COVERAGE	FILE	LINES	RELEVANT	COVERED	MISSED	HITS/LINE	BRANCH HITS	BRANCH MISSES
<div>↓ 76.67</div>	api/v1/v1_forms/views.py	204	102	84 -6	18 +6	1.0	8 -2	10 +2
<div>↑ 81.97</div>	api/v1/v1_data/views.py	1778	638	554 +10	84 -10	1.0	169 +6	75 -6
<div>↑ 83.08</div>	utils/custom_permissions.py	52	35	29 +1	6 -1	1.0	25 +1	5 -1
<div>↓ 87.56</div>	api/v1/v1_users/views.py	685	320	296 -1	24 +1	1.0	70 -1	28 +1
<div>↑ 87.69</div>	api/v1/v1_data/serializers.py	1244	596	532 +61	64 -61	1.0	202 +19	39 -18
<div>↑ 100.0</div>	...1/v1_users/management/commands/fake_user_seeder.py	57	32	32 +1	0 -1	1.0	8 +1	0 -1

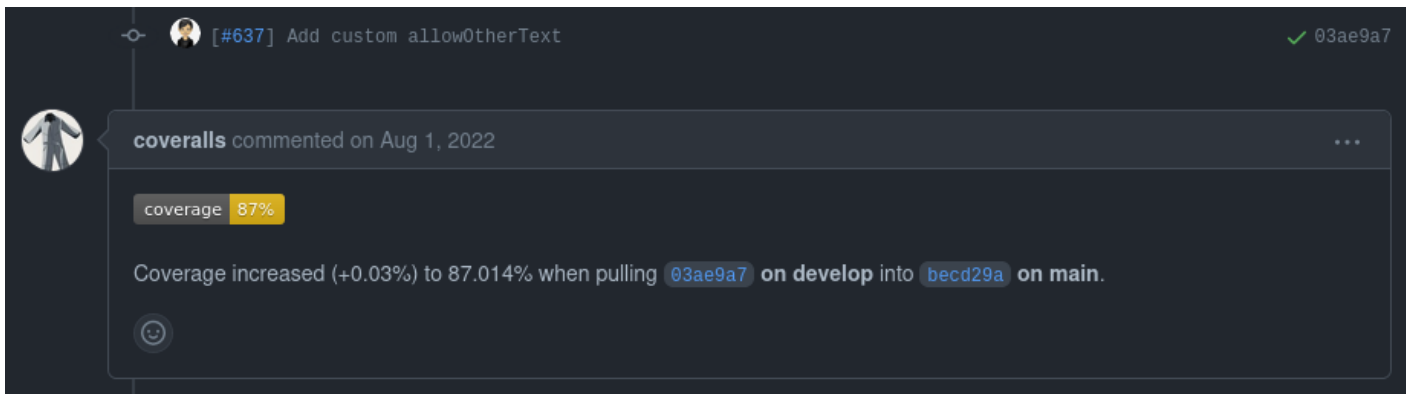
SHOW 10

ENTRIES

Showing 1 to 6 of 6 entries

< PREVIOUS 1 NEXT >

Example coveralls notifications on Pull Requests:



## Minimum Coverage Requirements

Based on the Tech Consultancy KPI, we must established a minimum code coverage requirement of **80%**

This means that at least 80% of the source code in our projects must be executed during testing. We have set this requirement to ensure that our software is thoroughly tested and that we are delivering high-quality products to our customers. By striving for a minimum code coverage of 80%, we can identify areas of the code that are not being adequately tested, and we can ensure that we are catching potential issues before they make it into production. We encourage the TC team to aim for a code coverage percentage higher than the minimum requirement, as this will help us to build more reliable and robust software.

## Continuous Integration / Continuous Delivery

By implementing Continuous Integration/Continuous Delivery (CI/CD) with **Semaphore** or **GitHub Workflows**, we continuously build, test, and deploy code changes in a repeatable, automated way. This means that our development and operations teams can collaborate more effectively and identify issues early on in the development process, leading to faster resolution and a more stable application.

## Semaphore CI

Semaphore (Semaphore CI) is the most frequently used for Tech Consultancy Projects. Particularly dealing with Kubernetes deployment, Semaphore provides more customization options for workflow execution environments and integrates with Kubernetes for seamless deployment.

Semaphore also provides more advanced customization options for workflow execution environments, such as custom Docker images and caching, which can help speed up your builds and deployments.

Example config:

```
---
version: v1.0
name: RTMIS
agent:
  machine:
    type: e1-standard-2
    os_image: ubuntu1804
global_job_config:
  secrets:
    - name: GCP
    - name: docker-hub-credentials
    - name: coveralls
    - name: rtmis
  prologue:
    commands:
      - echo "${DOCKER_PASSWORD}" | docker login --username
        "${DOCKER_USERNAME}" --password-stdin
      - export CI_COMMIT="${SEMAPHORE_GIT_SHA:0:7}"
      - export CI_BRANCH="${SEMAPHORE_GIT_BRANCH}"
      - export CI_TAG="${SEMAPHORE_GIT_TAG_NAME}"
      - export CI_PULL_REQUEST="${SEMAPHORE_GIT_REF_TYPE/pull-request/true}"
      - export CI_COMMIT_RANGE="${SEMAPHORE_GIT_COMMIT_RANGE}"
      - export CLOUDSDK_CORE_DISABLE_PROMPTS=1
      - export COMPOSE_INTERACTIVE_NO_CLI=1
```

- export COVERALLS\_REPO\_TOKEN="\${COVERALLS\_RTMS\_TOKEN}"
- export SERVICE\_ACCOUNT=/home/semaphore/credentials

blocks:

- name: 'Build, Test & Push'

skip:

when: "tag =~ '.\*'"

task:

prologue:

commands:

- checkout
- cache restore "npm-\$(checksum frontend/package.json)"
- cache restore "node-modules-\$(checksum frontend/package.json)"
- cache restore "pip-\$(checksum backend/requirements.txt)"
- cache restore "images-\${SEMAPHORE\_PROJECT\_ID}"
- cache restore "\${SEMAPHORE\_PROJECT\_ID}-\${SEMAPHORE\_GIT\_BRANCH}"

epilogue:

commands:

- cache store "npm-\$(checksum frontend/package.json)" "\$HOME/.npm"
- cache store "node-modules-\$(checksum frontend/package.json)"  
"frontend/node\_modules"
- cache store "pip-\$(checksum backend/requirements.txt)"  
"backend/.pip"
- cache store "images-\${SEMAPHORE\_PROJECT\_ID}" "ci/images"
- cache store "\${SEMAPHORE\_PROJECT\_ID}-\${SEMAPHORE\_GIT\_BRANCH}"  
"\$HOME/.cache"

jobs:

- name: Build & Test

commands:

- ./ci/build.sh
- ./ci/deploy.sh

- name: "Promote to production"

run:

when: "tag =~ '.\*'"

task:

jobs:

- name: Promote to production

commands:

- checkout
- ./ci/deploy.sh

Source: <https://github.com/akvo/rtmis>

## GitHub Workflows

In some other cases, when we only need to run tests or code quality checks (e.g. develop a package library), Semaphore may not be needed. That's why we take advantage of the GitHub Workflow, because it's simple, integrated, and easy to use.

Example config:

```
name: Build & Test

on:
  push:
    branches:
      - main
      - feature*
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python: [3.8, 3.9]
    steps:
      - uses: actions/checkout@v2
      - name: Setup Python
        uses: actions/setup-python@v2
        with:
          python-version: ${ matrix.python }
      - name: Install Tox and any other packages
        run: pip install tox
      - name: Run Tox
        # Run tox using the version of Python in `PATH`
        run: tox -e py

  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
```

```
- name: Run Container
  env:
    COVERALLS_REPO_TOKEN: ${ secrets.COVERALLS_REPO_TOKEN }
  run: docker compose -f "docker-compose.ci.yml" up --exit-code-from backend
```


Source: <https://github.com/akvo/Akvo-ResponseGrouper>

## Slack Notifications

When a build fails in your CI/CD pipeline, it's important for developers to take prompt action to resolve the issue. Here are some steps that developers can take when a build fails:

1. Review the build logs: The first step is to review the build logs to identify the cause of the failure. The logs should provide detailed information about what went wrong, such as error messages or stack traces.
2. Reproduce the issue: Once the cause of the failure has been identified, the next step is to reproduce the issue locally.
3. Fix the issue: Once the issue has been reproduced, developers should work to fix the problem.
4. Test the fix: After making the necessary changes, developers should test the fix locally to ensure that it resolves the issue.
5. Communicate with the team: Finally, it's important for developers to communicate with their team about the issue and the steps taken to resolve it.


Example GitHub Workflows Notification on Slack:

 **GitHub** APP 9 minutes ago  
Workflow was triggered via push by [dedenbangkit](#)

**Build & Test #215**  
**Status**  
❌ Failure  
**Duration**  
1m 31s  
akvo/Akvo-ResponseGrouper | Today at 06:17  
[Re-run all jobs](#) [Re-run failed jobs](#)


**Commit**  
[fb64179 \(feature/30-code-quality-check\)](#)

2 replies


 **GitHub** APP 9 minutes ago  
**Jobs(3)**

- ✅ test  
Succeeded after 1m 22s
- ❌ build (3.8)  
Failed after 45s
- ❌ build (3.9)  
Failed after 47s


#\* Also sent to the channel

 **GitHub** APP 8 minutes ago  
❌ Workflow [Build & Test #215](#) failed


Example Semaphore Notification on Slack:


**# proj-wcaro-mis-dev-notifications**  5

+ Add a bookmark

 **semaphore** APP 14:52  
**national-wash-mis**  
dedenbangkit's [National WASH MIS](#) passed — [e896a76a0](#) Merge pull request #52 from akvo/feature/50-missing on develop

**Yesterday** ▾

 **GitHub** APP 15:50  
replied to [a thread](#)  
Pull request merged by [dedenbangkit](#)

**#51 Develop**  
 [akvo/national-wash-mis](#) Apr 28th

Typically you will get notification of the CI status in different channels on Slack (e.g [#proj-wcaro-mis-dev-notifications](#))

1.



# Release / Deployment

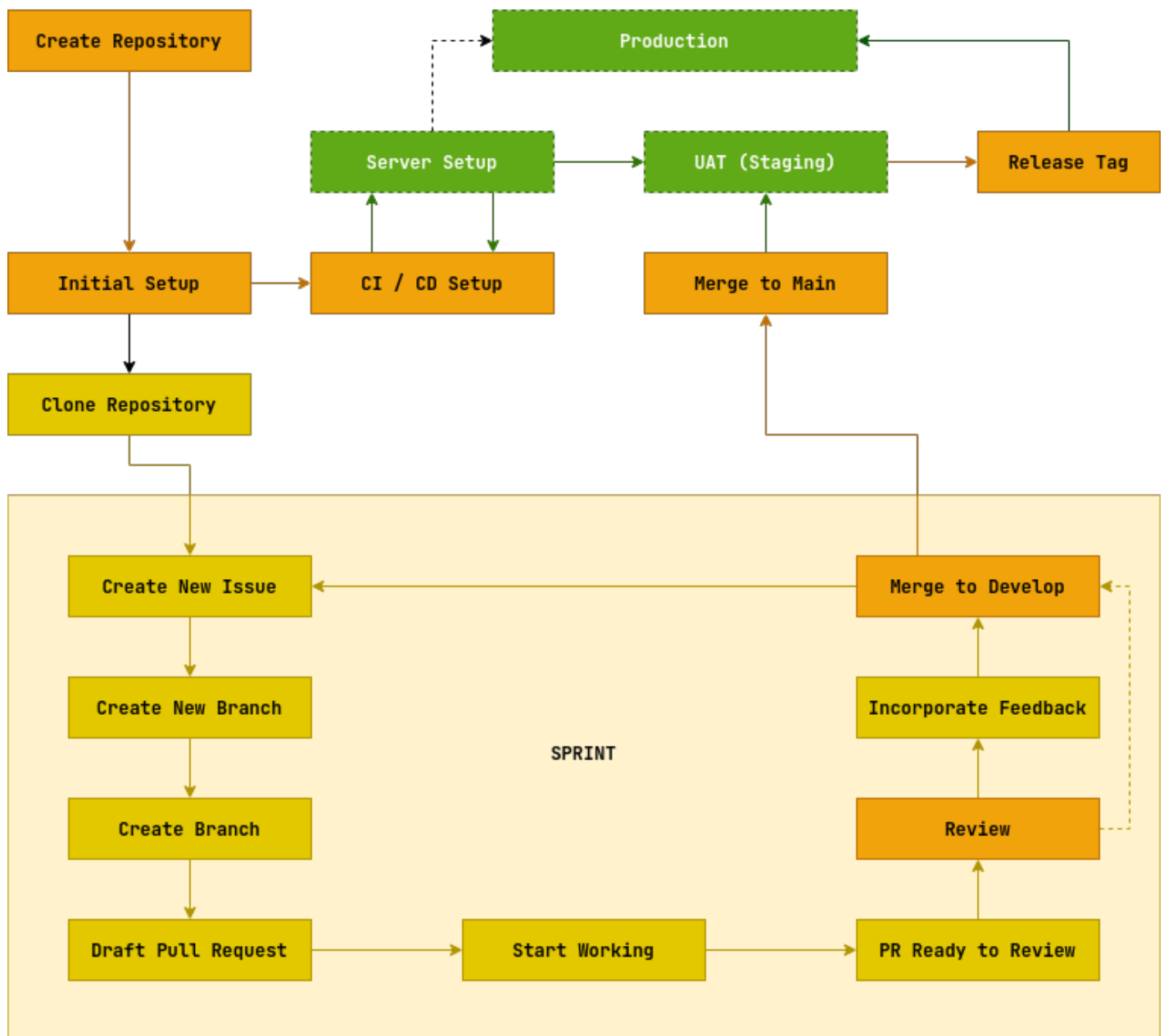
Draft for @Anjar

---

## Summary

Developer Sprint should only started after we have a design document that outlines the system architecture, data models, algorithms, and other technical details. During the developer sprint, the development team works on developing new features or fixing bugs that have been identified. Sprints are typically short, time-boxed periods of development that focus on a specific set of tasks that already estimated in Asana tasks.








■ Project Maintainer ■ Developer ■ DevOps



At the end of each sprint, the team should mark the **Asana** tasks as completed with it's **Actual time**.



Example of a competed task in Asana:

✓ Completed





## JMP Category JSON




Assignee

 Iwan Firmawan 

Due date

 Apr 13 – 14 

Projects

 WCARO A1 Dev tasks Ready for Testing  


Add to projects

Dependencies


Add dependencies

Show hidden dependencies



Estimated time


9h 00m 




Actual time


8h 37m 

GitHub

 #32 Feature/31 jmp category json 

 Pull request in akvo/national-wash-mis · View in GitHub

Review status	Build	PR status	Line changes
 Changes approv...	 Success...	Merged	 +13445 -198...

Created in GitHub Apr 12 at 5:08pm 

## FAQ

I did git commit and found that my local branch is not updated, how to justify it?

If you merge your local branch into the origin branch, there will be only a single commit reported on the origin branch you've merged even if your local branch has multiple commits, regardless of whether you use a plain `git pull` or with `--squash merge`. You should always do `git pull --rebase` instead of `git merge`.

I've run the test locally and everything is fine, but why did the build fails?

Become friends with your broken builds. So, your build fails... what exactly does that mean? You can always check the pipeline build from

**[https://akvo.semaphoreci.com/projects/<repo\\_name>](https://akvo.semaphoreci.com/projects/<repo_name>)**. Checking the job output should point you to a failing test. There are several reasons:

- Build or Deploy code has an error, DevOps to blame.
- Network test (eg. [basic.sh](#)) is error. Meaning that there are missing URLs or the network/proxy is misconfigured.
- Code Quality (Flake, ES-Lint, or Prettier) warning