TEAMAGOCHI - てぃーまごっち

VIRTUAL RIOT PET BY THE RIOT PROJECT SOSE24 TEAM

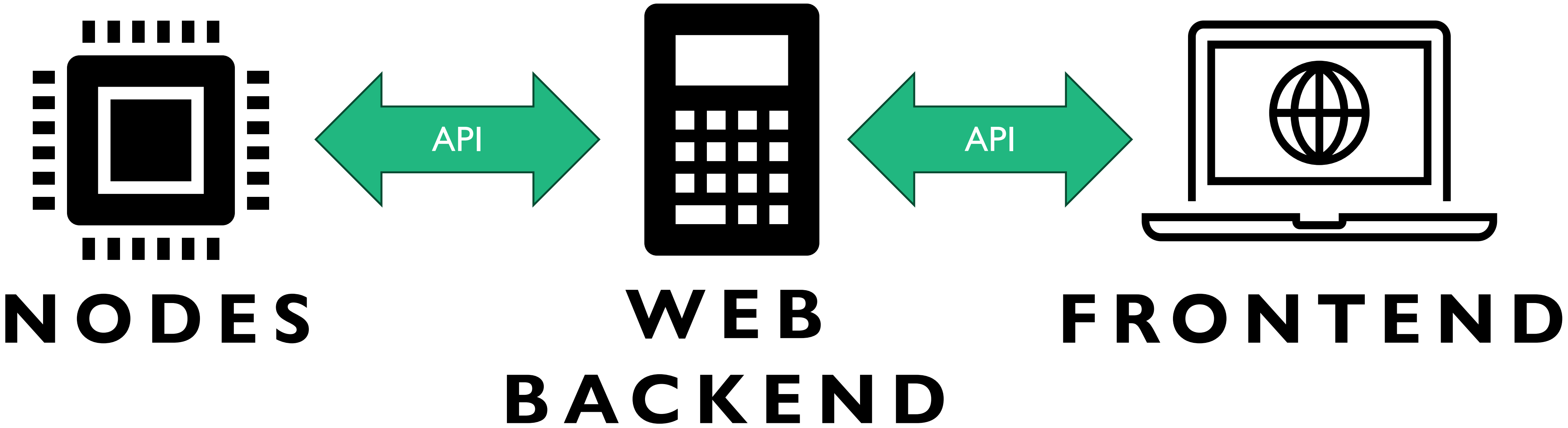
- Tom Hert

OVERVIEW (CONCEPT & ORGANIZATION)

THE GENERAL CONCEPT

A synchronized always-online pet simulator with multiplayer functionality

STRUCTURE

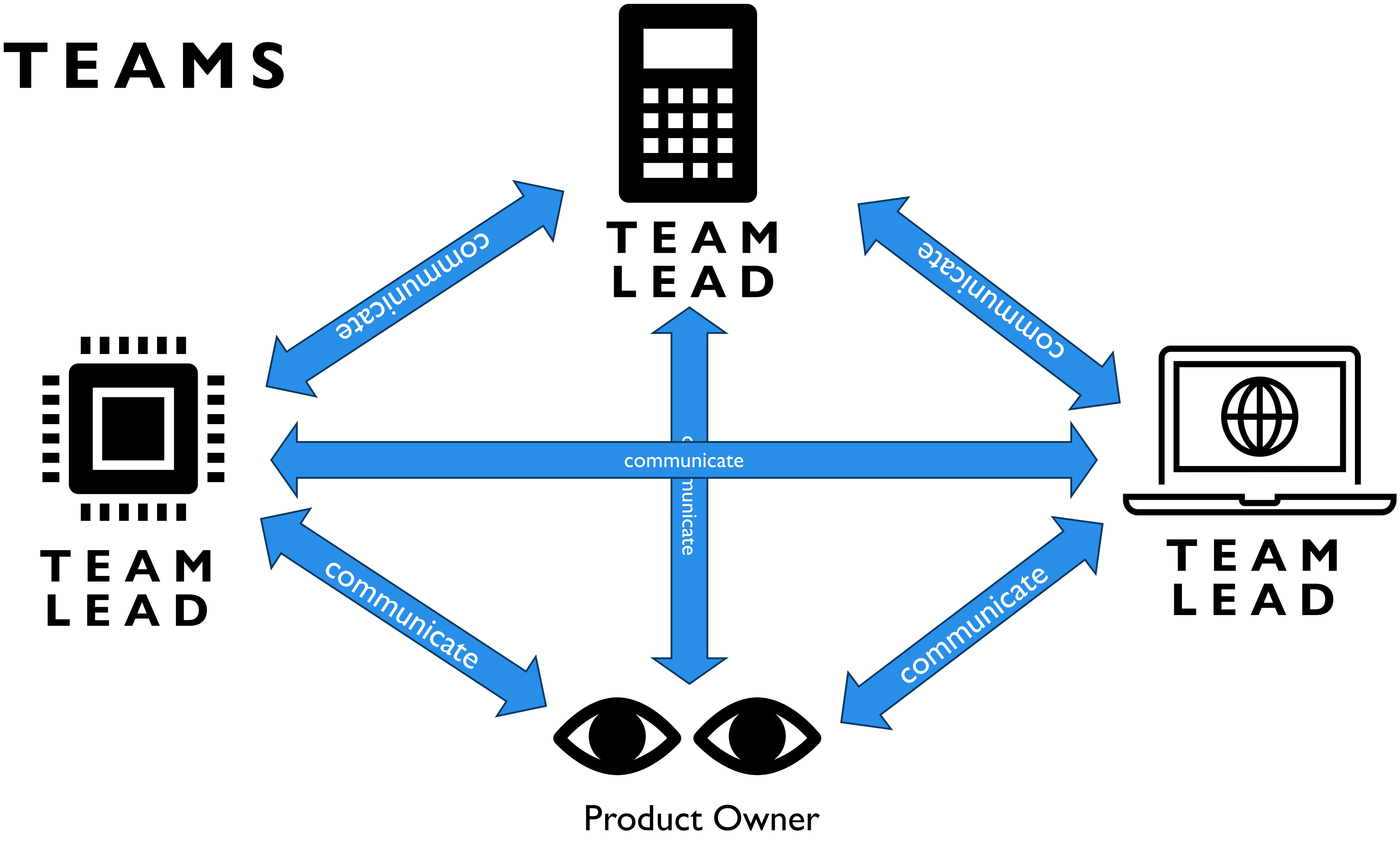


NODES

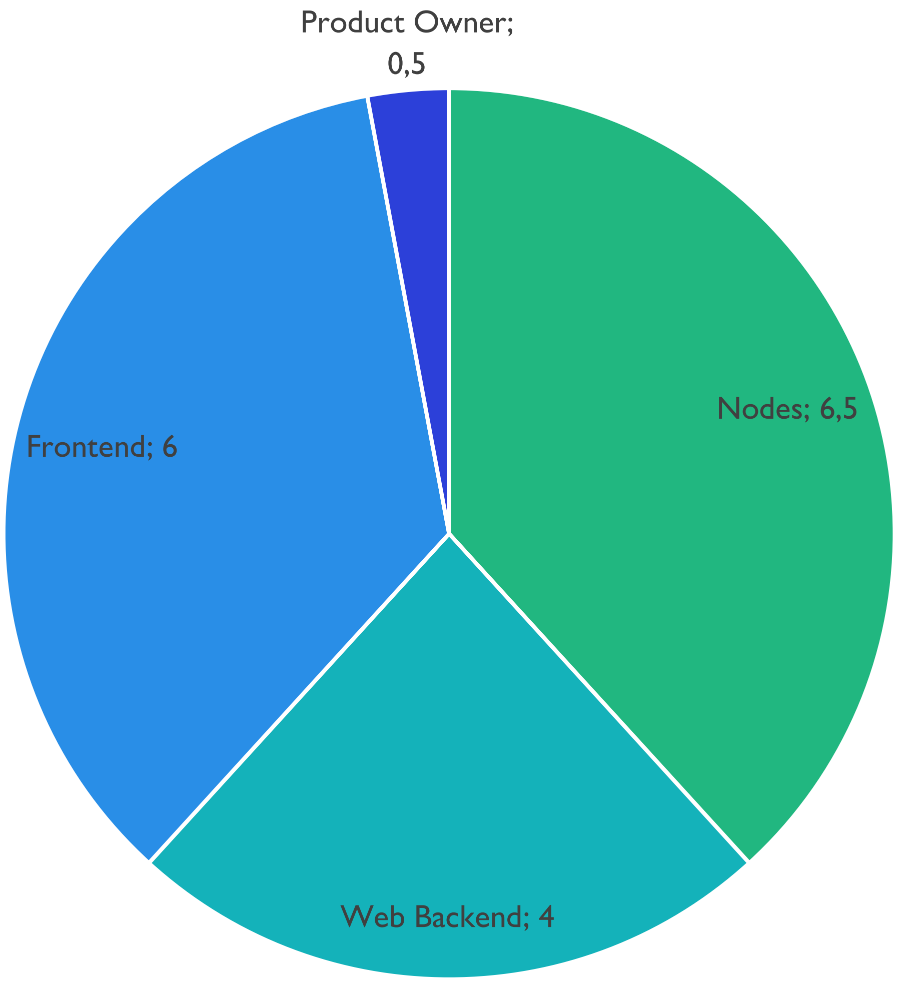
WEB BACKEND

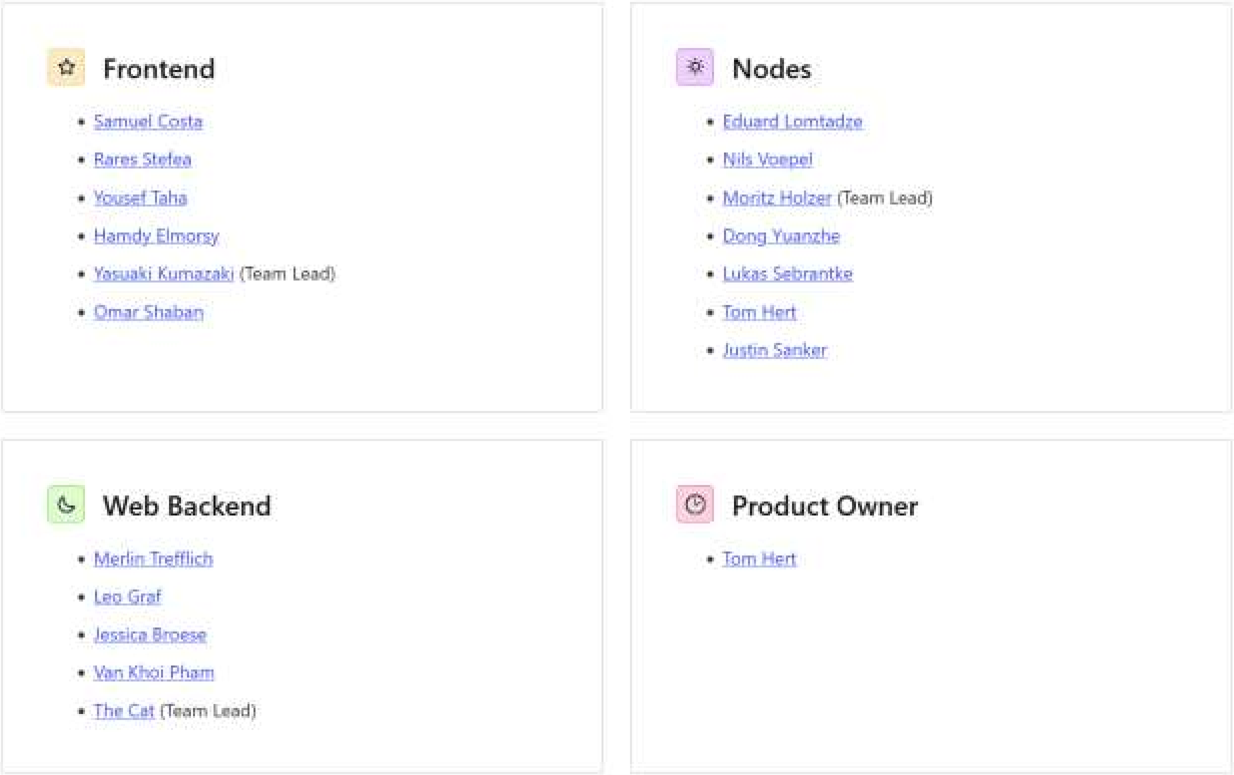
FRONTEND

TEAMS


TEAM LEAD
TEAM LEAD
TEAM LEAD
communicate
communicate
communicate
communicate
communicate
communicate
Product Owner

Ressource Allocation (Students)





Frontend

- Samuel Costa
- Rares Stefea
- Yousef Taha
- Hamdy Elmorsy
- Yasuaki Kumazaki (Team Lead)
- Omar Shaban

Nodes

- Eduard Lomtadze
- Nils Voepel
- Moritz Holzer (Team Lead)
- Dong Yuanzhe
- Lukas Sebrantke
- Tom Hert
- Justin Sanker

Web Backend

- Merlin Trefflich
- Leo Graf
- Jessica Broese
- Van Khoi Pham
- The Cat (Team Lead)

Product Owner

- Tom Hert

MONOREPO & CONTRIBUTING

- All contributions to the project are made to a singular Monorepo on GitHub: https://github.com/smartuni/teamagochi
- Documentation is shared on: https://smartuni.github.io/teamagochi/
- Issues are created on the project board: https://github.com/orgs/smartuni/projects/2
- Pull Requests must be approved before merging

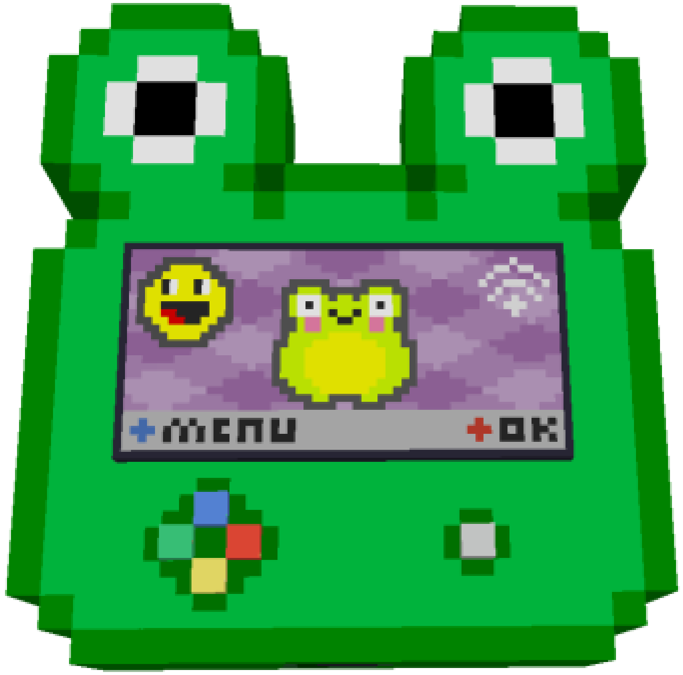
CONSTRAINTS

High-Level Requirements	• Two devices are synchronized by the means of a digital twin. • Backend application and node communicates with via standard IoT application protocols. • The devices should have at least one actuator and one sensor, be battery powered, and use wireless communication.
Technical	• Device is based on Adafruit Feather nRF52840 Sense. • Device software is built upon the RIOT operating system. • Devices communicate with a border router over IEEE 802.15.4. • Devices communicate with a LwM2M Server to report their sensor and actuator values.
Organizational	• The project is organized in three teams: Nodes, Frontend and Web-Backend. • The project is divided into four milestones, the last of which is the presentation of the results (8.7.24). • Git is used for version control and all code is published to Github.

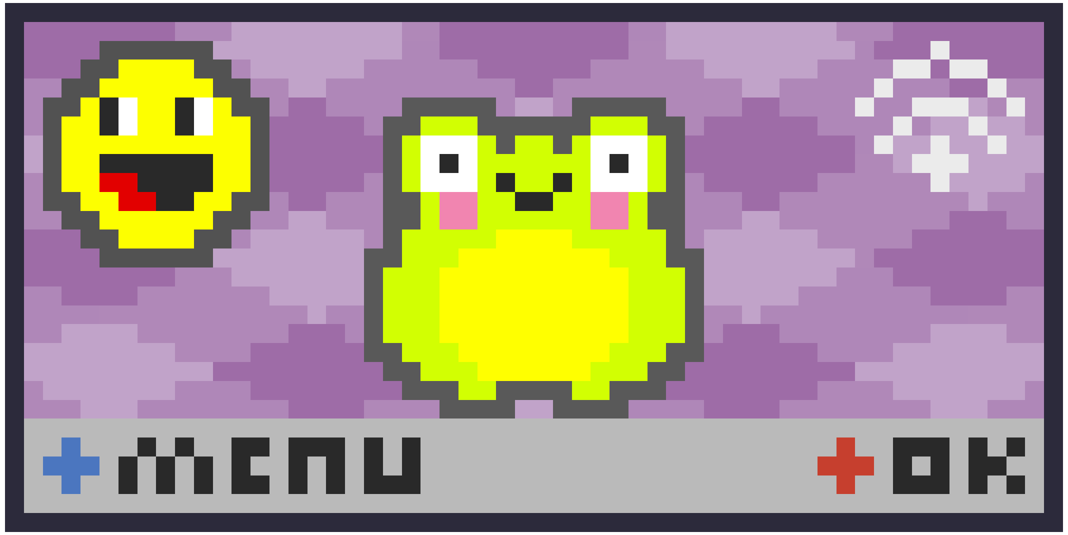
NODES TEAM

- Eduard Lomtadze
- Nils Voepel
- Moritz Holzer
- Dong Yuanzhe
- Lukas Sebrantke
- Tom Hert
- Justin Sanker

MOCKUP OF THE CASE



MOCKUP OF MENU



SENSORS

Temperature

Light

Gyro / Accelerometer

Buttons

OUTPUT

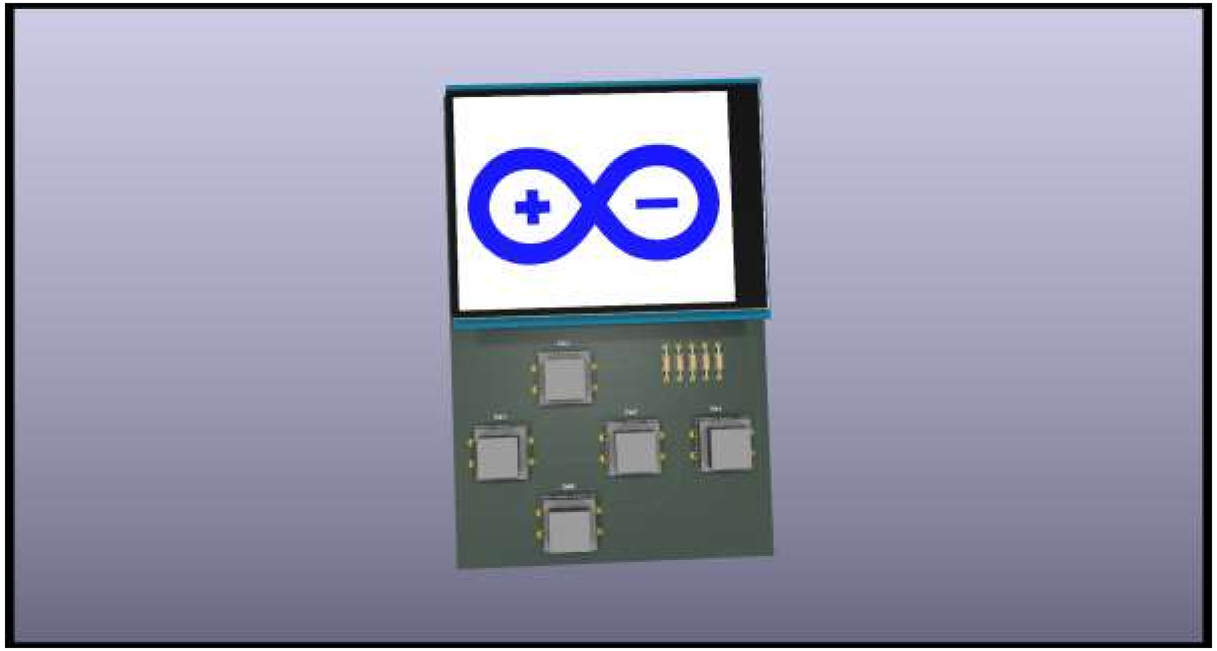
Display

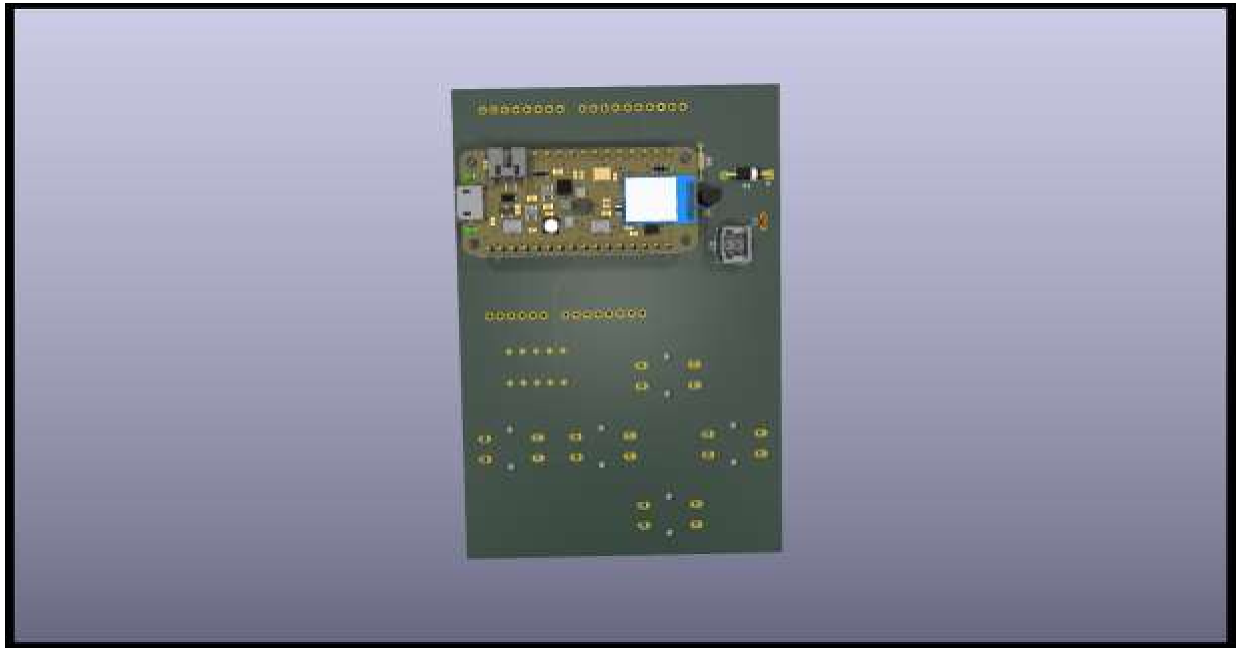
LEDs

Vibration

Beeper

PCB DESIGN

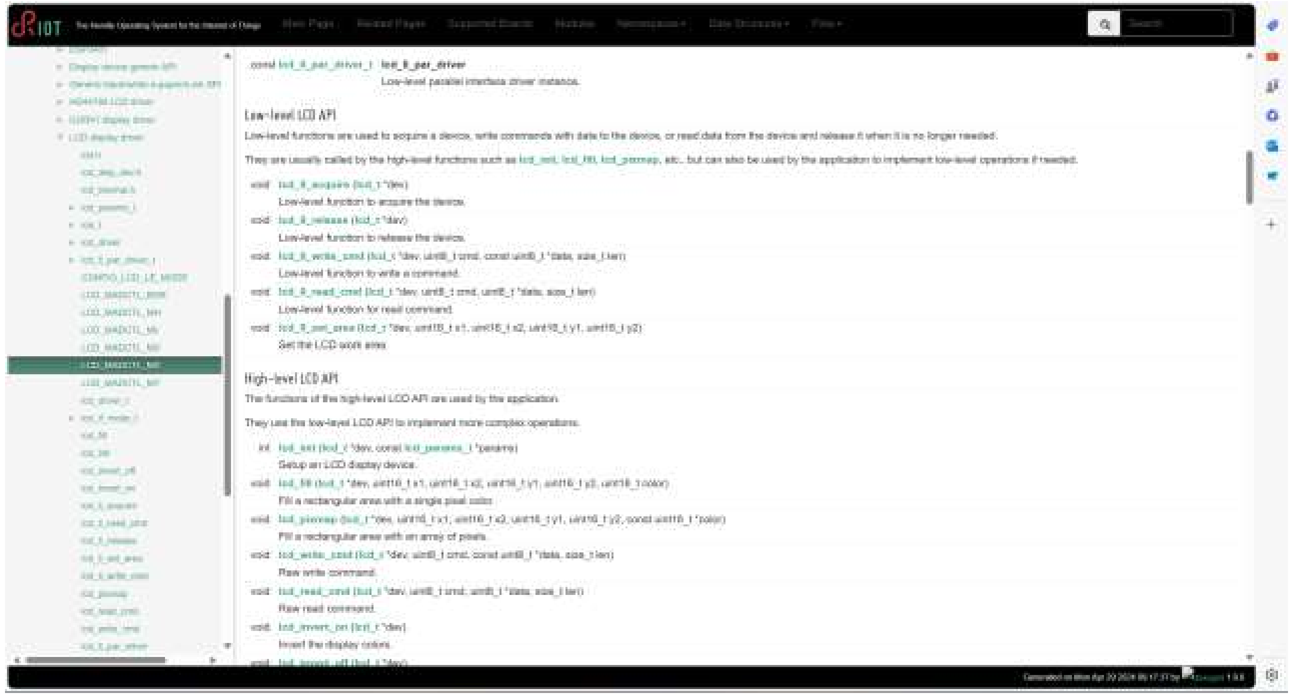




DISPLAY DESIGN

LVGL

ILI9324



const lcd_ll_par_driver_t lcd_ll_par_driver
Low-level parallel interface driver instance.

Low-level LCD API

Low-level functions are used to acquire a device, write commands with data to the device, or read data from the device and release it when it is no longer needed.

They are usually called by the high-level functions such as lcd_init, lcd_fill, lcd_pixmap, etc., but can also be used by the application to implement low-level operations if needed.

```
void lcd_ll_acquire (lcd_t *dev)
     Low-level function to acquire the device.
void lcd_ll_release (lcd_t *dev)
     Low-level function to release the device.
void lcd_ll_write_cmd (lcd_t *dev, uint8_t cmd, const uint8_t *data, size_t len)
     Low-level function to write a command.
void lcd_ll_read_cmd (lcd_t *dev, uint8_t cmd, uint8_t *data, size_t len)
     Low-level function for read command.
void lcd_ll_set_area (lcd_t *dev, uint16_t x1, uint16_t x2, uint16_t y1, uint16_t y2)
     Set the LCD work area.
```

High-level LCD API

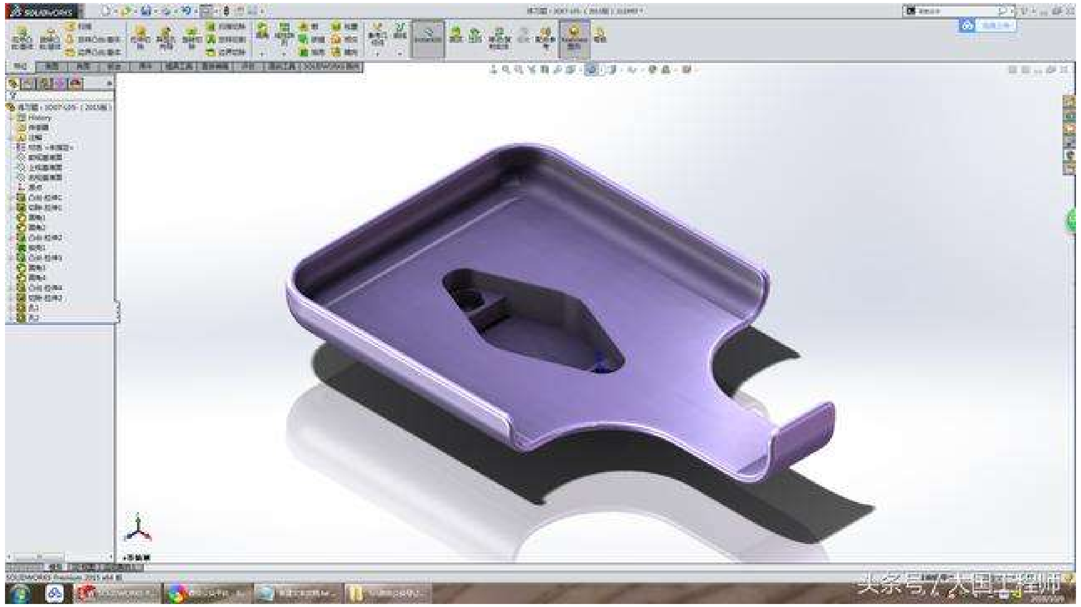
The functions of the high-level LCD API are used by the application.

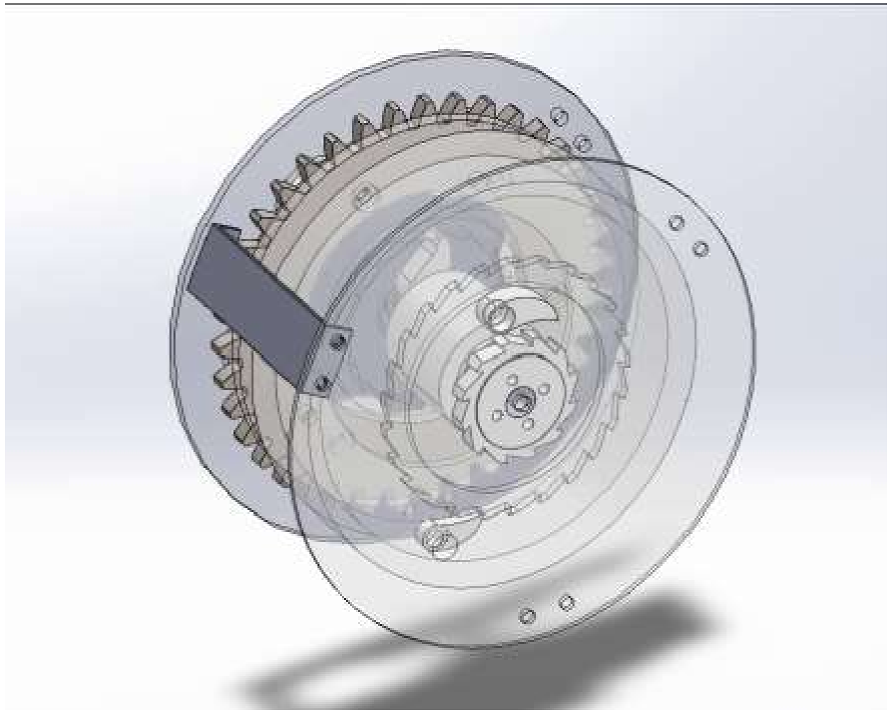
They use the low-level LCD API to implement more complex operations.

```
 int lcd_init (lcd_t *dev, const lcd_params_t *params)
     Setup an LCD display device.
void lcd_fill (lcd_t *dev, uint16_t x1, uint16_t x2, uint16_t y1, uint16_t y2, uint16_t color)
     Fill a rectangular area with a single pixel color.
void lcd_pixmap (lcd_t *dev, uint16_t x1, uint16_t x2, uint16_t y1, uint16_t y2, const uint16_t *color)
     Fill a rectangular area with an array of pixels.
void lcd_write_cmd (lcd_t *dev, uint8_t cmd, const uint8_t *data, size_t len)
     Raw write command.
void lcd_read_cmd (lcd_t *dev, uint8_t cmd, uint8_t *data, size_t len)
     Raw read command.
void lcd_invert_on (lcd_t *dev)
     Invert the display colors.
void lcd_invert_off (lcd_t *dev)
```

RIOT driver with LCD API

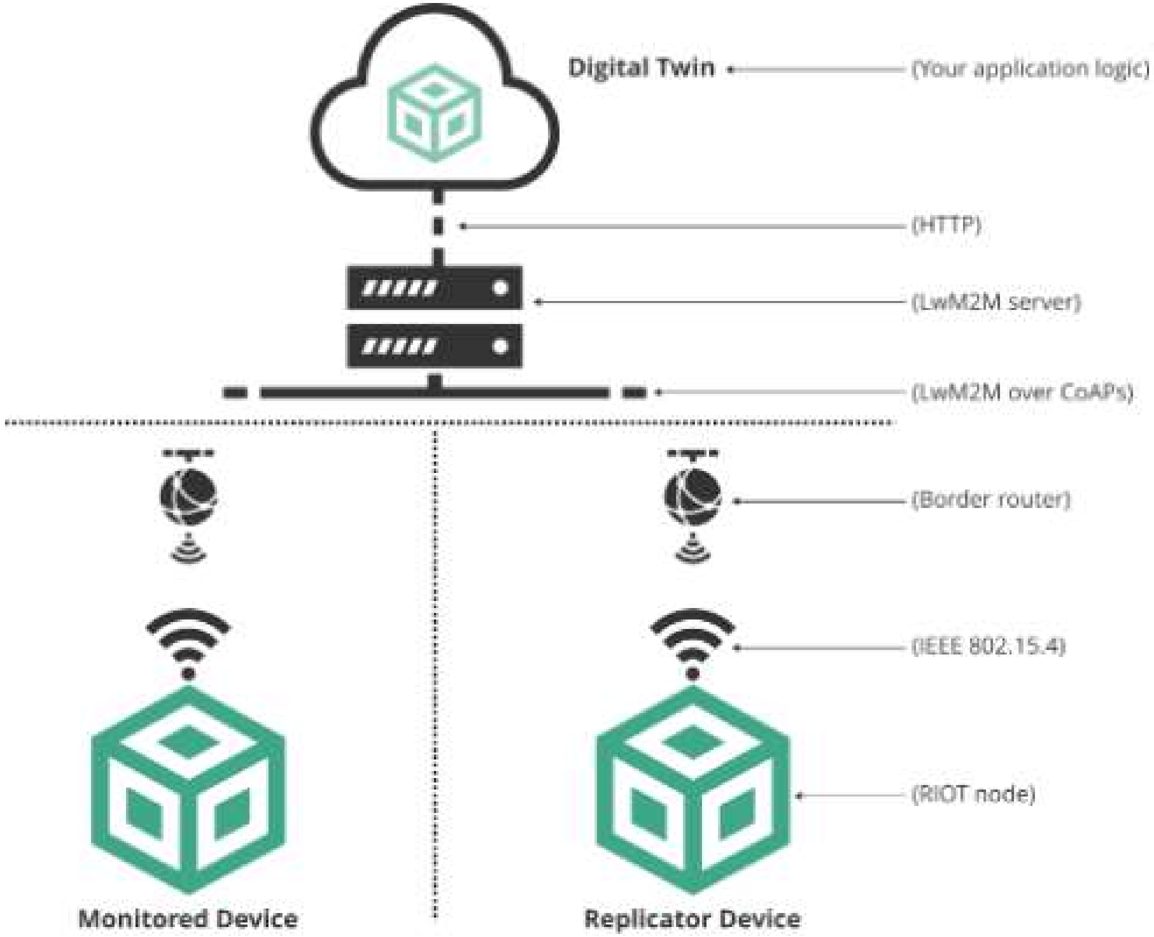
3D PRINT





CONNECTION

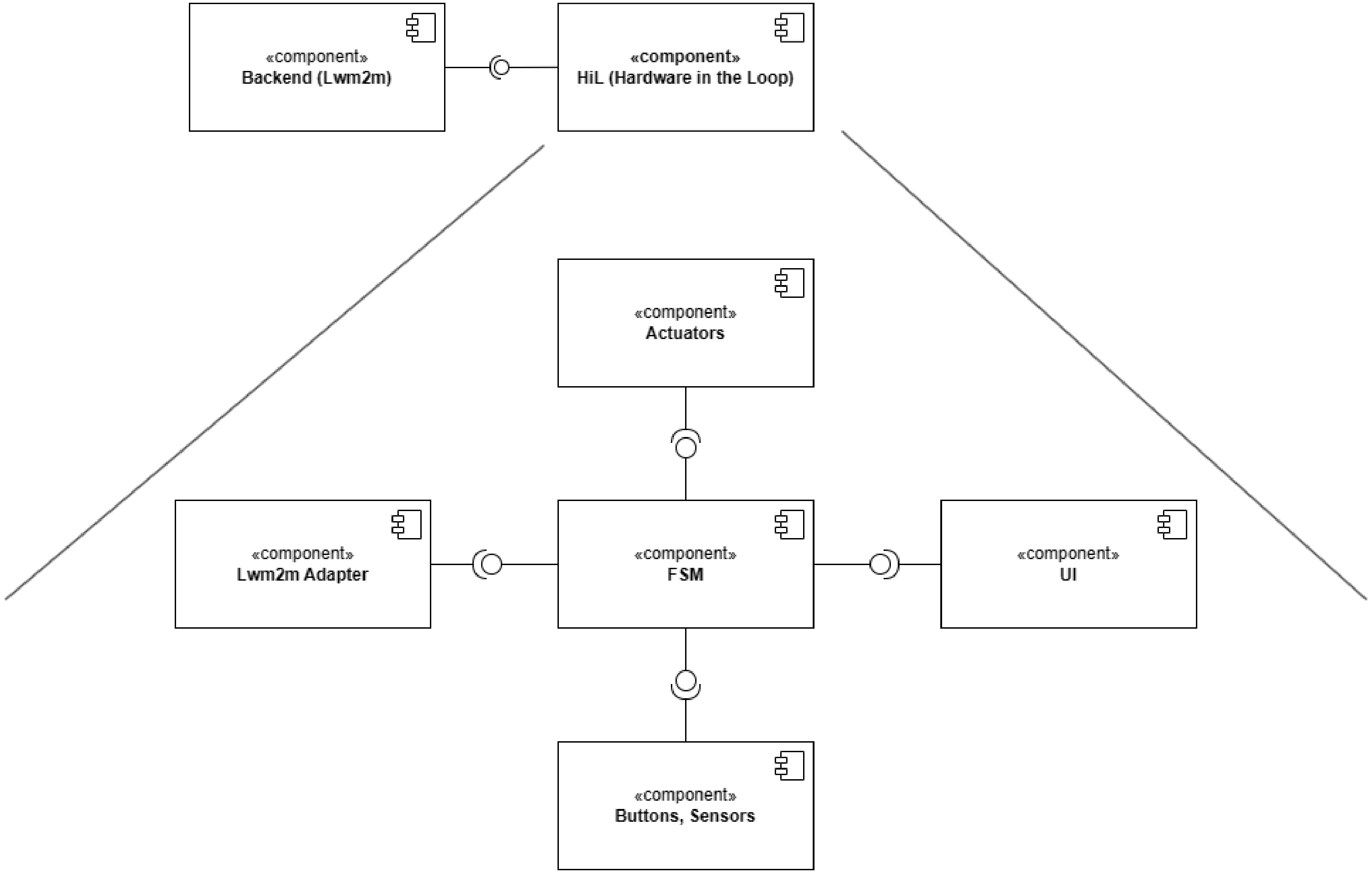
- Onboard WiFi module is used
- 6LoWPAN (Low power personal wireless personal area network)



THE PET & ITS NEEDS

- three different pets
- Different Needs are send to the Node and input for the FSM
- Some first Ideas are:
    - ➢ Feed (Hunger)
    - ➢ Clean (?)
    - ➢ Play (Happiness)
    - ➢ Pet (Happiness)
    - ➢ Medicine (Health)

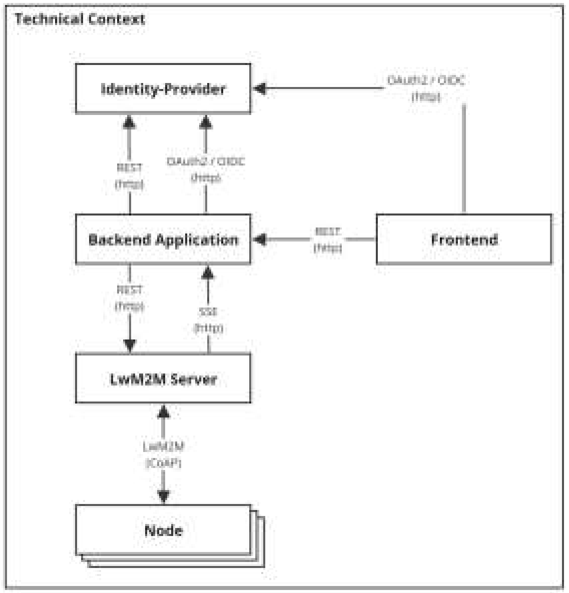
FIRST COMPONENT DIAGRAM

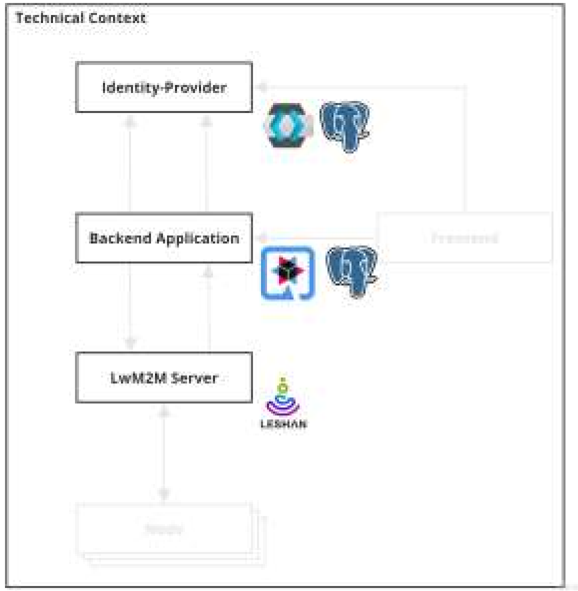


WEB BACKEND TEAM

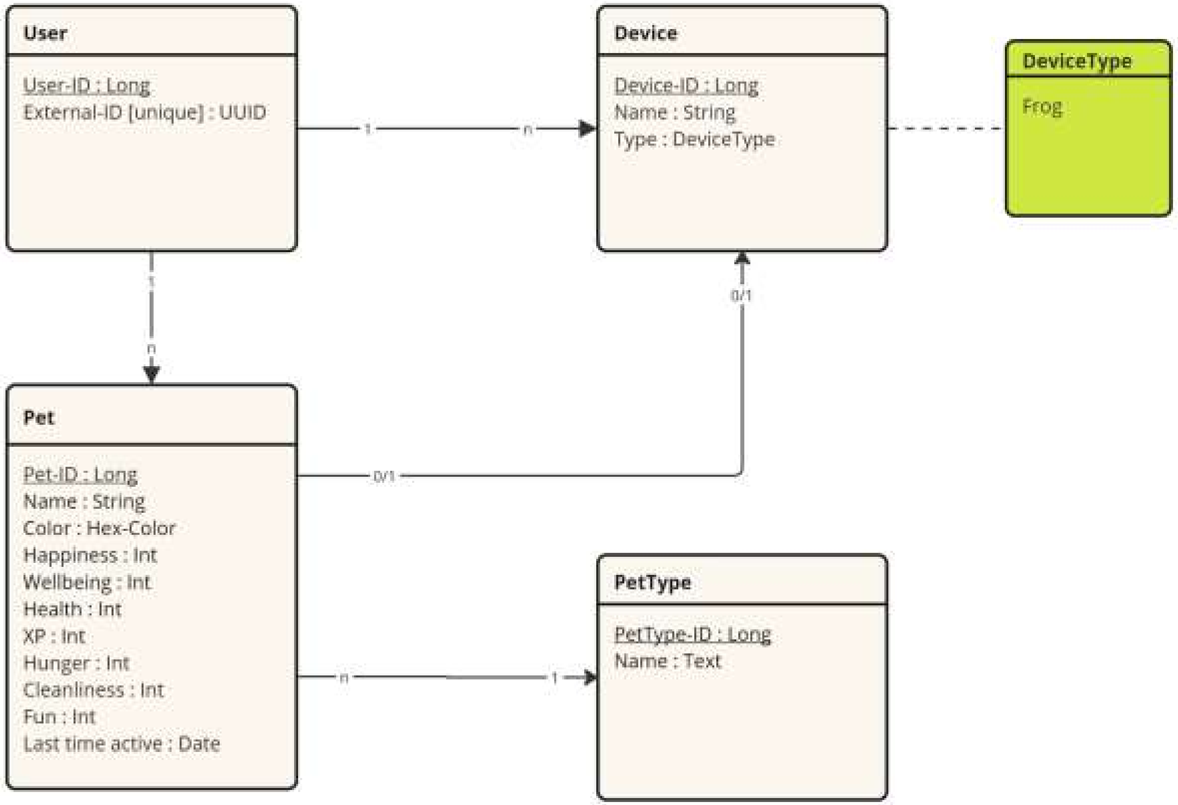
- Merlin Trefflich
- Leo Graf
- Jessica Broese
- Van Khoi Pham

TECHNICAL CONTEXT





DATAMODEL



User

User-ID : Long
External-ID [unique] : UUID

Device

Device-ID : Long
Name : String
Type : DeviceType

DeviceType

Frog

Pet

Pet-ID : Long
Name : String
Color : Hex-Color
Happiness : Int
Wellbeing : Int
Health : Int
XP : Int
Hunger : Int
Cleanliness : Int
Fun : Int
Last time active : Date

PetType

PetType-ID : Long
Name : Text

QUALITY GOALS

Functional Suitability

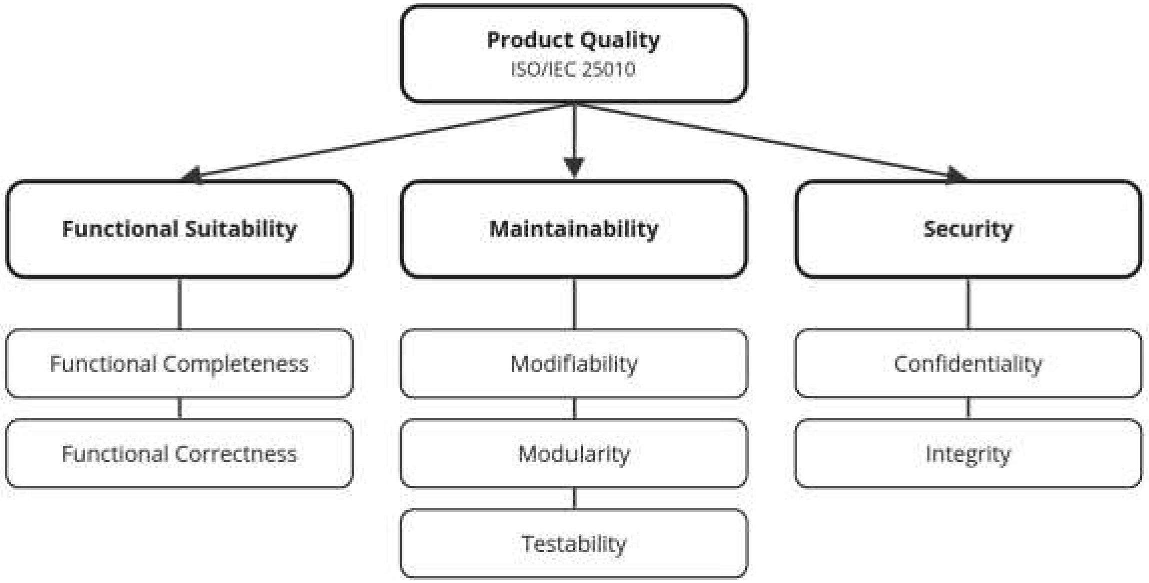
We implement only features which are necessary for the project, and we implement them correctly.

Maintainability

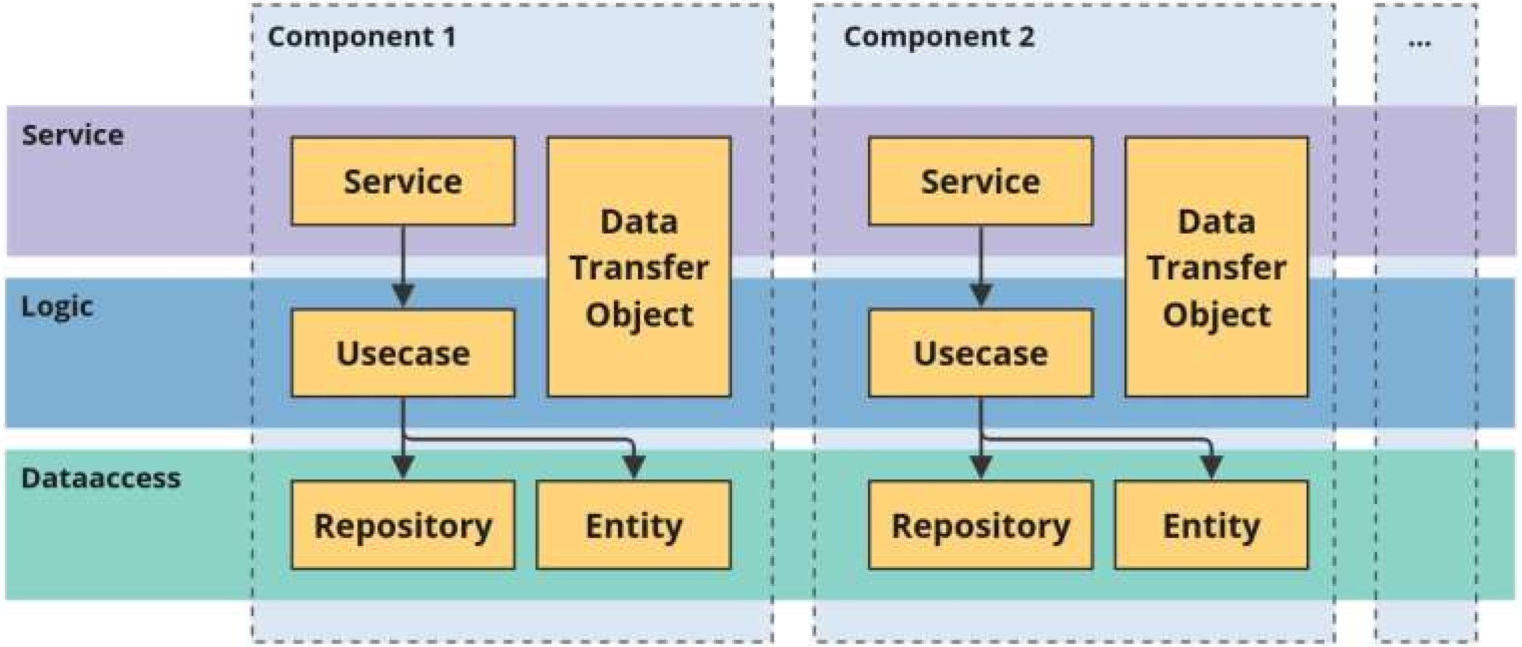
We prepare for iterating development and agile project goals and scope.

Security

We take into account that security is a central IoT challenge.



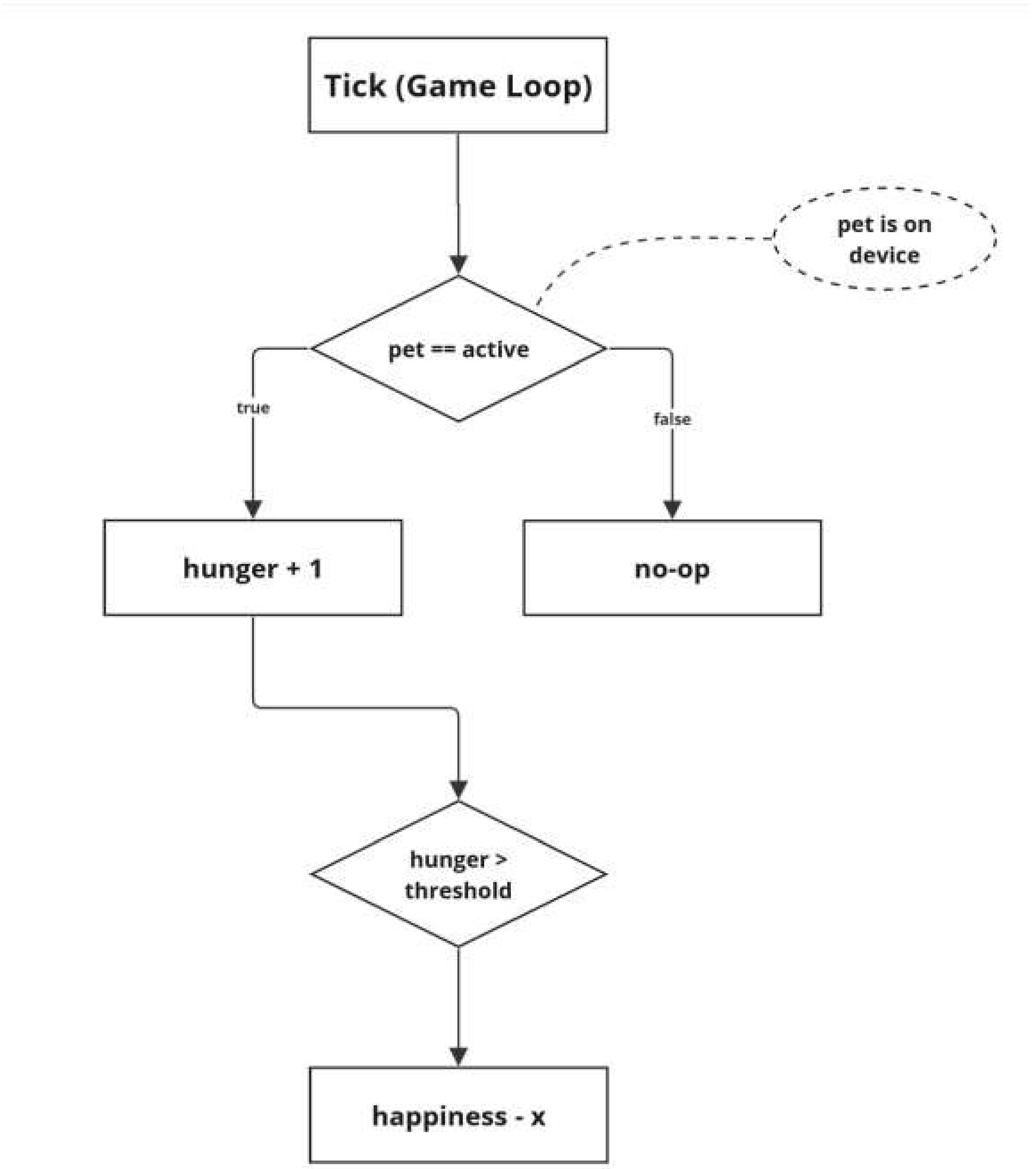
APPLICATION ARCHITECTURE



THE GAME LOOP

OR:

HOW DO THE ATTRIBUTES AFFECT THE HAPPINESS?



FRONTEND TEAM

- Samuel Costa
- Rares Stefea
- Yousef Taha
- Hamdy Elmorsy
- Yasuaki Kumazaki
- Omar Shaban

GENERAL CONCEPT

Giving the user an online platform where they can utilize the extended functionalities of the physical device

EXTENDED FUNCTIONALITIES

- Creation/Customization of the pets

  User is able to pick a **name** for their pet/s as-well as choosing the **type** of it.
  User will also be able to customize their pet/s with items such as clothes/toys.

- Better visualization

  Friendly user interface which shows the stats of the pets as-well as a better picture of the pet itself.

EXTENDED FUNCTIONALITIES

- Communication with other users (e.g. Friendlist)

  Allowing us to see other friends' pets on the leaderboard

- Notifications (e.g. Hungry/Sad)

  Will be displayed when the pet is below a certain stat threshold

EXTENDED FUNCTIONALITIES

- Statistics (e.g. Health/XP/Happiness)

  Stat bars will be shown to the user based on a 100% percentile

- Settings (e.g. Linked devices/User info)

  Ability to connect/disconnect devices from the user's account as-well as see the devices that are already connected and display the User's ID

EXTENDED FUNCTIONALITIES

- Achievements

  Will give the user points/items

- Challenges

  User/pet will gain XP from different challenges provided

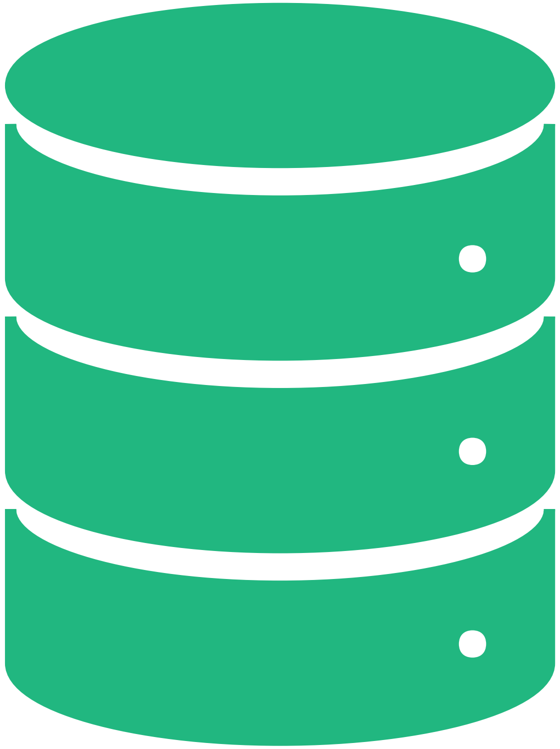
- Leaderboards

  Will give a ranking for the users/pets

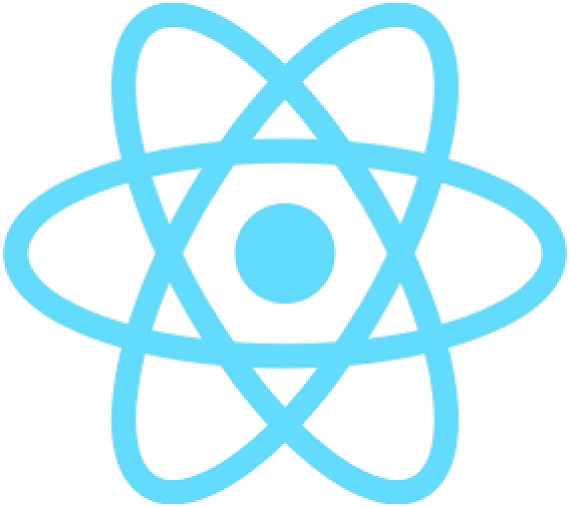
APP OVERVIEW

- **Home page/frame** – basic instructions about how to navigate the app
- **Sign up / login page** – user can sign up and make initial device linking and email
- **Pet page** – see own pet data and leader board ranking achievements , XP
- **Settings page** – user can see own data, disconnect device, add new devices sign out etc.
- **Friend page –** users can manage their friend list
- **Inventory page –** users can manage their rewards for completing challenges

DATA FLOW



TECHNOLOGIES



ReactJS

Flutter

THE END