MVP Completed

Contextualization

- **Already DONE**
  - Backend without Edge Data
  - Mobile App integrated with API
  - Website platform integrated with API

- **To Do**
  - MVP Deploy
  - Photo anonymization
  - ATCLL Integration

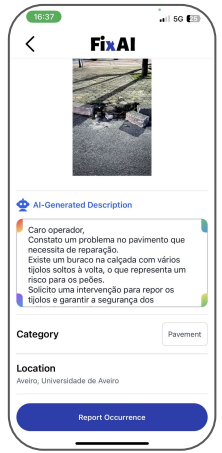
Mobile Application

- **React Native**
- **Authentication** with Bearer Tokens
- **Backend** integration

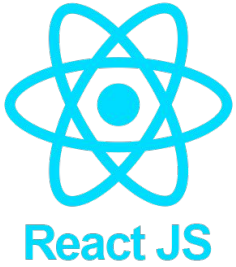
TODO:

- ➔ Refresh expired tokens
- ➔ Android 100% compatibility
- ➔ Compile mobile application (.apk & .ipa)

We need a **Mac computer** to compile a .ipa file for iOS



Web Platform



- **React JS**
- **Authentication** with Cookies
- **Backend** integration

TODO:

➔ Refresh expired tokens

➔ Future personas… (next slides)

Backend

- **FastAPI**
- **Tokens** management
- **H3-index** integration
- **Email Server** with SendGrid
- **Beta AI Generation** with Gemini
- **Stateless** for Kubernetes

TODO:

- ➔ Incidents aggregation with location
- ➔ Async Jobs (e.g. AI) with Kafka
- ➔ Anonymization

Databases

These are the only **stateful** services

- **Cassandra** for main database
- **Minlo** for photos database

TODO:

➔ Different buckets for different photos quality

➔ New features coming…

Auth

Auth.users

```
email text,

password_hash text,
created_at timestamp,
user_id uuid,
PRIMARY KEY (email)
```

Auth.operators

```
email text,

password_hash text,
created_at timestamp,
organization_id uuid,
operator_id uuid,
PRIMARY KEY (email)
```

FALTA: OAUTH com um Provider

Auth.confirmation_codes

```
email text,
code text, (TTL=5min)
PRIMARY KEY (email)
```

H3_index

H3_index.organization_regions

```
h3_index text,

organization_id uuid,
PRIMARY KEY (h3_index)
```

H3_index.incidents_regions

```
h3_index text,

incident_ids set<uuid>,
PRIMARY KEY (h3_index)
```

App_data

App_data.user_profile

```
user_id uuid,

name text,
email_notification_flag boolean,
PRIMARY KEY (user_id)
```

App_data.organization

```
organization_id uuid,

name text,
language text,
PRIMARY KEY (organization_id)
```

App_data.user_occurrence_by_status

```
user_id uuid,
status text,
time_id timeuuid,

occurrence_id uuid,
photo_id uuid,
photo_latitude double,
photo_longitude double,
category text,
PRIMARY KEY ((user_id, status), time_id);
```

App_data.occurrence_details

```
occurrence_id uuid,

user_id uuid,
organization_id uuid,
incident_id uuid,
time_id timeuuid,
photo_id uuid,
photo_latitude double,
photo_longitude double,
description text,
category text,
status text,
PRIMARY KEY (occurrence_id)
```

App_data.category

```
organization_id uuid,
category text,

description text,
num_pending int,
num_in_progress int,
num_resolved int,
PRIMARY KEY (organization_id, category)
```

App_data.incident_details

```
organization_id uuid,
incident_id uuid,

first_occurrence_time_id timeuuid,
first_photo_id uuid,
num_occurences int,
severity text,
centroid_latitude double,
centroid_longitude double,
main_description text,
main_category text,
status text,
PRIMARY KEY (incident_id);
```

App_data.incident_occurrences

```
organization_id uuid,
incident_id uuid,
occurrence_time_id timeuuid,

photo_id uuid,
photo_latitude double,
photo_longitude double,
occurrence_id uuid,
user_id uuid,
PRIMARY KEY (incident_id, occurrence_time_id)
WITH CLUSTERING ORDER BY (occurrence_time_id DESC);
```

App_data.incident_by_status

```
organization_id uuid,
first_occurrence_time_id timeuuid,
status text,
main_category text,
incident_id uuid,

first_photo_id uuid,
severity text,
centroid_latitude double,
centroid_longitude double,
num_occurrences int,
PRIMARY KEY ((organization_id, status), first_occurrence_time_id)
WITH CLUSTERING ORDER BY (first_occurrence_time_id DESC);
```

App_data.incident_by_status_and_category

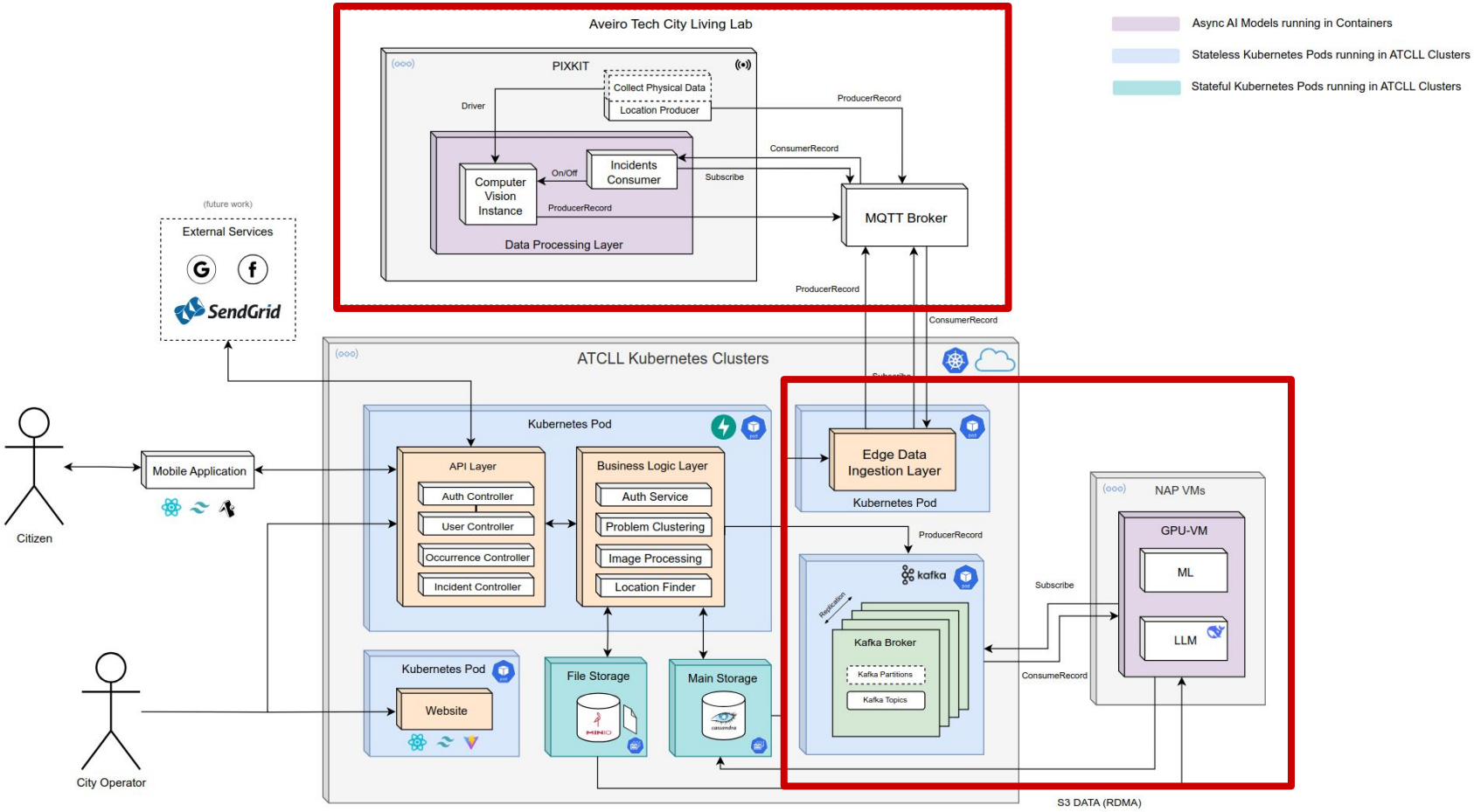
```
organization_id uuid,
first_occurrence_time_id timeuuid,
status text,
main_category text,
incident_id uuid,

first_photo_id uuid,
severity text,
centroid_latitude double,
centroid_longitude double,
num_occurrences int,
PRIMARY KEY ((organization_id, status, main_category), first_occurrence_time_id)
WITH CLUSTERING ORDER BY (first_occurrence_time_id DESC);
```

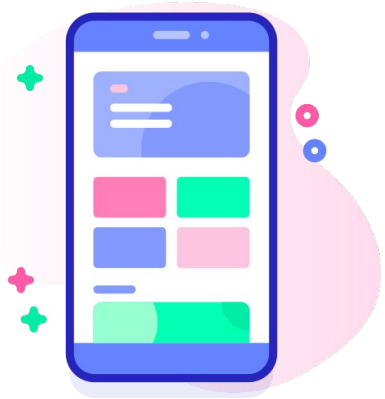
cassandra

- **High** duplication
- **Low** consistency
- **High** speed
- **High** scalability

To do:


Async AI Models running in Containers
Stateless Kubernetes Pods running in ATCLL Clusters
Stateful Kubernetes Pods running in ATCLL Clusters
Aveiro Tech City Living Lab
PIXKIT
Collect Physical Data
Location Producer
Driver
ProducerRecord
ConsumerRecord
Computer Vision Instance
On/Off
Incidents Consumer
Subscribe
ProducerRecord
Data Processing Layer
MQTT Broker
ProducerRecord
ConsumerRecord
(future work)
External Services
SendGrid
ATCLL Kubernetes Clusters
Kubernetes Pod
API Layer
Auth Controller
User Controller
Occurrence Controller
Incident Controller
Business Logic Layer
Auth Service
Problem Clustering
Image Processing
Location Finder
Edge Data Ingestion Layer
Kubernetes Pod
ProducerRecord
kafka
Replication
Kafka Broker
Kafka Partitions
Kafka Topics
NAP VMs
GPU-VM
ML
LLM
Subscribe
ConsumeRecord
Citizen
Mobile Application
City Operator
Kubernetes Pod
Website
File Storage
Main Storage
S3 DATA (RDMA)

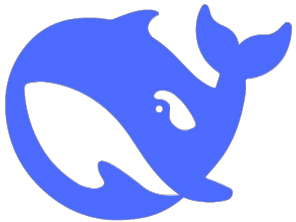
Demo



AI Integration Solutions

ChatGPT

Gemini



deepseek

LLaVA

Comparison

Feature				
Multimodal Capability	✅	✅	✅	✅
Scalability	✅	✅	✅	❌
Free Tier	❌	✅	✅	✅
Accuracy	✅	✅	❌	✅
Self-hosted	❌	❌	✅	✅
for 1M tokens:	5.0€	0.1€	GPU price	

Gemini

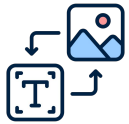
Paid Version

Average Total Tokens: 2000 per request

Token Price: 100 000 -> 0.01€

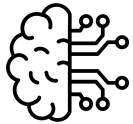
Gemini

- **Multimodal Capability**: The LLM must handle image inputs and generate text outputs (description and classification).



- **Customizable Outputs**: Supports tailored responses via prompts, allowing specific classification labels or descriptive styles as needed for your project

- **Contextual Reasoning**: Analyzes the image in the context of the provided prompt, enabling accurate categorization (e.g., accident, flooding, vandalism).



- **Free Tier Access**: The Gemini API offers a free tier with a limit of <NUMBER> requests per minute, ideal for prototyping and small-scale urban incident analysis projects.

>_ Context

You are a concerned **citizen** reporting urban issues to city operators.

Analyze the image and return a JSON object with keys **"description"**, **"category"**, and **"severity"**.

Use %s for language, classify the issue as one of %s, and assign a severity level ["low", "medium", "high"].

Begin the **"description"** with a general greeting that includes an operator introduction. Be concise and clear in your description without being verbose, you should include changes of line in the description.

In description, your task is to analyze the image and describe the problem in three detailed sentences.

Include specific details such as concerns or any other relevant visual elements that help resolve the issue.

Focus solely on the issue without using offensive language.

Gemini

tests > tree.jpeg

Description

Category
+
Severity

Gemini

tests > rain.jpg

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 1 COMMENTS backend - tests

```
(feature/llm_integration) tests > python3 llm_client.py
Olá,

Gostaria de reportar um problema de acumulação de água significativo numa rua residencial após uma forte chuva. A água está a cobrir grande parte da via, tornando a passagem de veículos e peões perigosa, especialmente na área da passadeira. A drenagem inadequada parece ser a causa, necessitando de atenção urgente para evitar acidentes e danos adicionais.

[urban_drainage, medium]

Input tokens: 1974
Output tokens: 87
Total tokens: 2061
(feature/llm_integration) tests >
```

New Requirements after CMA feedback

Project Validation

- ➔ PIXKIT
- ➔ AI
- ➔ New Persona
- ➔ Requirements Change

Application Goal & Project Deadlines