

# Correct Exam

Modern software architecture in practise

Press Space for next page →





# My mojo

<https://www.linkedin.com/pulse/i-have-right-do-research-software-engineering-hafedh-mili/>

Parnas noted "I would never have realized the nature of the problem, unless I had been working on that project, reviewing development documents, and sitting at that lunch table". Well, minimally, I need to be able to understand the conversation at that lunch table!

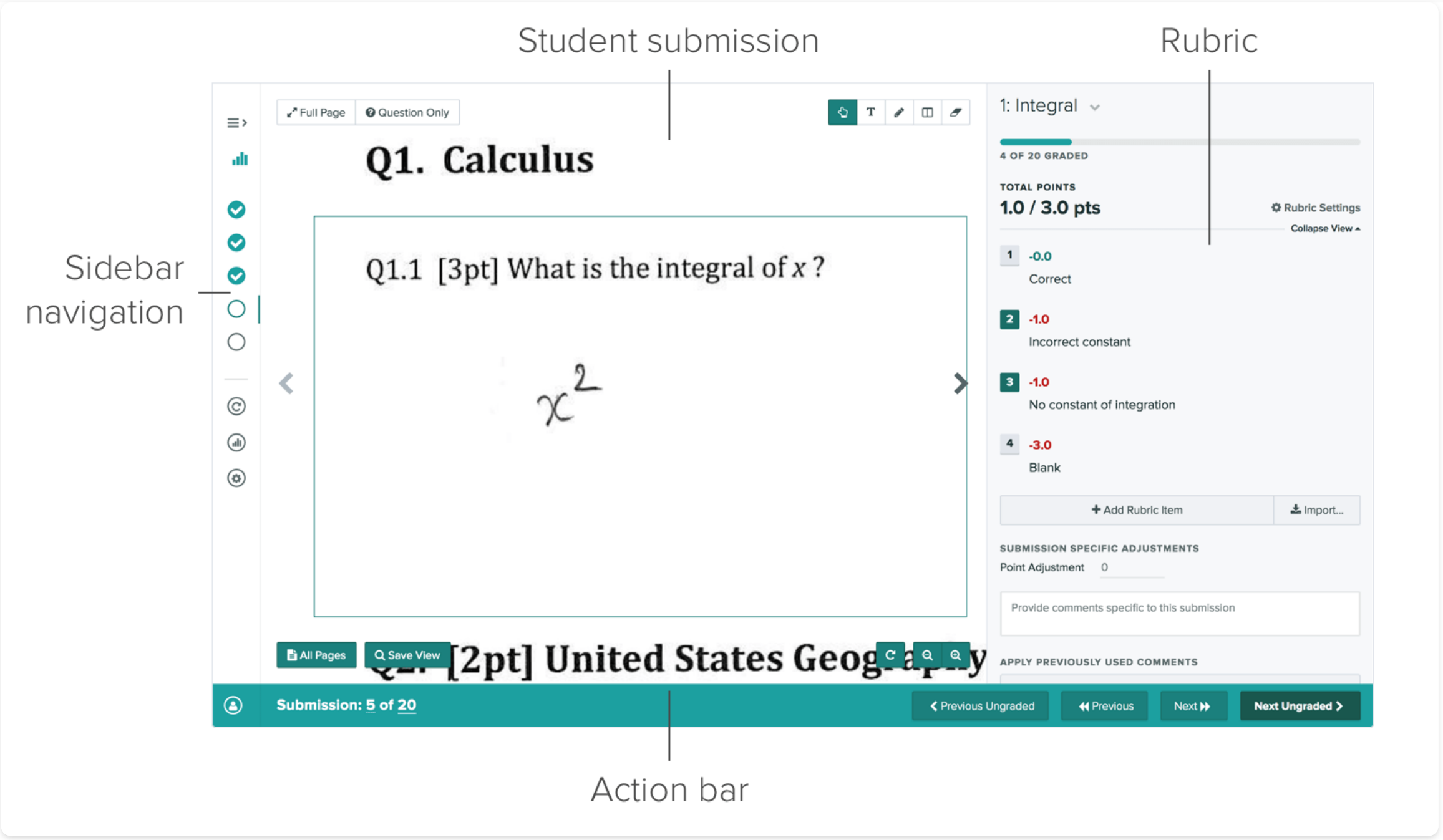
So here is a practical definition of what "understanding the conversation" means in this context: **You have no credibility to do software engineering research unless you have at least the development skills/vocabulary of your graduating bachelor students.**

That is why: I enjoy building real software, doing consultancy, working with students, ...

The project: correct exam

# Highly inspired by GradeScope Solution

Gradescope grading software allows students to receive faster and more detailed feedback on their work, and allows instructors to see detailed assignment and question analytics. It is an easy way to take submissions digitally in order to preserve the original work and allow for quick and easy viewing from anywhere.



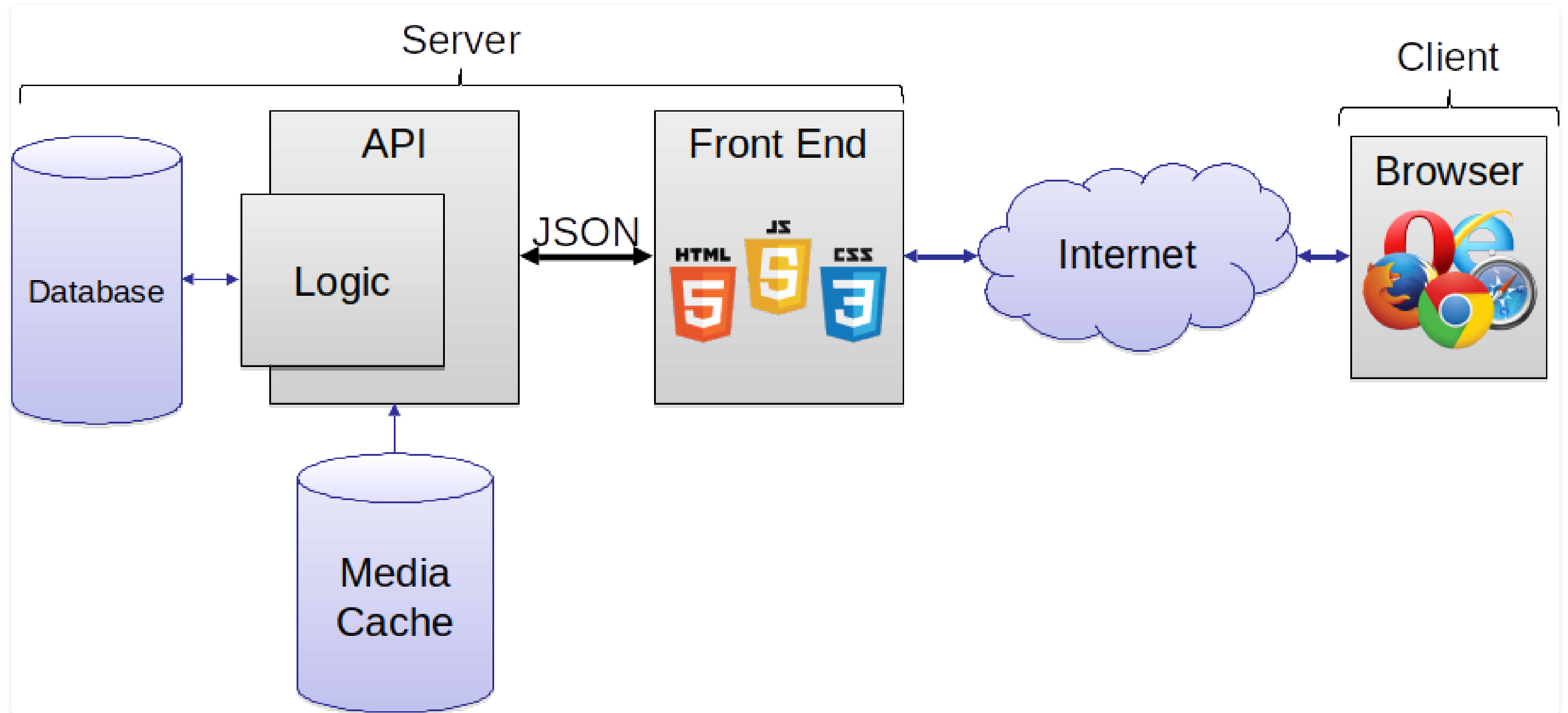
# Why building that piece of software ?

- Enable to correct exams during meetings 😊
- Save \$5 per student copy
- Try to initiate a project with Younup to see how we could engage efficiently software engineer available for some days
- Create an open source implementation of real software with complex architecture to have a case study for
  - experiments in software engineering research
  - explaining modern software architecture to students
- Try to keep the credibility (in my vision) to do research in software engineering

# The technical architecture

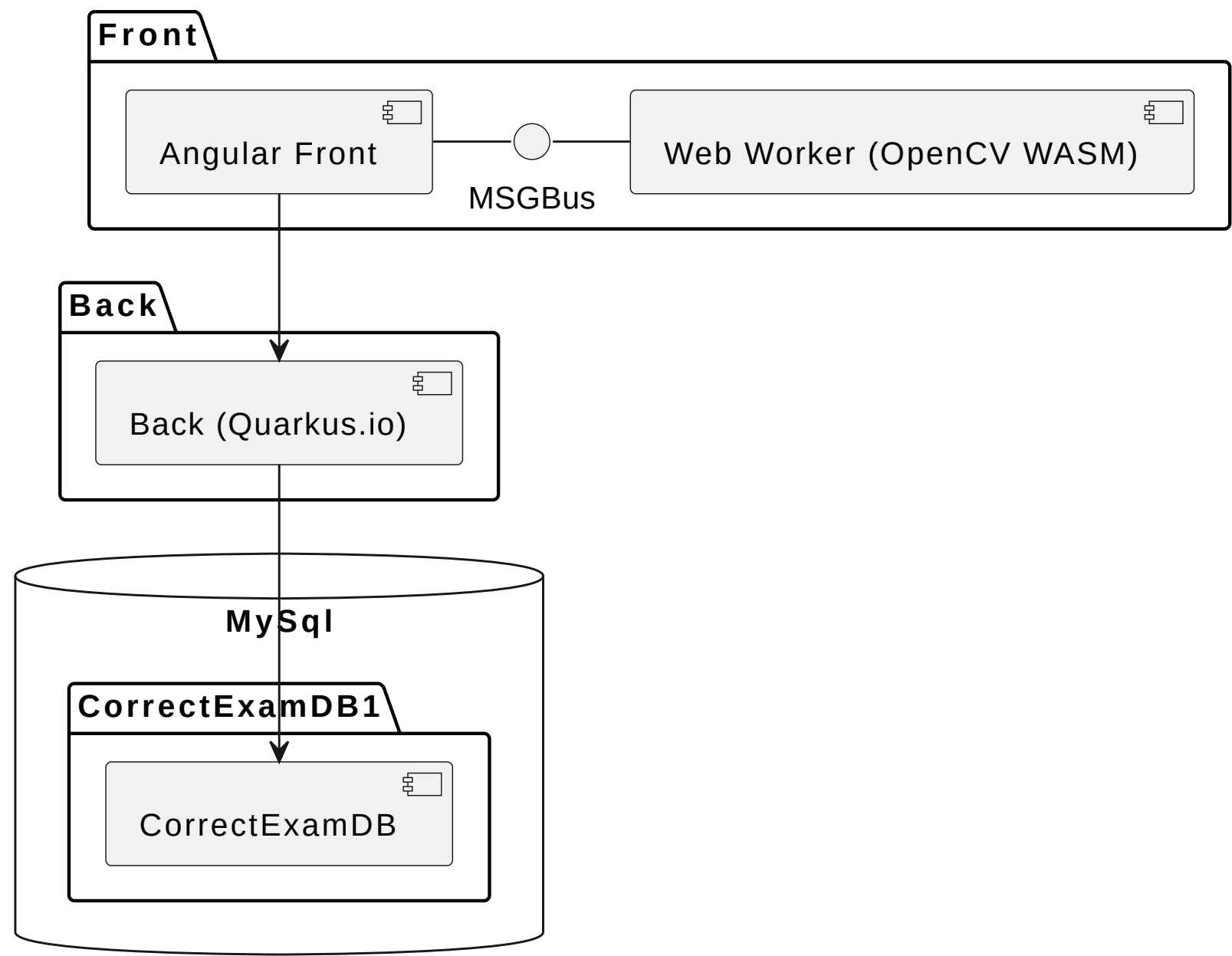
- Quarkus for the back (Java + native compilation through GraalVM)
- Angular 13.3 for the front
  - pdf.js to play with pdf (exam, scan exam, feedback for students)
  - fabric.js to draw on top of a pdf
  - opencv in wasm within a web worker to analyse the scan
  - tensorflow JS with the browser for digit and letter recognition
  - ...
- Docker and K8S to deploy the back and the monitoring layer
- Front is hosted in a CDN to follow the JamStack architecture (currently github page)
- CI/CD using github action, dockerhub webhook, and gowebhook
- Pluggable architecture based on micro-services to integrate question type automatic correction (let developer use their own programming language to provides solutions that automatically correct some questions)

# Architecture overview

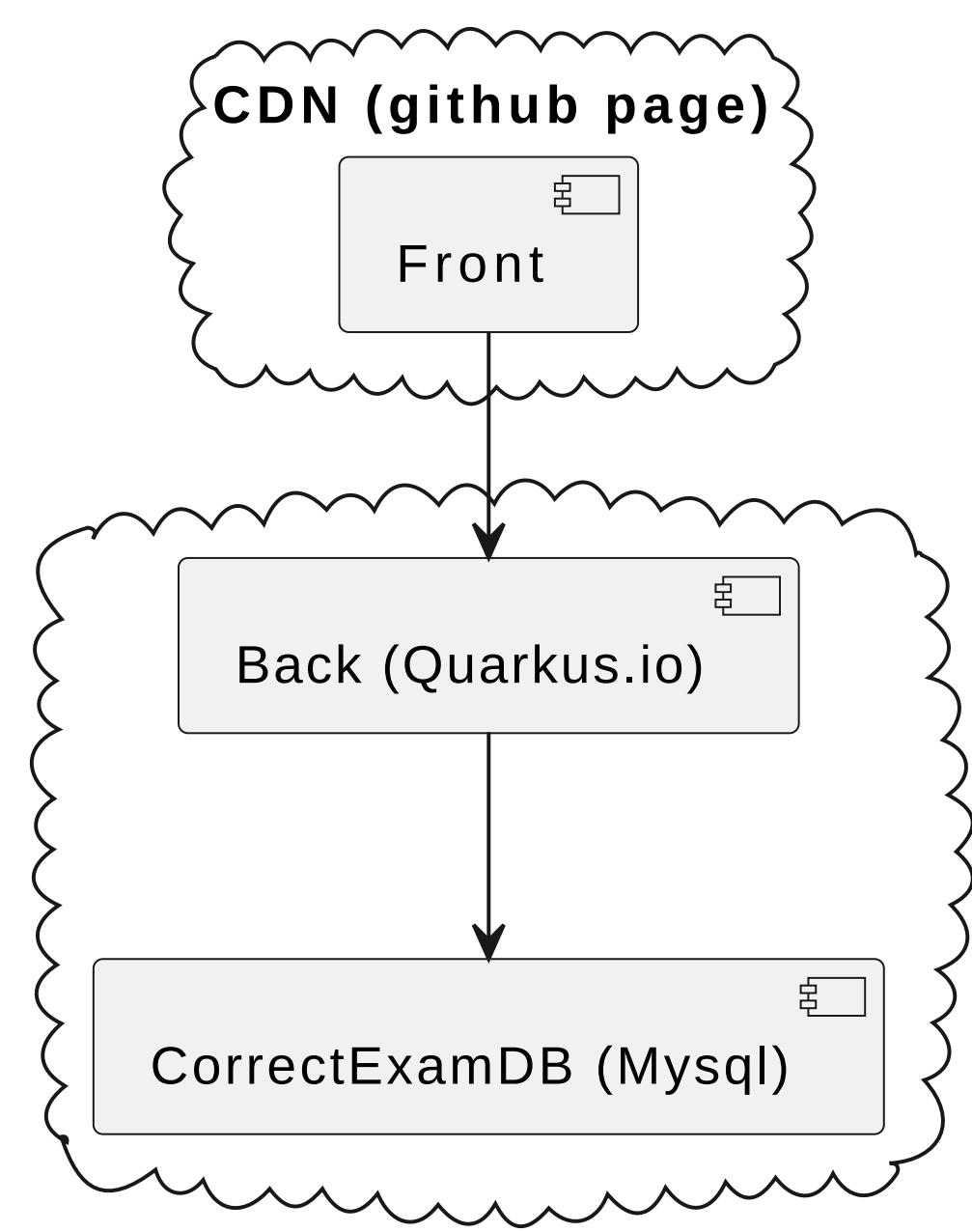


# Diagrams

Conceptual architecture

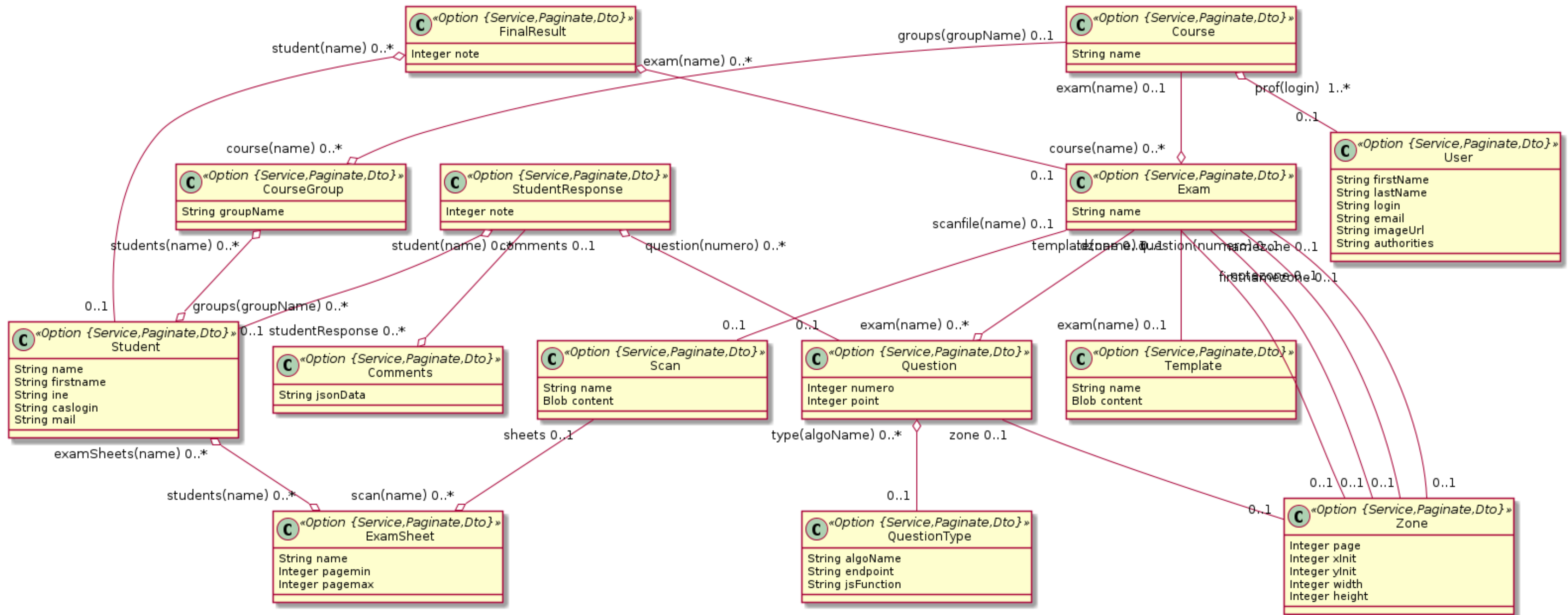


deployed architecture





# Business model



# JHipster to generate the application skeleton

JHipster is a development platform to quickly generate, develop, & deploy modern web applications & microservice architectures.

- `jhipster 7.7.0` for the front
- `jhisper 6.10.15`

Manual alignement of the API evolution 😊 (@Djamel)



# Where we are

## Done

- [x] Vue home
- [x] Donnée factice
- [x] Afficher vraie liste de cours (requête au back)
- [x] Masquer vue entité pour user
- [x] Première page
- [x] Composant Cours viusualisation
- [x] Composant Insérer groupe étudiant
- [x] Composant supprimer UE
- [x] Voir groupe étudiant pour une UE incluant VIDER LISTE ETUDIANT pour cet UE
- [x] Refactoring module
- [x] Composant Créer exam

# Done

- [x] Composant ExamEditorzone
  - [x] Créer type de rectangle fabric
- [x] Vérifier compilation native
- [x] Vue question
- [x] Mise en place CI
- [x] Mise en place CD (front)
- [x] Vue home admin
- [x] Bouton retour dans la partie annoter document
- [x] Bouton charger scan
- [x] Bouton associé copies
- [x] Bouton Corriger
- [x] Intégration opencv
- [x] Test intégration opencv
- [x] Mise en place CD (back)



# Ongoing

- [ ] Backup alignement pdf
- [ ] Découpe numéro (bounding box, opencv)
- [ ] Association étudiants (angular)
- [ ] Icône verte dans le workflow de correction

# TODO

- ☐ TensorFlow.js, reconnaissance chiffre (match étudiant)
- ☐ Bouton voir et exporter résultat
- ☐ Bouton envoyer résultats aux étudiants
- ☐ Bouton voir copie corrigée
- ☐ Bouton associé copies
- ☐ Bouton Corriger
- ☐ Bouton voir et exporter résultat
- ☐ Bouton envoyer résultats aux étudiants
- ☐ Bouton voir copie corrigée
- ☐ Traduction title route
- ☐ Traduction composant module scanexam
- ☐ Retravailler traduction
- ☐ Mettre à jour la vue du numéro de question quand mise à jour dans le menu propriété.



# TO Think

- ☐ Réfléchir barème positif ou négatif
- ☐ Réfléchir annotation automatique par question
- ☐ Module alignement image
- ☐ Module reconnaissance nom

# To fix

- ☐ Configuration email back
- ☐ Remove unused dependencies (crop ngx, opencvngx)

# Fixed

- ☒ Exam
- ☒ Question
- ☒ ExamSheet
- ☒ Final Result
- ☒ Student Response
- ☒ Comments
- ☒ opencv path in the deployed version



# Demo and code overview

- demo
- repo back
- repo front

# A simple RPC to manage the communication between openCV worker and Angular

In the worker

```
addEventListener('message', e => {  
  switch (e.data.msg) {  
    case 'hello': {  
      const response = `worker response to ${e.data.msg}`;  
      postMessage({ msg: response, uid: e.data.uid });  
      break;  
    }  
    ...  
  }  
});
```

## In the service

```
ready!: Promise<void>;
subjects = new Map<string, Subject<any>>();
constructor() {
  if (typeof Worker !== 'undefined') {
    // Create a new
    this.worker = worker;
    this.worker.onmessage = ({ data }) => {
      if (this.subjects.has(data.uid)) {
        this.subjects.get(data.uid)?.next(data.payload);
        this.subjects.get(data.uid)?.complete();
      }
    };
    this.worker.onerror = e => {
      if (this.subjects.has((e as any).uid)) {
        this.subjects.get((e as any).uid)?.error(e);
      }
    };
  };
  const p = new Subject();
  this.subjects.set('0', p); // Ask to load opencv wasm to the worker
  this.ready = new Promise((res, rej) => {
    this.subjects
      .get('0')?.asObservable().subscribe(() => {res();}, () => rej());
  });

  this.worker.postMessage({ msg: 'load', uid: '0' });
```



## In the service

```
private _dispatch<T>(msg1: any, pay: any): Observable<T> {
  const uuid1 = uuid(); // ⇒ '9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d'
  this.ready.then(() => {
    this.worker.postMessage({ msg: msg1, uid: uuid1, payload: pay });
  });
  const p = new Subject<T>();
  this.subjects.set(uuid1, p);
  this.ready = new Promise((res, rej) => {
    this.subjects.get(uuid1)?.asObservable().subscribe(() =>
      {res();},
      () => rej()
    );
  });
  return p.asObservable();
}

public imageProcessing(image: ImageData): Observable<ImageData> {
  return this._dispatch('imageProcessing', image);
}
```

# OpenCV architecture

## Alignement algo

- Marker detection
  - circle detection
  - verify if the circle is filled
  - If we find 4 markers on the original exam and in the scan copy

```
let h = cv.findHomography(dstTri, srcTri, cv.RANSAC);  
cv.warpPerspective(srcMat2, dst, h, dsize);
```

- ELSE find markers based on AKAZE (longer process), filter found marker with custom heuristic

```
let h = cv.findHomography(dstTri, srcTri, cv.RANSAC);  
cv.warpPerspective(srcMat2, dst, h, dsize);
```

# Plug your own microservice to correct a custom question type

- Two kinds: *Preprocessor*, *Automatic correction*

correct(qs: Questions)



# Learn More

[Documentations](#) · [GitHub](#) · [Showcases](#)