- Home
- Candidates

---

# Assignment

## Part one

Like you may already know, Emi is used by huge companies to massively screen candidates. They apply through a job board like Zona Jobs, Bumeran, Indeed, etc. Once they apply, they receive an SMS and an e-mail to start chatting with Emi.

Then, when the screeing interview is finished, the results are processed by a backend application that classifies candidates in two groups: *approved* and *rejected*.

## Your goal:

- Create a new backend application that provides the necessary endpoints to retrieve the hardcoded in testData.js. You can use any programming language or framework you feel comfortable with (we suggest using Node).
- Then, use this new backend to fetch the candidates and the columns to display (not all recruiters want to see the same data).
- List all candidates.
- Add (or at least, name) other desirable features you can think of, for this list view.

Note: How do we know if a candidate was approved or rejected? Each candidate has a 'reason' field. If the field is empty it means that the candidate was approved. On the other hand, if the value is not empty, it means that the candidate was rejected. There may be more than one rejection reason. E.g. salary out of range or minimum age not met.

## Part two

After displaying the data we want to update it.

The backend learns how to classify candidates with the feedback of recruiters. This is core part of the feedback loop for this AI.

This is why recruiters may occasionally need to re-classify a candidate. This is a manual process that is done based on the results that are shown on the previous step.

# Your task:

Allow the recruiters to reclassify candidates.

**Manually approving a candidate**

1. Hit the API endpoint to remove the rejection reasons assigned to a candidate.
2. Make sure the table is refreshed with the new data.

**Manually rejecting a candidate**

1. Hit the API to list all the available rejection reasons.
2. Let the recruiter select one or more rejection reasons.
3. Hit the API again to update that candidate's data.

Note: for this second part, you are allowed to handle state locally, so just add dummy endpoints in your API to be able to do so.

# Guidelines for part 1 and 2

1. This is a productive application, so it's expected you make changes you would do as if you were working in a real environment.
2. Adapt and create React components using the application in this repository to solve the previous requirements.