Awesome! Based on the final n8n workflow you provided (`CrowdWisdomTrading.json`), here's a comprehensive summary of **all the major problems faced** and **how you solved each** during the development process:

□ Challenges & Solutions Recap for `CrowdWisdomTrading`

□ 1. **Invalid JSON in HTTP Request Node**

Problem:

Directly inserting expressions like `={{ $json["prompt"] }}` into a raw `jsonBody` field triggered:

> *`NodeOperationError: JSON parameter needs to be valid JSON`*

Cause:

You can't use `={{ ... }}` inside JSON keys/values unless it's a structured JSON input or handled through a node beforehand.

Solution:

□ Used a `Code` node (`setFR`, `setAR`, `setDE`) to generate the correct request body JSON for OpenRouter:

```
{
  json: {
    body: {
      model: "...",
      messages: [...]
    }
  }
}
```

□ 2. **Model Output Appeared in `reasoning`, Not `content`**

Problem:

Translated output was expected in `.message.content`, but instead came in `.message.reasoning`.

Cause:

The selected OpenRouter model (`arliai/qwq-32b-arliai-rpr-v1`) uses `reasoning` for outputs.

Solution:

 Updated all code nodes (e.g. `Generate Binary FR`, `Generate Binary AR`) to use:

```
$input.first().json.choices[0].message.reasoning
```

---

3. **Buffer and btoa Not Defined in Function/Code Node**

Problem:

Using `Buffer.from(...)` or `btoa(...)` caused errors:

> *`Cannot find name 'Buffer'` or `btoa is not defined`*

Cause:

n8n's `Code` node runs in a strict VM and doesn't support Node.js or browser globals.

Solution:

 Replaced them with:

- `TextEncoder` + custom base64 encoding function
- Full UTF-8 and Base64 encoder from scratch (in `Generate Binary *` nodes)

---

4. **Red Lines Under `binary` and `Buffer` Keywords**

Problem:

Even if the node executed, n8n showed red squiggly lines under certain keywords like `binary` and `Buffer`.

Cause:

Type definitions in the editor flagged them as not existing in the current context.

Solution:

 Ignored type warnings

 Avoided `Buffer` entirely

 Used dynamic return structure in safe JavaScript-only encoding

---

□ 5. **Incorrect Binary Output Caused Write Node Failure**

Problem:

Write node gave:

> *"The first argument must be of type string or an instance of Buffer..."*

Cause:

Returned item was missing actual binary data under `binary.data`, or the encoding was invalid.

Solution:

□ Returned data like this:

```
return [{
  binary: {
    data: {
      data: base64String,
      mimeType: 'application/octet-stream',
      fileName: 'daily-report-fr.txt'
    }
  }
}];
```

□ Matched the `dataPropertyName = data` in the Write node

□ 6. **Workflow Modularity Issues**

Problem:

Workflow was becoming messy due to hardcoded prompt construction and duplication.

Solution:

□ Modularized into reusable translation and file-writing paths:

```
Summarize → Translate (FR/AR/DE) → Generate Binary → Write File
```

Each handled in its own branch using prompt → body → response → file output.

□ 7. **Incorrect Query Parameter Binding**

Problem:

In the NewsAPI node, dynamic values like `from` and `to` needed to be injected properly.

Solution:

 Created a `Code` node to compute and inject:

```
{
  from: oneHourAgo.toISOString(),
  to: now.toISOString()
}
```
