**Case Study**

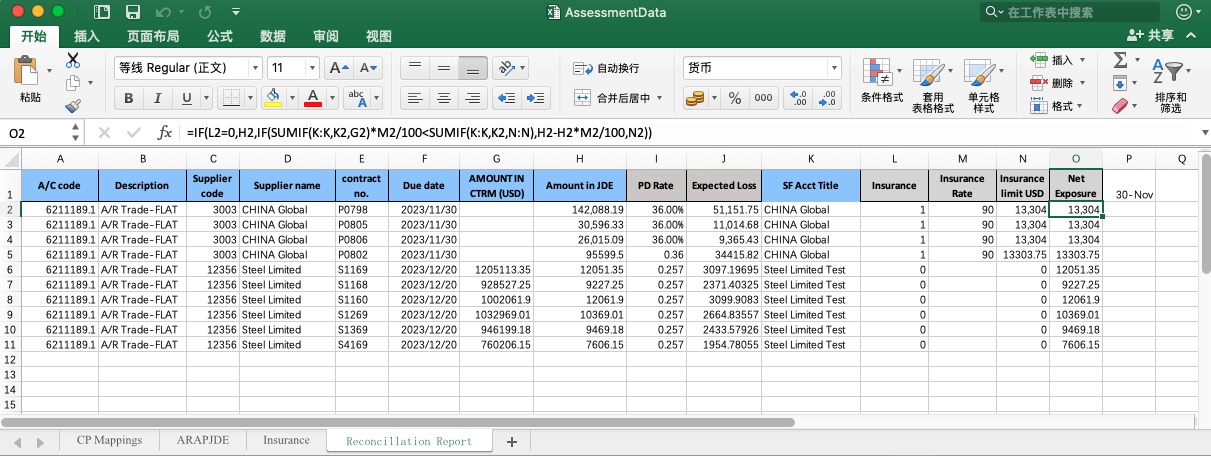
1. **Requirements**

|  |
| --- |
| Scope: API + UI + DB |

The expectation is for the candidate to code up a .NET core solution with React for this case study. You are free to choose the libraries that you are familiar with to solve the problem.

You are expected to upload the coded solutions (API & UI) to the GIT and kindly share the link with us.

**Input:**



**Output:**

* Design document link: [Case Study](https://atg8lg0m8r.feishu.cn/docx/NbJCdZw4NoYFfpxRCM6c1jmPnVd)
* Git link: https://github.com/dandebin/case\_study

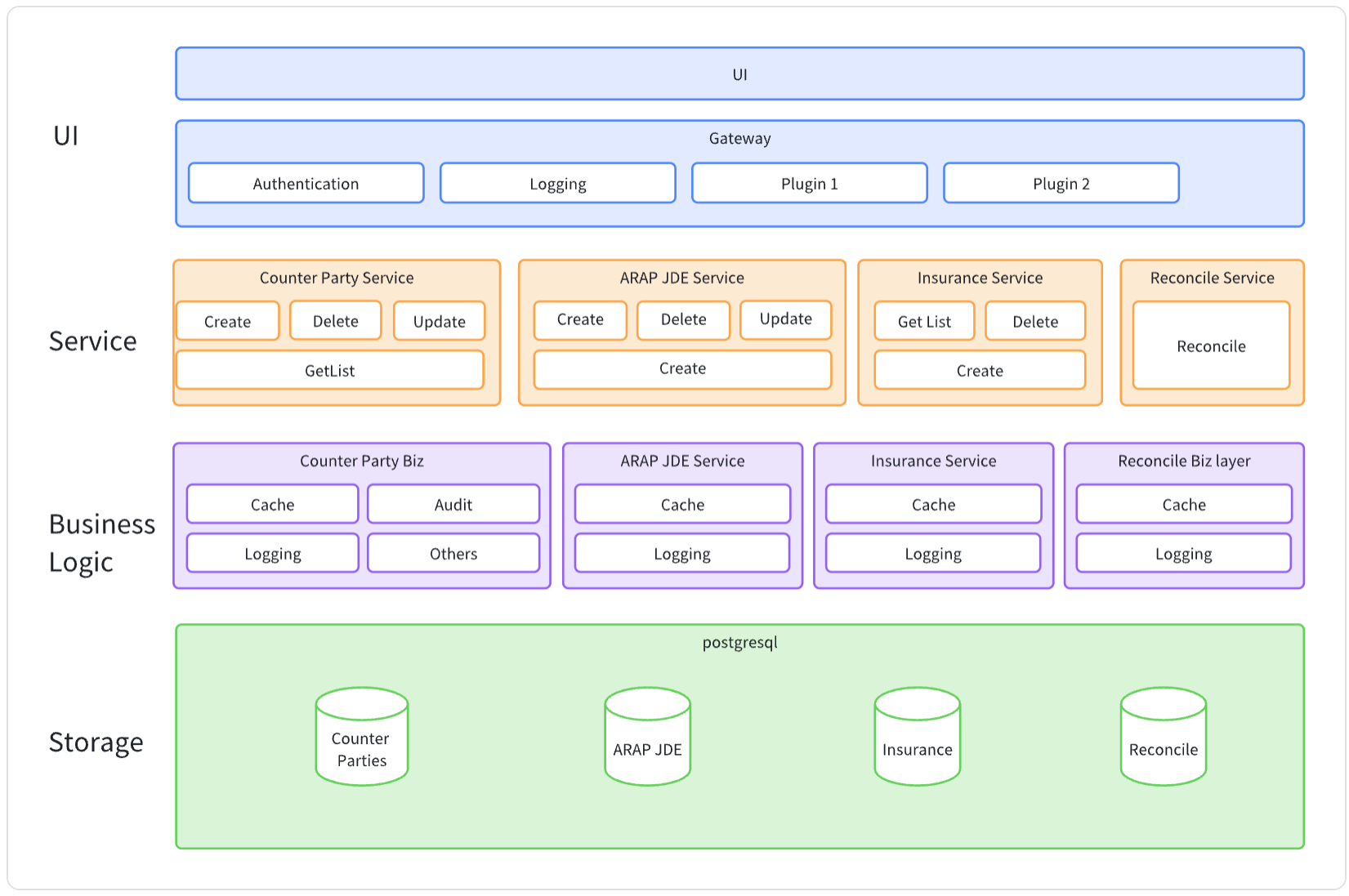
2. **Assumptions**

* Counterparty volume: <10,000
* Insurance items volume: <10,000
* ARAP JED volume: <100,000
* Daily reconciliation volume: <100,000

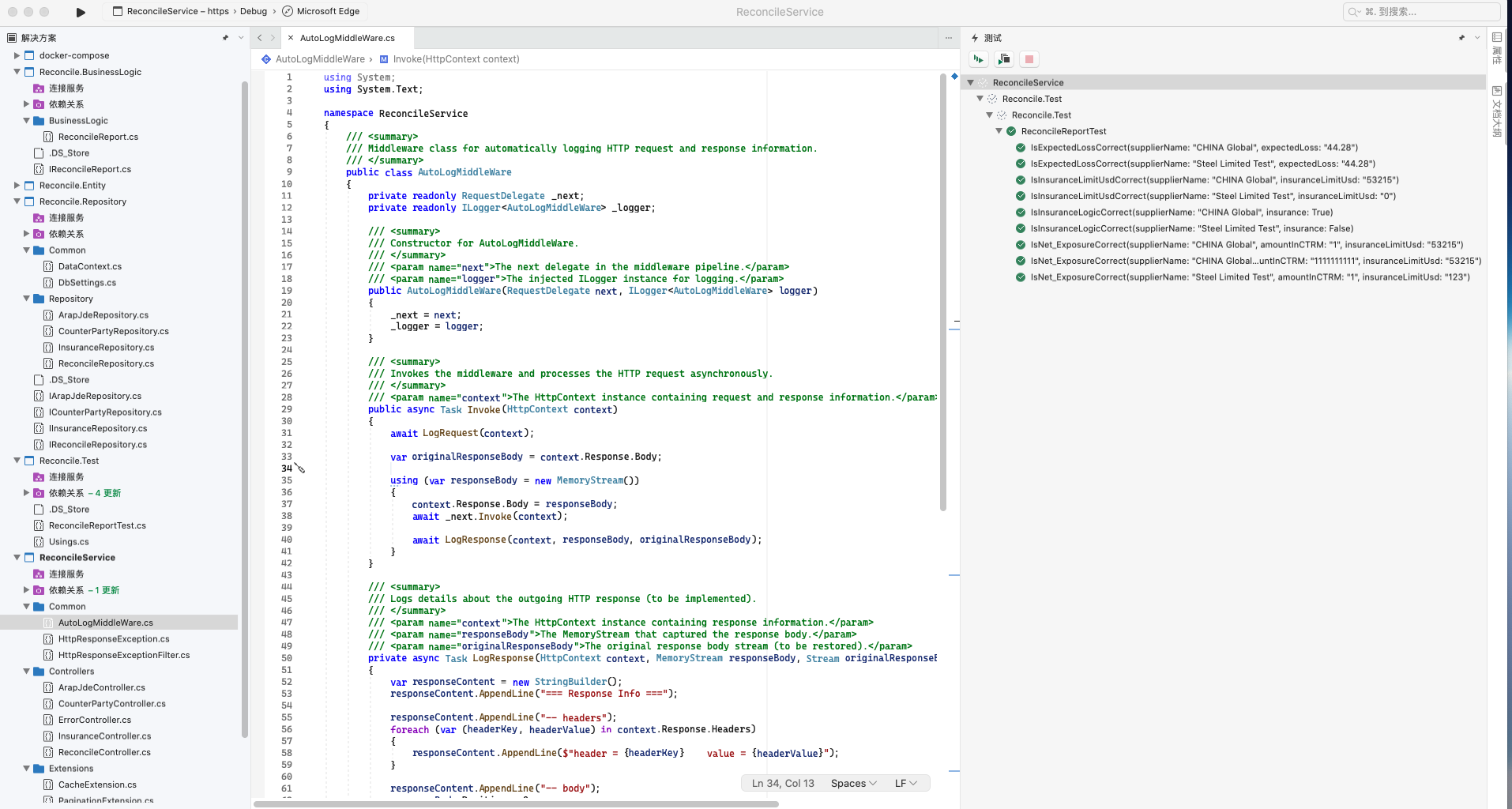
3. **Techniques**

|  |  |  |
| --- | --- | --- |
| **Functions** | **Tech** | **Comments** |
| Client | React Admin | **Use pre-built React Admin components:** This clarifies that you'll be leveraging existing components within the React Admin framework, not building a custom template from scratch.  **Integrate bespoke UI components:** "Bespoke" emphasizes the custom nature of these components, tailored to your specific needs.  **Implement uniform pagination for a seamless user experience:** This highlights the focus on user-friendliness when navigating through large datasets.  **Decouple the UI from backend API calls for independent development:** This emphasizes the separation of concerns and potential for parallel development of UI and backend.  **Centralize backend API calls with a consistent approach:** This conveys the importance of a unified and maintainable way to interact with the backend API. |
| Service | .Net 7.0 | **Leverage Memory Cache for enhanced API performance:** This emphasizes the positive impact on response times.  **Utilize Dapper for data mapping and stored procedure execution:** This clarifies the specific functionalities of Dapper in your project.  **Implement a comprehensive exception handler:** This emphasizes the robustness of the system in managing errors.  **Integrate logging middleware for detailed request/response tracking:** This highlights the comprehensiveness of the logging solution. |
| DB | Postgresql | PostgreSQL (Secure against SQL injection) |
| CICD | Git, Jira, AZure | Docker compose |
| UT | XUnit | Behavior Driven Development, Unit test first |
| Others | ChatGPT4  Gemini | **Enhanced Development Support:**   * **Copilot:** Assists with code generation. * **Large Language Models (LLMs) like ChatGPT/Gemini:** * **Improved Commenting:** Generate comprehensive and informative comments explaining code functionality. * **Streamlined Troubleshooting:** Assist in identifying potential issues and suggesting solutions based on code analysis. |

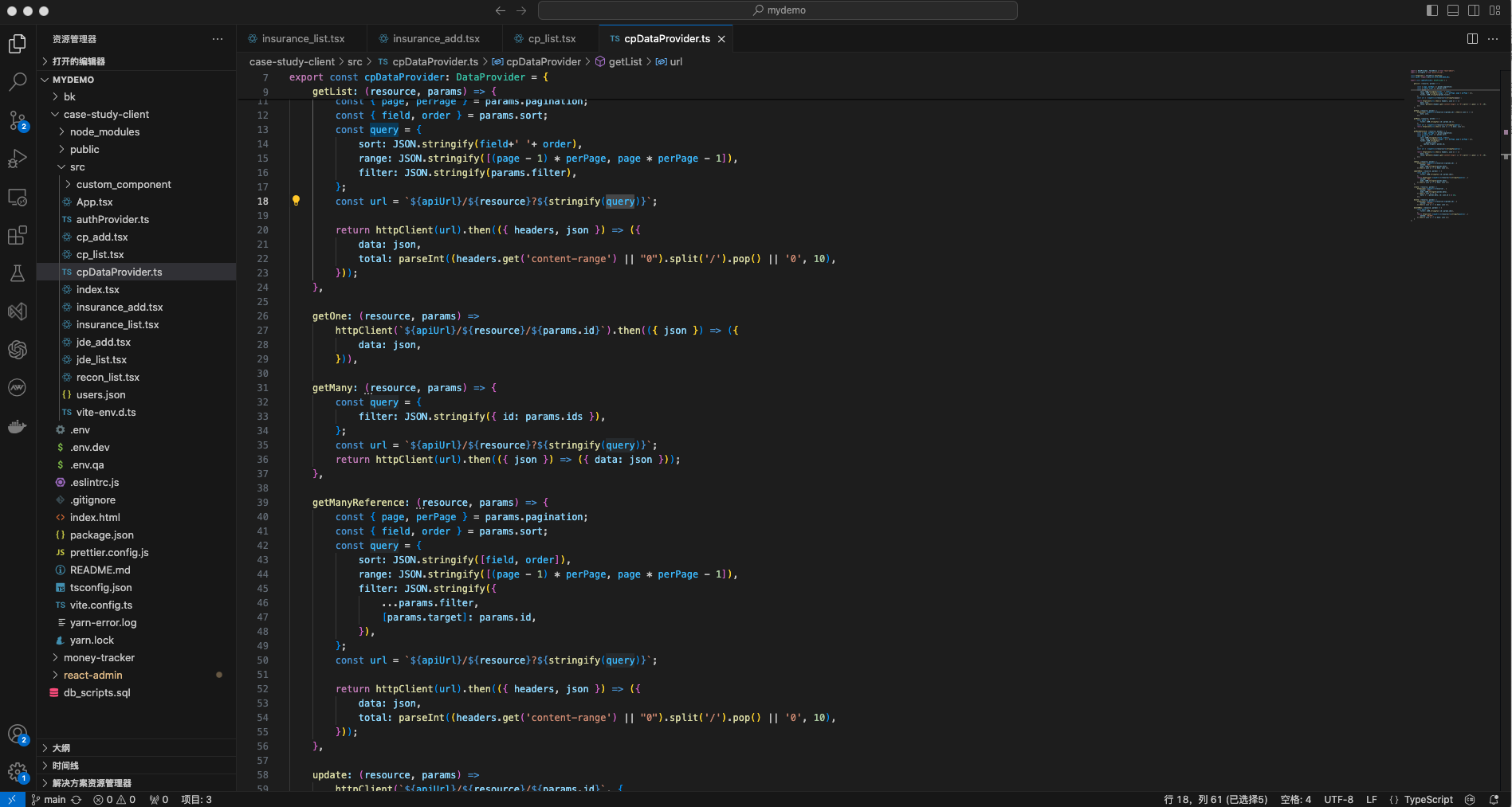
4. **Design**



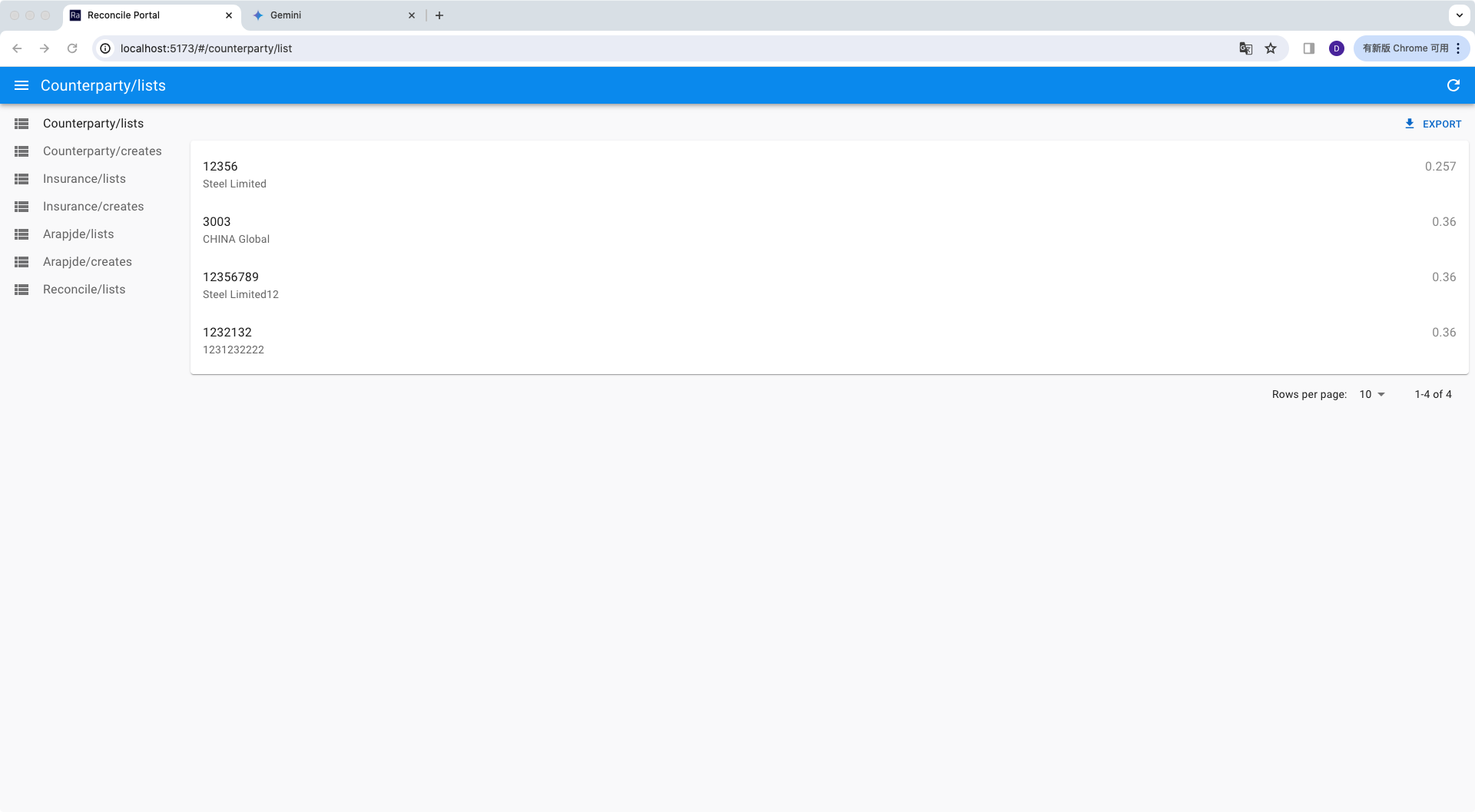
* Service UI

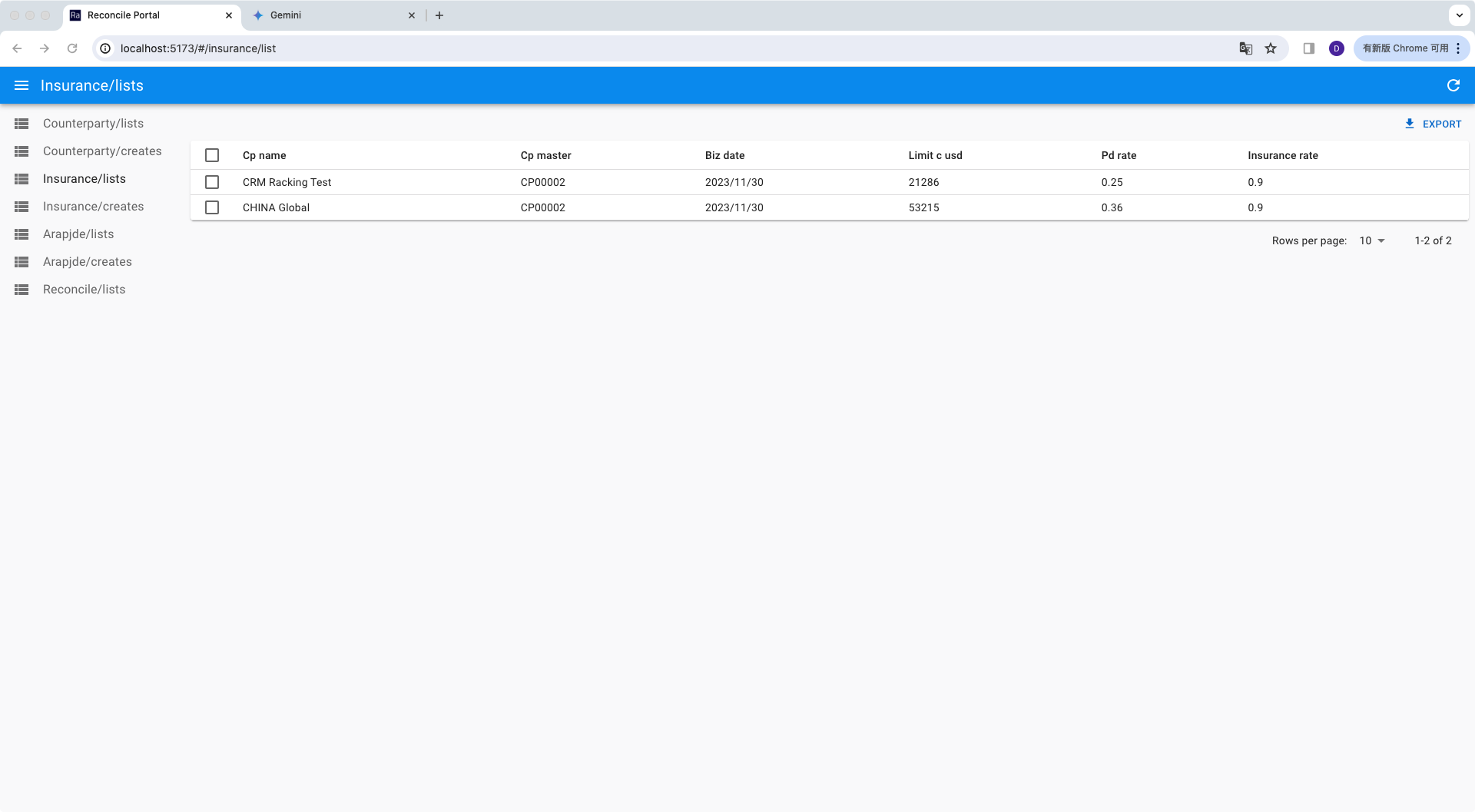


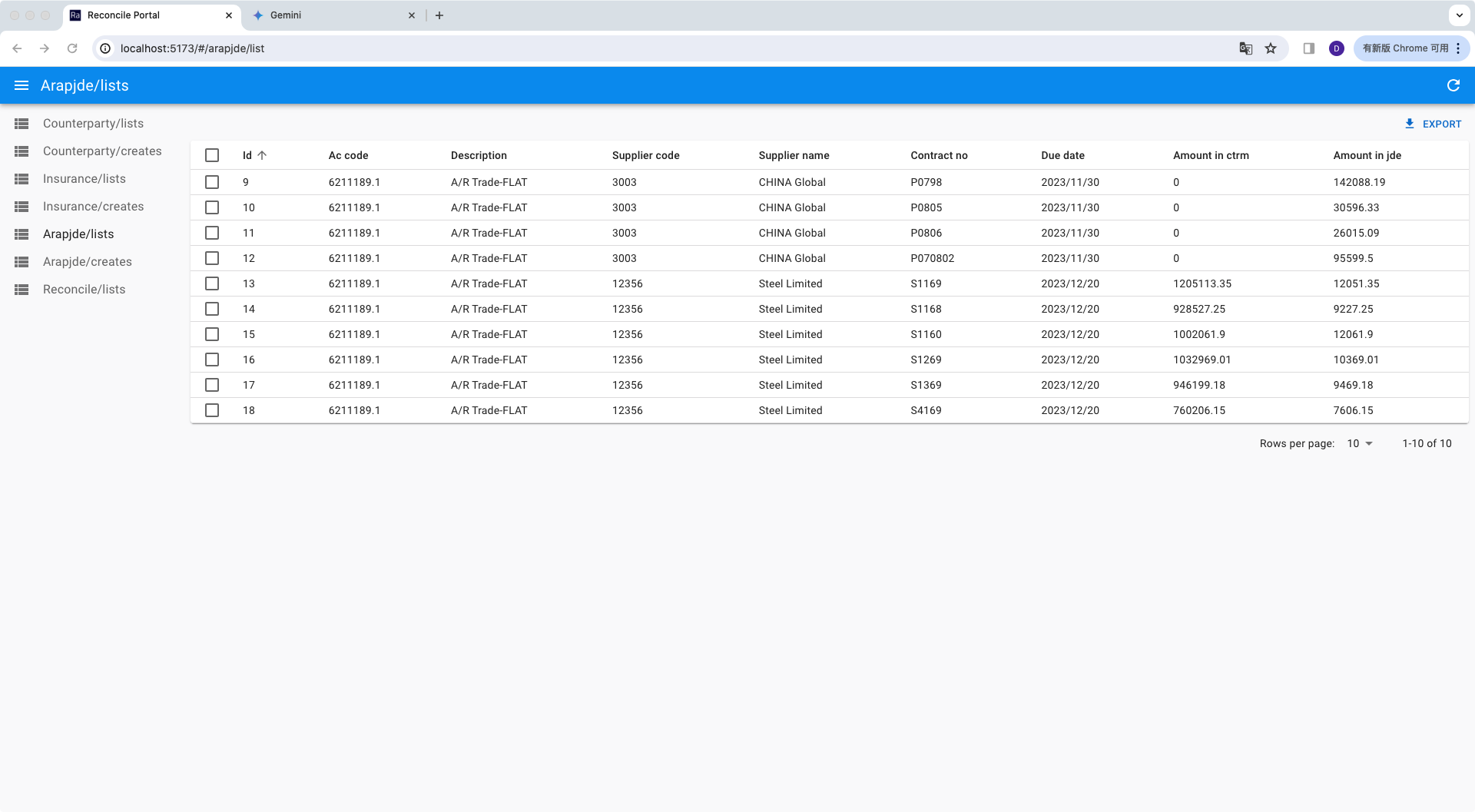
* Client UI

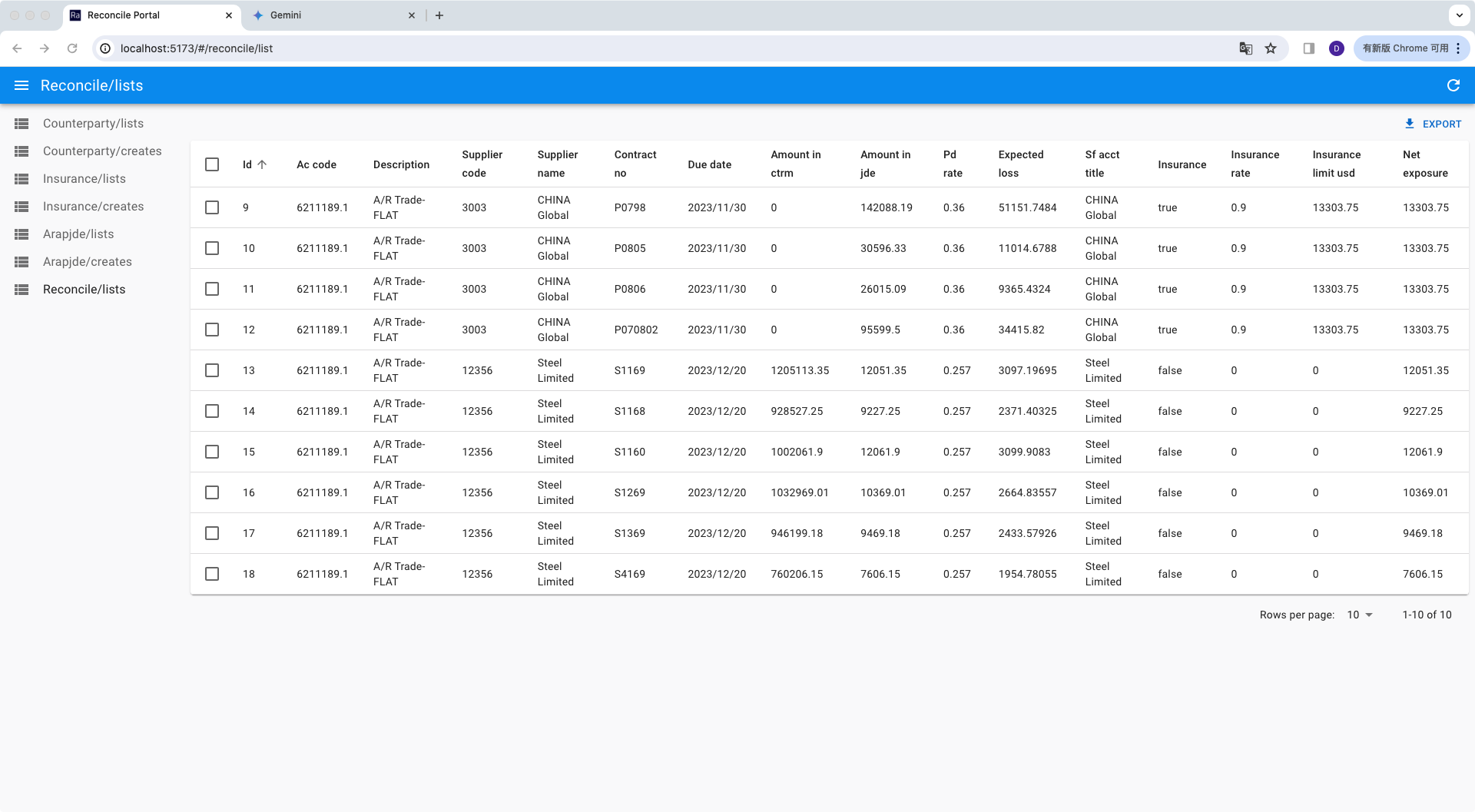


5. **Demo**

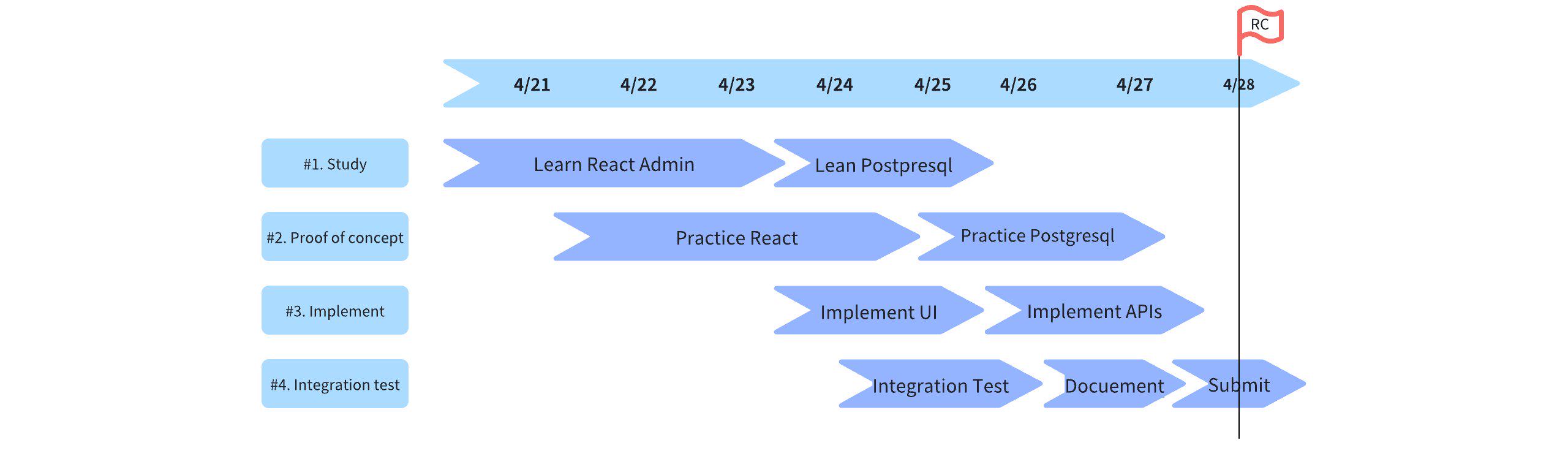








6. **Plan**



7. **To-dos**

* UI Enhancement:
* Further evaluate whether adjustments are necessary to the template and style.
* Validate Assumptions:
* If volumes increase tenfold, adjustments will be necessary. For instance, we could implement a new service to pre-load reconciliation results. This would allow the UI to fetch data directly from the database, thereby reducing latency.