Trello Real-time WebSockets + API Frontend Assignment

Objective

Evaluate interns' competency in building a Trello-like frontend with real-time synchronization. They must use Trello's public REST API as the backend data store and implement real-time updates using a WebSocket server (their own) that integrates with Trello webhooks or push events.

High-level Requirements

• Build a single-page React frontend (create-react-app, Vite, Next.js — any modern stack) that replicates a Trello-style board (boards → lists → cards).
• Build a lightweight backend (Node.js/Express recommended) that: (a) Exposes four testable HTTP APIs (see below); (b) Listens for Trello webhooks and broadcasts real-time events to connected clients over WebSockets (Socket.IO or ws).
• Every frontend action must be reflected in Trello via the Trello REST API (the Trello account chosen by the candidate).
• Undo/optimistic UI and drag & drop are optional bonuses.

Four Required Backend API Endpoints (must be implemented and tested)

```
1) Add new task (create card)
   - HTTP: POST /api/tasks
   - Payload example:
     { "boardId": "<board id>", "listId": "<list id>", "name": "Task title", "desc": "Task details"
   - Action: Create a Trello card via POST https://api.trello.com/1/cards

2) Update existing task (update card)
   - HTTP: PUT /api/tasks/:cardId
   - Payload example:
     { "name": "New title", "desc": "Updated description", "idList": "<new list id>" }
   - Action: Update Trello card via PUT https://api.trello.com/1/cards/{id}

3) Delete task (delete card)
   - HTTP: DELETE /api/tasks/:cardId
   - Action: Delete or close the Trello card via Trello API. (Candidates should document whether th

4) Make new board
   - HTTP: POST /api/boards
   - Payload example:
     { "name": "Hiring Test Board", "defaultLists": true }
   - Action: Create Trello board via POST https://api.trello.com/1/boards
```

Real-time (WebSockets) Requirements

Your WebSocket server must broadcast changes to all connected clients in real time. The canonical approach for this assignment is: 1) Candidate registers a Trello webhook (POST /1/webhooks) that points to a publicly reachable callback URL on their backend. 2) The backend receives webhook events from Trello, translates them into a normalized event shape, and emits them to clients over the WebSocket channel. 3) Clients subscribe to the WebSocket channel and apply incoming events to update the UI live. Note: Trello webhooks are the recommended mechanism — polling is allowed as a fallback but will be penalized in grading.

Trello Authentication (API key & token)

Candidates must create their own Trello API key and token (do NOT share private keys). Use the API key and token on every Trello REST call as query parameters: key=YOUR_KEY&token;=YOUR_TOKEN. Example placeholder values (do not use in production): API_KEY = {YOUR_API_KEY} API_TOKEN = {YOUR_API_TOKEN} See Trello's official developer docs for how to obtain them (included in references).

Example Trello API calls (placeholders shown)

```
Create a card (POST):
curl -X POST "https://api.trello.com/1/cards?key={API_KEY}&token={API_TOKEN}&idList={LIST_ID}&name=

Update a card (PUT):
curl -X PUT "https://api.trello.com/1/cards/{CARD_ID}?key={API_KEY}&token={API_TOKEN}&name=Updated+

Delete (close) a card (PUT to set closed=true) or DELETE if candidate prefers:
curl -X PUT "https://api.trello.com/1/cards/{CARD_ID}?key={API_KEY}&token={API_TOKEN}&closed=true"

Create board (POST):
curl -X POST "https://api.trello.com/1/boards/?key={API_KEY}&token={API_TOKEN}&name=TestBoard"
```

Deliverables

• Link to a GitHub repo with frontend and backend separated (two folders).
• README with setup instructions, how to obtain Trello key/token, and how to register the webhook (include the webhook creation command).
• Deployed backend (ngrok or public hosting) so Trello can reach the webhook callback URL.
• Short demo video (3–5 min) showing real-time sync across two browser windows and the four APIs being hit.
• A postman collection or curl examples to exercise the four required endpoints.

Evaluation Rubric (100 points)

• Correctness of required APIs: 30 points (each API must call Trello correctly).
• Real-time behavior & WebSocket design: 30 points (webhook handling, broadcasting, low latency).
• Frontend quality & UX: 15 points (drag & drop, responsiveness, CSS quality).
• Code quality & separation (frontend vs backend): 10 points.
• Documentation & demo: 10 points.
• Bonus (optimistic UI, offline support, tests): up to 5 points.

Automated / Manual Test Cases (suggested)

• Create Board: POST /api/boards -> assert board exists in Trello (check via GET /1/boards/{id}).
• Add Task: POST /api/tasks -> assert card appears in Trello list and is broadcast to connected clients.
• Update Task: PUT /api/tasks/:cardId -> assert fields change in Trello and clients receive update event.
• Delete Task: DELETE /api/tasks/:cardId -> assert card closed/deleted and clients receive delete event.
• Simulate two browser clients and show real-time updates for each operation.

Notes, Hints & Constraints

• Trello rate limits exist — avoid extremely frequent polling. Prefer webhooks.
• Webhook callback URL must be publicly reachable (ngrok is acceptable for testing).
• For security, do not commit API keys or tokens to the repository. Use environment variables.
• Candidates should handle errors from Trello gracefully and surface user-friendly messages.

References (official Trello docs)

• Trello API overview & how to get API key/token: https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/
• Cards API reference: https://developer.atlassian.com/cloud/trello/rest/api-group-cards/
• Boards API reference: https://developer.atlassian.com/cloud/trello/rest/api-group-boards/
• Webhooks: https://developer.atlassian.com/cloud/trello/guides/rest-api/webhooks/ and https://developer.atlassian.com/cloud/trello/rest/api-group-webhooks/

How to grade quickly (suggested)

• Clone repo, set env keys, run backend, register webhook with provided script, open two browser windows and perform ops.
• Run the included Postman collection to hit the four APIs and observe real-time updates.