# Deck Dropout Challenge

## Abstract

School is fun, but the fastest way to learn is to take the leap and get out there. The theory of automation is easy; the reality of modern, hostile websites is not.

In this challenge, we're testing your ability to build an automated agent that is not just functional, but resilient under real-world conditions: unclear user flows, hidden friction, and aggressive anti-bot systems.

We are providing a **Mock University Portal**. Your mission is simple to state and hard to execute reliably:
**build an agent that successfully drops out.**

## The Objective

Your agent must autonomously reach a final state where the user is officially **Withdrawn** from the university.

Account creation **may be done manually**.
From authentication onward, **everything must be automated**.

You will **not** be told:

- Where the dropout flow lives
- What steps are required
- Which paths are "correct"

You must explore, understand, and automate the process yourself.

At a minimum, your agent must:

- **Authenticate automatically** with an existing account
- Navigate the site's internal structure
- Discover and initiate the withdrawal process

- Handle confirmations, blockers, or modals
- Verify the final status is **Withdrawn**

## The Real Challenge

This is not a “find the right button” challenge.

The flow may appear manageable to a human, but once you introduce automation, things get progressively harder at every step.

The **primary difficulty** is:

- Embedded **anti-bot and anti-automation mechanisms**
- Friction that only appears under scripted interaction
- Increasing resistance as the agent progresses

UI changes and DOM instability still exist, but they are **not the main obstacle**.

## AI Is Encouraged

We actively encourage the use of AI.

LLM-assisted reasoning, DOM interpretation, vision-based approaches, and heuristic navigation are all fair game. We care about results and engineering quality, not dogma.

## Technical Constraints & Rules of Engagement

### 🚫 Banned (Easy Buttons)

You may **not** use pre-built, fully managed browsing or scraping platforms that automatically handle resilience or anti-bot behavior.

Examples (non-exhaustive):

- Atlas
- Browserbase

- Browseruse
- Deck
- Similar managed services

### ✅ Allowed (Raw Materials)

Open-source libraries and SDKs running locally or on your own infrastructure are allowed.

Examples:

- Playwright
- Puppeteer
- Selenium
- Beautiful Soup
- LangChain (logic only)
- OpenAI SDK (vision / parsing / reasoning)

### The “One SDK” Rule

Rely primarily on **one major automation framework**. Do not chain multiple browser engines together.

Vibe coding is allowed. It just has to be good.

# Evaluation

There is **no Phase 1 evaluation**.

All submissions are evaluated **once**, at the end of the challenge, after defenses and changes are live.

Judging is based on a 100-point scale:

### 1. Reliability & Resilience (50 points)

- Does the bot complete the full dropout flow?
- Does it survive friction, UI changes, and anti-bot defenses without manual intervention?

### 2. Execution Speed (25 points)

- Time measured from automated login to confirmed **Withdrawn** state
- Bots taking longer than ~3 minutes will be penalized

### 3. Code Hygiene & Engineering (25 points)

- Clean, modular, readable code
- Thoughtful resilience strategy
- Graceful handling of timeouts, retries, and failures
- Avoidance of brittle selectors and hardcoded paths

## Submission Requirements

- **GitHub Repository** (public or shared with judges)
- **Video Demo** showing a successful automated run
- **README** with setup and execution instructions

## The Prize

For those bold enough to drop out:

- **$3,000 Cash (split per team)**
- **Interviews for a Deck Summer Internship**
- **4× Deck Skateboards (limited edition custom graphics)**

## ⚠️ Important Note

**All cash prizes are awarded in USD (not CAD).**

---