```
"""
claude_quota.py — Claude Code / Claude.ai Quota Monitor

Drop into src/infrastructure/ in the Timmy Time Dashboard repo.
Provides real-time quota visibility and metabolic protocol decisions.

Usage:
    from infrastructure.claude_quota import QuotaMonitor

    monitor = QuotaMonitor()
    status = monitor.check()

    print(status.five_hour_pct)       # 0.42
    print(status.five_hour_resets_in) # "2h 15m"
    print(status.seven_day_pct)       # 0.29
    print(status.recommended_tier)    # "cloud" | "local-14b" | "local-8b"

    # Metabolic protocol: auto-select model based on quota
    model = monitor.select_model(task_complexity="high")
    # Returns "claude-sonnet-4-6" if quota allows, else "qwen3:14b"
"""

import json
import subprocess
import logging
from dataclasses import dataclass
from datetime import datetime, timezone
from typing import Optional
from enum import Enum

logger = logging.getLogger(__name__)


class MetabolicTier(str, Enum):
    """The three-tier metabolic protocol from the Timmy Time architecture."""
    BURST = "burst"       # Cloud API (Claude/Groq) — expensive, best quality
    ACTIVE = "active"     # Local 14B (Qwen3-14B) — free, good quality
    RESTING = "resting"   # Local 8B (Qwen3-8B) — free, fast, adequate


@dataclass
class QuotaStatus:
    """Current Claude quota state."""
    five_hour_utilization: float      # 0.0 to 1.0
```

```
    five_hour_resets_at: Optional[str]
    seven_day_utilization: float      # 0.0 to 1.0
    seven_day_resets_at: Optional[str]
    raw_response: dict
    fetched_at: datetime

    @property
    def five_hour_pct(self) -> int:
        return int(self.five_hour_utilization * 100)

    @property
    def seven_day_pct(self) -> int:
        return int(self.seven_day_utilization * 100)

    @property
    def five_hour_resets_in(self) -> str:
        return _time_remaining(self.five_hour_resets_at)

    @property
    def seven_day_resets_in(self) -> str:
        return _time_remaining(self.seven_day_resets_at)

    @property
    def recommended_tier(self) -> MetabolicTier:
        """Metabolic protocol: determine which inference tier to use."""
        # If weekly quota is critical, go full local
        if self.seven_day_utilization >= 0.80:
            return MetabolicTier.RESTING

        # If 5-hour window is critical, use local for routine
        if self.five_hour_utilization >= 0.80:
            return MetabolicTier.ACTIVE

        # If 5-hour window is past half, be selective about cloud
        if self.five_hour_utilization >= 0.50:
            return MetabolicTier.ACTIVE

        # Quota healthy — cloud available for high-value tasks
        return MetabolicTier.BURST

    def summary(self) -> str:
        """Human-readable status string."""
        return (
            f"5h: {self.five_hour_pct}% (resets {self.five_hour_resets_in}) | "
            f"7d: {self.seven_day_pct}% (resets {self.seven_day_resets_in}) | "
            f"tier: {self.recommended_tier.value}"
        )
```

```
class QuotaMonitor:
    """
    Monitors Claude Code / Claude.ai quota via the internal OAuth API.

    The token is read from macOS Keychain where Claude Code stores it.
    Falls back gracefully if credentials aren't available (e.g., on Linux VPS).
    """

    API_URL = "https://api.anthropic.com/api/oauth/usage"
    KEYCHAIN_SERVICE = "Claude Code-credentials"
    USER_AGENT = "claude-code/2.0.32"

    def __init__(self):
        self._token: Optional[str] = None
        self._last_status: Optional[QuotaStatus] = None
        self._cache_seconds = 30  # Don't hammer the API

    def _get_token(self) -> Optional[str]:
        """Extract OAuth token from macOS Keychain."""
        if self._token:
            return self._token

        try:
            result = subprocess.run(
                ["security", "find-generic-password", "-s", self.KEYCHAIN_SERVICE
                capture_output=True, text=True, timeout=5
            )
            if result.returncode != 0:
                logger.warning("Claude Code credentials not found in Keychain")
                return None

            creds = json.loads(result.stdout.strip())
            oauth = creds.get("claudeAiOauth", creds)
            self._token = oauth.get("accessToken")
            return self._token

        except (json.JSONDecodeError, KeyError, FileNotFoundError, subprocess.Tim
            logger.warning(f"Could not read Claude Code credentials: {e}")
            return None

    def check(self, force: bool = False) -> Optional[QuotaStatus]:
        """
        Fetch current quota status.

        Returns None if credentials aren't available (graceful degradation).
```

```
    Caches results for 30 seconds to avoid rate limiting the quota API itself
    """
    # Return cached if fresh
    if not force and self._last_status:
        age = (datetime.now(timezone.utc) - self._last_status.fetched_at).tot
        if age < self._cache_seconds:
            return self._last_status

    token = self._get_token()
    if not token:
        return None

    try:
        import urllib.request

        req = urllib.request.Request(
            self.API_URL,
            headers={
                "Accept": "application/json",
                "Content-Type": "application/json",
                "User-Agent": self.USER_AGENT,
                "Authorization": f"Bearer {token}",
                "anthropic-beta": "oauth-2025-04-20",
            },
        )

        with urllib.request.urlopen(req, timeout=10) as resp:
            data = json.loads(resp.read().decode())

        five_hour = data.get("five_hour") or {}
        seven_day = data.get("seven_day") or {}

        self._last_status = QuotaStatus(
            five_hour_utilization=five_hour.get("utilization", 0.0),
            five_hour_resets_at=five_hour.get("resets_at"),
            seven_day_utilization=seven_day.get("utilization", 0.0),
            seven_day_resets_at=seven_day.get("resets_at"),
            raw_response=data,
            fetched_at=datetime.now(timezone.utc),
        )
        return self._last_status

    except Exception as e:
        logger.warning(f"Failed to fetch quota: {e}")
        return self._last_status  # Return stale data if available

def select_model(self, task_complexity: str = "medium") -> str:
```

```
        """
        Metabolic protocol: select the right model based on quota + task complexi

        Returns an Ollama model tag or "claude-sonnet-4-6" for cloud.

        task_complexity: "low" | "medium" | "high"
        """
        status = self.check()

        # No quota info available — assume local only (sovereign default)
        if status is None:
            return "qwen3:14b" if task_complexity == "high" else "qwen3:8b"

        tier = status.recommended_tier

        if tier == MetabolicTier.BURST and task_complexity == "high":
            return "claude-sonnet-4-6"  # Cloud — best quality
        elif tier == MetabolicTier.BURST and task_complexity == "medium":
            return "qwen3:14b"  # Save cloud for truly hard tasks
        elif tier == MetabolicTier.ACTIVE:
            return "qwen3:14b"  # Local 14B — good enough
        else:  # RESTING
            return "qwen3:8b"  # Local 8B — conserve everything

    def should_use_cloud(self, task_value: str = "normal") -> bool:
        """
        Simple yes/no: should this task use cloud API?

        task_value: "critical" | "high" | "normal" | "routine"
        """
        status = self.check()
        if status is None:
            return False  # No credentials = local only

        if task_value == "critical":
            return status.seven_day_utilization < 0.95  # Almost always yes
        elif task_value == "high":
            return status.five_hour_utilization < 0.60
        elif task_value == "normal":
            return status.five_hour_utilization < 0.30
        else:  # routine
            return False  # Never waste cloud on routine


def _time_remaining(reset_at: Optional[str]) -> str:
    """Format time until reset as human-readable string."""
    if not reset_at or reset_at == "null":
```

```
        return "unknown"

    try:
        reset = datetime.fromisoformat(reset_at.replace("Z", "+00:00"))
        now = datetime.now(timezone.utc)
        diff = reset - now

        if diff.total_seconds() <= 0:
            return "resetting now"

        hours = int(diff.total_seconds() // 3600)
        mins = int((diff.total_seconds() % 3600) // 60)

        if hours > 0:
            return f"{hours}h {mins}m"
        return f"{mins}m"

    except (ValueError, TypeError):
        return "unknown"


# ── CLI entrypoint ──
if __name__ == "__main__":
    import sys

    monitor = QuotaMonitor()
    status = monitor.check()

    if status is None:
        print("Could not fetch quota. Are Claude Code credentials in Keychain?")
        sys.exit(1)

    print()
    print("  CLAUDE QUOTA STATUS")
    print("  " + "─" * 40)
    print(f"  5-hour window:  {status.five_hour_pct:3d}%  (resets in {status.five
    print(f"  7-day window:   {status.seven_day_pct:3d}%  (resets in {status.seve
    print(f"  Recommended:    {status.recommended_tier.value}")
    print(f"  Model select:   {monitor.select_model('high')}")
    print()

    if "--json" in sys.argv:
        print(json.dumps(status.raw_response, indent=2))
```