TaskTuner API Specification

This document provides complete API documentation for TaskTuner's REST endpoints.

Base URL

http://localhost:5000

Authentication

TaskTuner currently uses session-based authentication. No API keys or tokens are required for local development.

Response Format

All API responses return JSON with appropriate HTTP status codes.

Success Responses

- 200 OK - Request successful
- 201 Created - Resource created successfully

Error Responses

- 400 Bad Request - Invalid request data
- 404 Not Found - Resource not found
- 500 Internal Server Error - Server error

Tasks API

Base path: /tasks

Get All Tasks

Fetches all tasks from Trello and returns them with current effort estimates.

Endpoint:

GET /tasks

Parameters: None

Response:

```
[
  {
    "id": "trello_card_id_123",
    "name": "Design user interface mockups",
    "effort": 4
  },
  {
    "id": "trello_card_id_456",
    "name": "Implement user authentication",
    "effort": 6
  }
]
```


Response Fields:

- id (string) - Unique Trello card ID
- name (string) - Task name from Trello
- effort (number) - Estimated hours to complete (0 if not set)

Behavior:

- Clears local task cache
- Fetches fresh data from Trello API
- Falls back to task_data.json if Trello unavailable
- Preserves existing effort estimates for known tasks

Example Request:

```
curl -X GET http://localhost:5000/tasks
```

Update Task Effort

Updates the effort estimate for one or more tasks.

Endpoint:

POST /tasks/effort

Content-Type: application/json

Single Task Request:

```
{
  "taskId": "trello_card_id_123",
  "hours": 3
}
```

Multiple Tasks Request:

```
[
  {
    "taskId": "trello_card_id_123",
    "hours": 3
  },
  {
    "taskId": "trello_card_id_456",
    "hours": 5
  }
]
```

Request Fields:

- taskId (string, required) - Trello card ID
- hours (number, required) - Effort estimate in hours (can be 0)

Response:

```
[
  {
    "id": "trello_card_id_123",
```

```
    "name": "Design user interface mockups",
    "effort": 3
  }
]
```

Error Cases:

- Task ID not found: Task is skipped, no error thrown
- Invalid hours value: Task is skipped
- Empty request body: Returns empty array

Example Requests:

```
# Single task
curl -X POST http://localhost:5000/tasks/effort \
  -H "Content-Type: application/json" \
  -d '{"taskId": "123", "hours": 4}'

# Multiple tasks
curl -X POST http://localhost:5000/tasks/effort \
  -H "Content-Type: application/json" \
  -d '[{"taskId": "123", "hours": 4}, {"taskId": "456", "hours": 2}]'
```

Schedule API

Base path: /schedule

Generate Schedule

Creates an optimized daily schedule based on task effort estimates and selected algorithm.

Endpoint:

GET /schedule

Parameters:

- startDate (string, optional) - Start date in YYYY-MM-DD format
    - Default: Current date
    - Example: 2025-01-15
- alg (string, optional) - Scheduling algorithm
    - balanced (default) - Maintains task order
    - largest - Largest effort tasks first
    - smallest - Smallest effort tasks first

Response:


```
[
 {
  "date": "2025-01-15",
  "tasks": [
   {
    "id": "trello_card_id_123",
    "name": "Design user interface mockups",
    "effort": 4
   },
   {
    "id": "trello_card_id_456",
    "name": "Write unit tests",
    "effort": 3
   }
  ]
 },
 {
  "date": "2025-01-16",
```

```
  "tasks": [
    {
      "id": "trello_card_id_789",
      "name": "Implement payment processing",
      "effort": 8
    }
  ]
 }
]
```


Response Structure:

- Array of day objects
- Each day contains:
  - date (string) - Date in YYYY-MM-DD format
  - tasks (array) - Tasks scheduled for that day
- Task objects contain same fields as Tasks API

Scheduling Rules:

- Daily hour limit controlled by DAILY_HOUR_LIMIT config (default: 8)
- Tasks exceeding daily limit are split across multiple days
- Tasks with 0 effort are excluded from scheduling
- Empty days are still included in response

Example Requests:

```
# Default schedule (today, balanced algorithm)
curl -X GET http://localhost:5000/schedule

# Custom start date
curl -X GET "http://localhost:5000/schedule?startDate=2025-02-01"
```

Different algorithm

curl -X GET "http://localhost:5000/schedule?alg=largest"

Combined parameters

curl -X GET "http://localhost:5000/schedule?startDate=2025-02-01&alg=smallest"

Algorithm Details

Balanced Algorithm

- **Purpose**: Maintains original task order while fitting within daily limits
- **Logic**:
    1. Processes tasks in their current order
    2. Fills each day up to the hour limit
    3. Splits large tasks across multiple days if needed
- **Best for**: General use, preserving task priorities

Largest First Algorithm

- **Purpose**: Tackles high-effort tasks early
- **Logic**:
    1. Sorts all tasks by effort in descending order
    2. Schedules highest effort tasks first
    3. Fills remaining time with smaller tasks
- **Best for**: Getting big items done early, reducing cognitive load

Smallest First Algorithm

- **Purpose**: Builds momentum with quick wins
- **Logic**:
    1. Sorts all tasks by effort in ascending order

2. Schedules lowest effort tasks first
3. Builds up to larger tasks

- **Best for**: Building momentum, clearing backlog quickly

Data Persistence

Task Storage

- Tasks are stored in tasks_store.json
- File is automatically created and updated
- Thread-safe writes using file locking
- Preserves effort estimates between sessions

Storage Format

```
[
 {
  "id": "task_id",
  "name": "Task name",
  "effort": 5
 }
]
```

Error Handling

Trello API Errors

When Trello API is unavailable:

1. Console warning is logged
2. Fallback to task_data.json is attempted
3. If fallback fails, empty task list is returned
4. No HTTP error is thrown to the client

Invalid Task References

- Unknown task IDs in effort updates are silently ignored
- Only valid tasks are updated and returned
- No error messages for invalid references

Scheduling Errors

- Tasks with invalid effort values (negative, non-numeric) are treated as 0
- Empty task lists result in empty schedules
- Invalid date formats default to current date

Rate Limiting

Currently no rate limiting is implemented. For production use, consider:

- Request rate limiting per IP
- Trello API call throttling
- Concurrent request limits

Example Integration

JavaScript Frontend

```
// Fetch all tasks
const tasks = await fetch('/tasks').then(r => r.json());


// Update task effort
await fetch('/tasks/effort', {
  method: 'POST',
  headers: {'Content-Type': 'application/json'},
  body: JSON.stringify({taskId: 'abc123', hours: 5})
});
```

```
// Generate schedule
const schedule = await fetch('/schedule?alg=largest').then(r => r.json());
```

Python Client

```
import requests

# Fetch tasks
response = requests.get('http://localhost:5000/tasks')
tasks = response.json()

# Update effort
data = {'taskId': 'abc123', 'hours': 5}
response = requests.post('http://localhost:5000/tasks/effort', json=data)

# Generate schedule
params = {'startDate': '2025-02-01', 'alg': 'balanced'}
response = requests.get('http://localhost:5000/schedule', params=params)
schedule = response.json()
```

Future API Enhancements

Planned Endpoints

- PUT /tasks/{id} - Update individual task
- DELETE /tasks/{id} - Remove task from scheduling
- POST /schedule/save - Save custom schedule modifications
- GET /schedule/stats - Get scheduling statistics and analytics

Authentication

Future versions may include:

- API key authentication
- User-based task isolation
- Role-based access control

Webhooks

Potential webhook endpoints for real-time updates:

- POST /webhooks/trello - Receive Trello card updates
- POST /webhooks/schedule - Receive schedule change notifications