

SOFTWARE ENGINEERING

(BSCS3001)

Submission by Team 15

Yash Agrawal (22f3003228)

Vidhan Mertiya (22f1001289)

Neelotpal Dutta (21f3002397)

Aviral Kaintura (21f2000172)

Mihir R Shreshti (22f2001525)

Samadhan Kashinath Patil (21f1006852)



IITM BS Degree Program

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS, CHENNAI

Tamil Nadu, India – 600036

Contents

1	Introduction	4
2	Test Environment Setup	4
2.1	Authentication Fixtures	4
2.1.1	Parent Setup Fixture	4
2.1.2	Teacher Setup Fixture	5
2.1.3	Child Setup Fixture	6
3	Authentication Endpoints	7
3.1	User Registration	7
3.1.1	Valid Parent Registration	7
3.1.2	Invalid Registration - Duplicate Email	8
3.2	User Login	9
3.2.1	Valid Login	9
3.2.2	Invalid Login	10
4	User Management Endpoints	11
4.1	Get Current User Profile	11
4.1.1	Authenticated User Profile	11
4.2	Update User Profile	12
4.2.1	Valid Profile Update	12
5	Goals Management Endpoints	13
5.1	Get User Goals	13
5.1.1	Get Child Goals	13
5.2	Create Goal	15
5.2.1	Valid Goal Creation	15
5.3	Contribute to Goal	16
5.3.1	Valid Goal Contribution	16
6	Dashboard Endpoints	18
6.1	Get Child Dashboard	18
6.1.1	Child Dashboard Data	18
6.2	Get Parent Dashboard	20
6.2.1	Parent Dashboard Data	20
7	Task Management Endpoints	22
7.1	Get User Tasks	22
7.1.1	Get Child Tasks	23
7.2	Create Task	24
7.2.1	Valid Task Creation	24
8	Shop System Endpoints	25
8.1	Get Shop Items	25
8.1.1	Get Shop Items	25
8.2	Purchase Item	26

8.2.1	Valid Purchase	26
9	Redemption System Endpoints	28
9.1	Get Redemption Requests	28
9.1.1	Get Child Redemption Requests	28
9.2	Create Redemption Request	29
9.2.1	Valid Redemption Request	29
10	Health Check Endpoints	30
10.1	Health Check	30
10.1.1	Health Check	31
10.2	API Status	31
10.2.1	API Status	31
11	Error Handling Test Cases	32
11.1	Unauthorized Access	32
11.2	Invalid Token	33
11.3	Resource Not Found	33
12	Conclusion	34

MILESTONE 5

1 Introduction

This document outlines comprehensive test cases for the CoinCraft FastAPI backend application. The application supports multiple user roles (Parent, Teacher, Younger Child, Older Child) with role-specific functionalities including financial management, learning modules, task management, and redemption systems.

2 Test Environment Setup

2.1 Authentication Fixtures

These fixtures establish test clients and handle user authentication for different roles.

2.1.1 Parent Setup Fixture

Fixture Name: parent_setup_data

Scope: session

Description: Logs in as a Parent user and retrieves authentication token.

Test Case:

1. Login as Parent User

2. Passed Inputs:

```
1 {  
2   "email": "parent@test.com",  
3   "password": "Parent@123"  
4 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {  
2   "access_token": "valid_jwt_token",  
3   "token_type": "bearer",  
4   "user": {  
5     "id": "uuid",  
6     "email": "parent@test.com",  
7     "name": "Test Parent",  
8     "role": "parent",  
9     "avatar_url": null,  
10    "created_at": "2025-01-01T00:00:00Z",  
11    "updated_at": "2025-01-01T00:00:00Z",  
12    "is_active": true,  
13    "is_superuser": false,  
14    "is_verified": false  
15  }
```

```
16 }
```

4. Result: Passed

Pytest Code:

```
1 @pytest.fixture(scope='session')
2 def parent_setup_data(client):
3     response = client.post('/api/auth/register', json={
4         'email': 'parent@test.com',
5         'password': 'Parent@123',
6         'name': 'Test_Parent',
7         'role': 'parent'
8     })
9     assert response.status_code == 200
10    data = response.json()
11    token = data['access_token']
12    yield token, client
```

2.1.2 Teacher Setup Fixture

Fixture Name: teacher_setup_data

Scope: session

Description: Logs in as a Teacher user and retrieves authentication token.

Test Case:

1. Login as Teacher User

2. Passed Inputs:

```
1 {
2     "email": "teacher@test.com",
3     "password": "Teacher@123"
4 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2     "access_token": "valid_jwt_token",
3     "token_type": "bearer",
4     "user": {
5         "id": "uuid",
6         "email": "teacher@test.com",
7         "name": "Test Teacher",
8         "role": "teacher",
9         "avatar_url": null,
10        "created_at": "2025-01-01T00:00:00Z",
11        "updated_at": "2025-01-01T00:00:00Z",
12        "is_active": true,
```

```
13     "is_superuser": false,  
14     "is_verified": false  
15 }  
16 }
```

4. **Result:** Passed

2.1.3 Child Setup Fixture

Fixture Name: child_setup_data

Scope: session

Description: Logs in as a Younger Child user and retrieves authentication token.

Test Case:

1. Login as Child User

2. Passed Inputs:

```
1 {  
2   "email": "child@test.com",  
3   "password": "Child@123"  
4 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {  
2   "access_token": "valid_jwt_token",  
3   "token_type": "bearer",  
4   "user": {  
5     "id": "uuid",  
6     "email": "child@test.com",  
7     "name": "Test Child",  
8     "role": "younger_child",  
9     "avatar_url": null,  
10    "created_at": "2025-01-01T00:00:00Z",  
11    "updated_at": "2025-01-01T00:00:00Z",  
12    "is_active": true,  
13    "is_superuser": false,  
14    "is_verified": false  
15  }  
16 }
```

4. **Result:** Passed

3 Authentication Endpoints

3.1 User Registration

Endpoint: POST /api/auth/register

Description: Register a new user and immediately log them in.

Test Cases:

3.1.1 Valid Parent Registration

Test Case:

1. Valid Parent Registration

2. Passed Inputs:

```
1 {
2   "email": "newparent@test.com",
3   "password": "NewParent@123",
4   "name": "New Parent",
5   "role": "parent"
6 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2   "access_token": "valid_jwt_token",
3   "token_type": "bearer",
4   "user": {
5     "id": "uuid",
6     "email": "newparent@test.com",
7     "name": "New Parent",
8     "role": "parent",
9     "avatar_url": null,
10    "created_at": "2025-01-01T00:00:00Z",
11    "updated_at": "2025-01-01T00:00:00Z",
12    "is_active": true,
13    "is_superuser": false,
14    "is_verified": false
15  }
16 }
```

4. Result: Passed

Pytest Code:

```
1 def test_valid_parent_registration(client):
2     """Test valid parent registration."""
3     response = client.post('/api/auth/register', json={
4         'email': 'newparent@test.com',
```

```
5         'password': 'NewParent@123',
6         'name': 'New_Parent',
7         'role': 'parent'
8     })
9
10    assert response.status_code == 200
11    data = response.json()
12    assert 'access_token' in data
13    assert data['token_type'] == 'bearer'
14    assert data['user']['email'] == 'newparent@test.com'
15    assert data['user']['role'] == 'parent'
16    assert data['user']['is_active'] == True
```

3.1.2 Invalid Registration - Duplicate Email

Test Case:

1. Duplicate Email Registration

2. Passed Inputs:

```
1  {
2    "email": "existing@test.com",
3    "password": "Test@123",
4    "name": "Test User",
5    "role": "parent"
6  }
```

3. Expected Output:

- HTTP Status Code: 400
- JSON Response:

```
1  {
2    "detail": "User with this email already exists"
3  }
```

4. Result: Passed

Pytest Code:

```
1 def test_duplicate_email_registration(client):
2     """Test registration with duplicate email."""
3     # First registration
4     client.post('/api/auth/register', json={
5         'email': 'existing@test.com',
6         'password': 'Test@123',
7         'name': 'Test_User',
8         'role': 'parent'
9     })
10
11     # Second registration with same email
```



```
12 response = client.post('/api/auth/register', json={
13     'email': 'existing@test.com',
14     'password': 'Test@123',
15     'name': 'Test_User',
16     'role': 'parent'
17 })
18
19 assert response.status_code == 400
20 data = response.json()
21 assert data['detail'] == 'User_with_this_email_already_exists'
```

3.2 User Login

Endpoint: POST /api/auth/jwt/login

Description: Login with email and password using OAuth2 form data.

Test Cases:

3.2.1 Valid Login

Test Case:

1. Valid User Login

2. Passed Inputs:

```
1 {
2     "username": "parent@test.com",
3     "password": "Parent@123"
4 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2     "access_token": "valid_jwt_token",
3     "token_type": "bearer"
4 }
```

4. Result: Passed

Pytest Code:

```
1 def test_valid_login(client):
2     """Test valid user login."""
3     # First register a user
4     client.post('/api/auth/register', json={
5         'email': 'parent@test.com',
6         'password': 'Parent@123',
7         'name': 'Test_Parent',
```

```
8         'role': 'parent'
9     })
10
11     # Login with the registered user
12     response = client.post('/api/auth/jwt/login', data={
13         'username': 'parent@test.com',
14         'password': 'Parent@123'
15     }, headers={'Content-Type': 'application/x-www-form-urlencoded'})
16
17     assert response.status_code == 200
18     data = response.json()
19     assert 'access_token' in data
20     assert data['token_type'] == 'bearer'
```

3.2.2 Invalid Login

Test Case:

1. Invalid Credentials

2. Passed Inputs:

```
1  {
2      "username": "wrong@test.com",
3      "password": "WrongPassword"
4  }
```

3. Expected Output:

- HTTP Status Code: 401
- JSON Response:

```
1  {
2      "detail": "LOGIN_BAD_CREDENTIALS"
3  }
```

4. Result: Passed

Pytest Code:

```
1 def test_invalid_login(client):
2     """Test login with invalid credentials."""
3     response = client.post('/api/auth/jwt/login', data={
4         'username': 'wrong@test.com',
5         'password': 'WrongPassword'
6     }, headers={'Content-Type': 'application/x-www-form-urlencoded'})
7
8     assert response.status_code == 401
9     data = response.json()
10    assert data['detail'] == 'LOGIN_BAD_CREDENTIALS'
```

4 User Management Endpoints

4.1 Get Current User Profile

Endpoint: GET /api/users/me

Description: Get current user's profile information.

Test Cases:

4.1.1 Authenticated User Profile

Test Case:

1. **Get Current User Profile**

2. **Headers:**

```
1 {  
2   "Authorization": "Bearer valid_jwt_token"  
3 }
```

3. **Expected Output:**

- HTTP Status Code: 200
- JSON Response:

```
1 {  
2   "id": "uuid",  
3   "email": "parent@test.com",  
4   "name": "Test Parent",  
5   "role": "parent",  
6   "avatar_url": null,  
7   "created_at": "2025-01-01T00:00:00Z",  
8   "updated_at": "2025-01-01T00:00:00Z",  
9   "is_active": true,  
10  "is_superuser": false,  
11  "is_verified": false  
12 }
```

4. **Result:** Passed

Pytest Code:

```
1 def test_get_current_user_profile(client):  
2     """Test getting current user profile."""  
3     # Register and login to get token  
4     register_response = client.post('/api/auth/register', json={  
5         'email': 'parent@test.com',  
6         'password': 'Parent@123',  
7         'name': 'Test_Parent',  
8         'role': 'parent'  
9     })  
10    token = register_response.json()['access_token']  
11
```

```
12 # Get current user profile
13 response = client.get('/api/users/me', headers={
14     'Authorization': f'Bearer_{token}'
15 })
16
17 assert response.status_code == 200
18 data = response.json()
19 assert data['email'] == 'parent@test.com'
20 assert data['name'] == 'Test_Parent'
21 assert data['role'] == 'parent'
22 assert data['is_active'] == True
```

4.2 Update User Profile

Endpoint: PUT /api/users/user_id

Description: Update user profile information.

Test Cases:

4.2.1 Valid Profile Update

Test Case:

1. Update User Profile

2. Passed Inputs:

```
1 {
2     "name": "Updated Parent Name",
3     "avatar_url": "https://example.com/avatar.jpg"
4 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2     "id": "uuid",
3     "email": "parent@test.com",
4     "name": "Updated Parent Name",
5     "role": "parent",
6     "avatar_url": "https://example.com/avatar.jpg",
7     "created_at": "2025-01-01T00:00:00Z",
8     "updated_at": "2025-01-01T00:00:00Z",
9     "is_active": true,
10    "is_superuser": false,
11    "is_verified": false
12 }
```

4. Result: Passed

Pytest Code:

```
1 def test_update_user_profile(client):
2     """Test updating user profile."""
3     # Register and login to get token
4     register_response = client.post('/api/auth/register', json={
5         'email': 'parent@test.com',
6         'password': 'Parent@123',
7         'name': 'Test_Parent',
8         'role': 'parent'
9     })
10    token = register_response.json()['access_token']
11    user_id = register_response.json()['user']['id']
12
13    # Update user profile
14    response = client.put(f'/api/users/{user_id}', json={
15        'name': 'Updated_Parent_Name',
16        'avatar_url': 'https://example.com/avatar.jpg'
17    }, headers={'Authorization': f'Bearer_{token}'})
18
19    assert response.status_code == 200
20    data = response.json()
21    assert data['name'] == 'Updated_Parent_Name'
22    assert data['avatar_url'] == 'https://example.com/avatar.jpg'
```

5 Goals Management Endpoints

5.1 Get User Goals

Endpoint: GET /api/users/user_id/goals

Description: Get all goals for a user.

Test Cases:

5.1.1 Get Child Goals

Test Case:

1. Get Child Goals

2. Headers:

```
1 {
2     "Authorization": "Bearer parent_token"
3 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1  [
2    {
3      "id": "goal_uuid",
4      "user_id": "child_uuid",
5      "title": "Save for Toy",
6      "description": "Save coins for a new toy",
7      "target_amount": 100,
8      "current_amount": 25,
9      "icon": "toy",
10     "color": "blue",
11     "deadline": "2025-02-01T00:00:00Z",
12     "is_completed": false,
13     "created_at": "2025-01-01T00:00:00Z",
14     "updated_at": "2025-01-01T00:00:00Z"
15   }
16 ]
```

4. Result: Passed

Pytest Code:

```
1 def test_get_child_goals(client):
2     """Test getting child goals."""
3     # Register parent and child
4     parent_response = client.post('/api/auth/register', json={
5         'email': 'parent@test.com',
6         'password': 'Parent@123',
7         'name': 'Test_Parent',
8         'role': 'parent'
9     })
10    parent_token = parent_response.json()['access_token']
11
12    child_response = client.post('/api/auth/register', json={
13        'email': 'child@test.com',
14        'password': 'Child@123',
15        'name': 'Test_Child',
16        'role': 'younger_child'
17    })
18    child_token = child_response.json()['access_token']
19    child_id = child_response.json()['user']['id']
20
21    # Create a goal for child
22    client.post(f'/api/users/{child_id}/goals', json={
23        'title': 'Save_for_Toy',
24        'description': 'Save_coins_for_a_new_toy',
25        'target_amount': 100,
26        'icon': 'toy',
27        'color': 'blue',
28        'deadline': '2025-02-01T00:00:00Z'
29    }, headers={'Authorization': f'Bearer_{child_token}'})
30
31    # Get child goals as parent
32    response = client.get(f'/api/users/{child_id}/goals', headers={
```

```
33         'Authorization': f'Bearer_{parent_token}'
34     })
35
36     assert response.status_code == 200
37     data = response.json()
38     assert len(data) > 0
39     assert data[0]['title'] == 'Save_for_Toy'
40     assert data[0]['target_amount'] == 100
```

5.2 Create Goal

Endpoint: POST /api/users/user_id/goals

Description: Create a new financial goal.

Test Cases:

5.2.1 Valid Goal Creation

Test Case:

1. Create New Goal

2. Passed Inputs:

```
1  {
2    "title": "Save for Game",
3    "description": "Save coins for a new video game",
4    "target_amount": 200,
5    "icon": "game",
6    "color": "green",
7    "deadline": "2025-03-01T00:00:00Z"
8  }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1  {
2    "id": "goal_uuid",
3    "user_id": "child_uuid",
4    "title": "Save for Game",
5    "description": "Save coins for a new video game",
6    "target_amount": 200,
7    "current_amount": 0,
8    "icon": "game",
9    "color": "green",
10   "deadline": "2025-03-01T00:00:00Z",
11   "is_completed": false,
12   "created_at": "2025-01-01T00:00:00Z",
13   "updated_at": "2025-01-01T00:00:00Z"
14 }
```

4. **Result:** Passed

Pytest Code:

```
1 def test_create_goal(client):
2     """Test creating a new goal."""
3     # Register child user
4     child_response = client.post('/api/auth/register', json={
5         'email': 'child@test.com',
6         'password': 'Child@123',
7         'name': 'Test_Child',
8         'role': 'younger_child'
9     })
10    child_token = child_response.json()['access_token']
11    child_id = child_response.json()['user']['id']
12
13    # Create goal
14    response = client.post(f'/api/users/{child_id}/goals', json={
15        'title': 'Save_for_Game',
16        'description': 'Save_coins_for_a_new_video_game',
17        'target_amount': 200,
18        'icon': 'game',
19        'color': 'green',
20        'deadline': '2025-03-01T00:00:00Z'
21    }, headers={'Authorization': f'Bearer_{child_token}'})
22
23    assert response.status_code == 200
24    data = response.json()
25    assert data['title'] == 'Save_for_Game'
26    assert data['target_amount'] == 200
27    assert data['current_amount'] == 0
28    assert data['is_completed'] == False
```

5.3 Contribute to Goal

Endpoint: POST /api/users/user_id/goals/goal_id/contribute

Description: Add coins to a financial goal.

Test Cases:

5.3.1 Valid Goal Contribution

Test Case:

1. Contribute to Goal

2. Passed Inputs:

```
1 {
2     "amount": 10
3 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2   "goal": {
3     "id": "goal_uuid",
4     "user_id": "child_uuid",
5     "title": "Save for Toy",
6     "description": "Save coins for a new toy",
7     "target_amount": 100,
8     "current_amount": 35,
9     "icon": "toy",
10    "color": "blue",
11    "deadline": "2025-02-01T00:00:00Z",
12    "is_completed": false,
13    "created_at": "2025-01-01T00:00:00Z",
14    "updated_at": "2025-01-01T00:00:00Z"
15  },
16  "transaction": {
17    "id": "transaction_uuid",
18    "user_id": "child_uuid",
19    "type": "save",
20    "amount": 10,
21    "description": "Contributed to goal: Save for Toy",
22    "category": "goal",
23    "reference_id": "goal_uuid",
24    "reference_type": "goal",
25    "created_at": "2025-01-01T00:00:00Z"
26  },
27  "new_coin_balance": 15
28 }
```

4. Result: Passed

Pytest Code:

```
1 def test_contribute_to_goal(client):
2     """Test contributing to a goal."""
3     # Register child user
4     child_response = client.post('/api/auth/register', json={
5         'email': 'child@test.com',
6         'password': 'Child@123',
7         'name': 'Test Child',
8         'role': 'younger_child'
9     })
10    child_token = child_response.json()['access_token']
11    child_id = child_response.json()['user']['id']
12
13    # Create goal first
14    goal_response = client.post(f'/api/users/{child_id}/goals', json=
15        {
16            'title': 'Save for Toy',
17            'description': 'Save coins for a new toy',
18            'target_amount': 100,
19            'icon': 'toy',
```

```
19     'color': 'blue',
20     'deadline': '2025-02-01T00:00:00Z'
21 }, headers={'Authorization': f'Bearer_{child_token}'})
22 goal_id = goal_response.json()['id']
23
24 # Contribute to goal
25 response = client.post(f'/api/users/{child_id}/goals/{goal_id}/
26     contribute', json={
27     'amount': 10
28 }, headers={'Authorization': f'Bearer_{child_token}'})
29
30 assert response.status_code == 200
31 data = response.json()
32 assert 'goal' in data
33 assert 'transaction' in data
34 assert 'new_coin_balance' in data
35 assert data['goal']['current_amount'] == 10
```

6 Dashboard Endpoints

6.1 Get Child Dashboard

Endpoint: GET /api/child/dashboard

Description: Get dashboard data for child users.

Test Cases:

6.1.1 Child Dashboard Data

Test Case:

1. Get Child Dashboard

2. Headers:

```
1 {
2   "Authorization": "Bearer child_token"
3 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2   "user": {
3     "id": "child_uuid",
4     "email": "child@test.com",
5     "name": "Test Child",
6     "role": "younger_child",
7     "avatar_url": null,
8     "created_at": "2025-01-01T00:00:00Z",
```

```
9      "updated_at": "2025-01-01T00:00:00Z",
10      "is_active": true,
11      "is_superuser": false,
12      "is_verified": false
13    },
14    "stats": {
15      "total_coins": 25,
16      "level": 1,
17      "streak_days": 3,
18      "goals_count": 2,
19      "completed_tasks": 5
20    },
21    "active_goals": [
22      {
23        "id": "goal_uuid",
24        "user_id": "child_uuid",
25        "title": "Save for Toy",
26        "description": "Save coins for a new toy",
27        "target_amount": 100,
28        "current_amount": 25,
29        "icon": "toy",
30        "color": "blue",
31        "deadline": "2025-02-01T00:00:00Z",
32        "is_completed": false,
33        "created_at": "2025-01-01T00:00:00Z",
34        "updated_at": "2025-01-01T00:00:00Z"
35      }
36    ],
37    "pending_tasks": [
38      {
39        "id": "task_uuid",
40        "title": "Clean Room",
41        "description": "Clean your room",
42        "coins_reward": 10,
43        "assigned_by": "parent_uuid",
44        "assigned_to": "child_uuid",
45        "status": "pending",
46        "due_date": "2025-01-02T00:00:00Z",
47        "requires_approval": true,
48        "created_at": "2025-01-01T00:00:00Z"
49      }
50    ],
51    "recent_transactions": [
52      {
53        "id": "transaction_uuid",
54        "user_id": "child_uuid",
55        "type": "earn",
56        "amount": 10,
57        "description": "Completed task: Clean Room",
58        "category": "task",
59        "reference_id": "task_uuid",
60        "reference_type": "task",
61        "created_at": "2025-01-01T00:00:00Z"
62      }
63    ],
64    "achievements": [
65      {
66        "id": "achievement_uuid",
```

```
67         "title": "First Goal",
68         "description": "Created your first goal",
69         "icon": "star",
70         "rarity": "common",
71         "points_reward": 5,
72         "earned_at": "2025-01-01T00:00:00Z"
73     }
74 ]
75 }
```

4. Result: Passed

Pytest Code:

```
1 def test_get_child_dashboard(client):
2     """Test getting child dashboard data."""
3     # Register child user
4     child_response = client.post('/api/auth/register', json={
5         'email': 'child@test.com',
6         'password': 'Child@123',
7         'name': 'Test_Child',
8         'role': 'younger_child'
9     })
10    child_token = child_response.json()['access_token']
11
12    # Get child dashboard
13    response = client.get('/api/child/dashboard', headers={
14        'Authorization': f'Bearer_{child_token}'
15    })
16
17    assert response.status_code == 200
18    data = response.json()
19    assert 'user' in data
20    assert 'stats' in data
21    assert 'active_goals' in data
22    assert 'pending_tasks' in data
23    assert 'recent_transactions' in data
24    assert 'achievements' in data
25    assert data['user']['role'] == 'younger_child'
```

6.2 Get Parent Dashboard

Endpoint: GET /api/parent/dashboard

Description: Get dashboard data for parent users.

Test Cases:

6.2.1 Parent Dashboard Data

Test Case:

1. Get Parent Dashboard

2. Headers:

```
1 {  
2   "Authorization": "Bearer parent_token"  
3 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {  
2   "user": {  
3     "id": "parent_uuid",  
4     "email": "parent@test.com",  
5     "name": "Test Parent",  
6     "role": "parent",  
7     "avatar_url": null,  
8     "created_at": "2025-01-01T00:00:00Z",  
9     "updated_at": "2025-01-01T00:00:00Z",  
10    "is_active": true,  
11    "is_superuser": false,  
12    "is_verified": false  
13  },  
14  "stats": {  
15    "total_coins": 150,  
16    "level": 1,  
17    "streak_days": 0,  
18    "goals_count": 2,  
19    "completed_tasks": 8  
20  },  
21  "children": [  
22    {  
23      "id": "child_uuid",  
24      "email": "child@test.com",  
25      "name": "Test Child",  
26      "role": "younger_child",  
27      "avatar_url": null,  
28      "created_at": "2025-01-01T00:00:00Z",  
29      "updated_at": "2025-01-01T00:00:00Z",  
30      "is_active": true,  
31      "is_superuser": false,  
32      "is_verified": false  
33    }  
34  ],  
35  "recent_transactions": [  
36    {  
37      "id": "transaction_uuid",  
38      "user_id": "child_uuid",  
39      "type": "earn",  
40      "amount": 10,  
41      "description": "Completed task: Clean Room",  
42      "category": "task",  
43      "reference_id": "task_uuid",  
44      "reference_type": "task",  
45      "created_at": "2025-01-01T00:00:00Z"
```

```
46     }
47   ],
48   "pending_redemptions": [
49     {
50       "id": "redemption_uuid",
51       "user_id": "child_uuid",
52       "coins_amount": 50,
53       "cash_amount": 5.0,
54       "description": "Convert coins to cash",
55       "status": "pending",
56       "created_at": "2025-01-01T00:00:00Z"
57     }
58   ]
59 }
```

4. Result: Passed

Pytest Code:

```
1 def test_get_parent_dashboard(client):
2     """Test getting parent dashboard data."""
3     # Register parent user
4     parent_response = client.post('/api/auth/register', json={
5         'email': 'parent@test.com',
6         'password': 'Parent@123',
7         'name': 'Test_Parent',
8         'role': 'parent'
9     })
10    parent_token = parent_response.json()['access_token']
11
12    # Get parent dashboard
13    response = client.get('/api/parent/dashboard', headers={
14        'Authorization': f'Bearer_{parent_token}'
15    })
16
17    assert response.status_code == 200
18    data = response.json()
19    assert 'user' in data
20    assert 'stats' in data
21    assert 'children' in data
22    assert 'recent_transactions' in data
23    assert 'pending_redemptions' in data
24    assert data['user']['role'] == 'parent'
```

7 Task Management Endpoints

7.1 Get User Tasks

Endpoint: GET /api/tasks

Description: Get tasks for current user.

Test Cases:

7.1.1 Get Child Tasks

Test Case:

1. Get Child Tasks

2. Headers:

```
1 {  
2   "Authorization": "Bearer child_token"  
3 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 [  
2   {  
3     "id": "task_uuid",  
4     "title": "Clean Room",  
5     "description": "Clean your room thoroughly",  
6     "coins_reward": 10,  
7     "assigned_by": "parent_uuid",  
8     "assigned_to": "child_uuid",  
9     "status": "pending",  
10    "due_date": "2025-01-02T00:00:00Z",  
11    "requires_approval": true,  
12    "created_at": "2025-01-01T00:00:00Z"  
13  }  
14 ]
```

4. Result: Passed

Pytest Code:

```
1 def test_get_child_tasks(client):  
2     """Test getting child tasks."""  
3     # Register child user  
4     child_response = client.post('/api/auth/register', json={  
5         'email': 'child@test.com',  
6         'password': 'Child@123',  
7         'name': 'Test_Child',  
8         'role': 'younger_child'  
9     })  
10    child_token = child_response.json()['access_token']  
11  
12    # Get child tasks  
13    response = client.get('/api/tasks', headers={  
14        'Authorization': f'Bearer_{child_token}'  
15    })  
16  
17    assert response.status_code == 200  
18    data = response.json()
```

```
19 assert isinstance(data, list)
```

7.2 Create Task

Endpoint: POST /api/tasks

Description: Create a new task.

Test Cases:

7.2.1 Valid Task Creation

Test Case:

1. Create New Task

2. Passed Inputs:

```
1 {
2   "title": "Do Homework",
3   "description": "Complete math homework",
4   "coins_reward": 15,
5   "assigned_to": "child_uuid",
6   "due_date": "2025-01-03T00:00:00Z",
7   "requires_approval": true
8 }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2   "id": "task_uuid",
3   "title": "Do Homework",
4   "description": "Complete math homework",
5   "coins_reward": 15,
6   "assigned_by": "parent_uuid",
7   "assigned_to": "child_uuid",
8   "status": "pending",
9   "due_date": "2025-01-03T00:00:00Z",
10  "requires_approval": true,
11  "created_at": "2025-01-01T00:00:00Z"
12 }
```

4. Result: Passed

Pytest Code:

```
1 def test_create_task(client):
2     """Test creating a new task."""
3     # Register parent and child
4     parent_response = client.post('/api/auth/register', json={
5         'email': 'parent@test.com',
```



```
6         'password': 'Parent@123',
7         'name': 'Test_Parent',
8         'role': 'parent'
9     })
10    parent_token = parent_response.json()['access_token']
11
12    child_response = client.post('/api/auth/register', json={
13        'email': 'child@test.com',
14        'password': 'Child@123',
15        'name': 'Test_Child',
16        'role': 'younger_child'
17    })
18    child_id = child_response.json()['user']['id']
19
20    # Create task
21    response = client.post('/api/tasks', json={
22        'title': 'Do_Homework',
23        'description': 'Complete_math_homework',
24        'coins_reward': 15,
25        'assigned_to': child_id,
26        'due_date': '2025-01-03T00:00:00Z',
27        'requires_approval': True
28    }, headers={'Authorization': f'Bearer_{parent_token}'})
29
30    assert response.status_code == 200
31    data = response.json()
32    assert data['title'] == 'Do_Homework'
33    assert data['coins_reward'] == 15
34    assert data['status'] == 'pending'
```

8 Shop System Endpoints

8.1 Get Shop Items

Endpoint: GET /api/shop/items

Description: Get available shop items.

Test Cases:

8.1.1 Get Shop Items

Test Case:

1. Get Shop Items

2. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1  [
2    {
3      "id": "item_uuid",
4      "name": "Toy Car",
5      "description": "A cool toy car",
6      "price": 50,
7      "category": "toys",
8      "emoji": "[automobile]",
9      "available": true
10   },
11   {
12     "id": "item_uuid2",
13     "name": "Video Game",
14     "description": "Fun video game",
15     "price": 200,
16     "category": "games",
17     "emoji": "[Video-Game]",
18     "available": true
19   }
20 ]
```

3. Result: Passed

Pytest Code:

```
1 def test_get_shop_items(client):
2     """Test getting shop items."""
3     response = client.get('/api/shop/items')
4
5     assert response.status_code == 200
6     data = response.json()
7     assert isinstance(data, list)
8     if len(data) > 0:
9         assert 'id' in data[0]
10        assert 'name' in data[0]
11        assert 'price' in data[0]
12        assert 'available' in data[0]
```

8.2 Purchase Item

Endpoint: POST /api/shop/purchase

Description: Purchase an item from the shop.

Test Cases:

8.2.1 Valid Purchase

Test Case:

1. Purchase Item

2. Headers:

```
1 {  
2   "Authorization": "Bearer child_token"  
3 }
```

3. Passed Inputs:

```
1 {  
2   "item_id": "item_uuid",  
3   "quantity": 1  
4 }
```

4. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {  
2   "success": true,  
3   "message": "Item purchased successfully",  
4   "item": {  
5     "id": "item_uuid",  
6     "name": "Toy Car",  
7     "description": "A cool toy car",  
8     "price": 50,  
9     "category": "toys",  
10    "emoji": "[automobile]",  
11    "available": true  
12  },  
13  "coins_spent": 50,  
14  "new_balance": 25  
15 }
```

5. Result: Passed

Pytest Code:

```
1 def test_purchase_item(client):  
2     """Test purchasing an item from shop."""  
3     # Register child user  
4     child_response = client.post('/api/auth/register', json={  
5         'email': 'child@test.com',  
6         'password': 'Child@123',  
7         'name': 'Test_Child',  
8         'role': 'younger_child'  
9     })  
10    child_token = child_response.json()['access_token']  
11  
12    # Get shop items first  
13    shop_response = client.get('/api/shop/items')  
14    shop_items = shop_response.json()  
15  
16    if len(shop_items) > 0:
```

```
17     item_id = shop_items[0]['id']
18
19     # Purchase item
20     response = client.post('/api/shop/purchase', json={
21         'item_id': item_id,
22         'quantity': 1
23     }, headers={'Authorization': f'Bearer_{child_token}'})
24
25     assert response.status_code == 200
26     data = response.json()
27     assert 'success' in data
28     assert 'coins_spent' in data
29     assert 'new_balance' in data
```

9 Redemption System Endpoints

9.1 Get Redemption Requests

Endpoint: GET /api/users/user_id/conversion-requests

Description: Get user's redemption requests.

Test Cases:

9.1.1 Get Child Redemption Requests

Test Case:

1. Get Child Redemption Requests

2. Headers:

```
1  {
2    "Authorization": "Bearer child_token"
3  }
```

3. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1  [
2    {
3      "id": "redemption_uuid",
4      "user_id": "child_uuid",
5      "coins_amount": 50,
6      "cash_amount": 5.0,
7      "description": "Convert coins to cash",
8      "status": "pending",
9      "created_at": "2025-01-01T00:00:00Z"
10   }
11  ]
```

4. **Result:** Passed

Pytest Code:

```
1 def test_get_child_redemption_requests(client):
2     """Test getting child redemption requests."""
3     # Register child user
4     child_response = client.post('/api/auth/register', json={
5         'email': 'child@test.com',
6         'password': 'Child@123',
7         'name': 'Test_Child',
8         'role': 'younger_child'
9     })
10    child_token = child_response.json()['access_token']
11    child_id = child_response.json()['user']['id']
12
13    # Get redemption requests
14    response = client.get(f'/api/users/{child_id}/conversion-requests', headers={
15        'Authorization': f'Bearer_{child_token}'
16    })
17
18    assert response.status_code == 200
19    data = response.json()
20    assert isinstance(data, list)
```

9.2 Create Redemption Request

Endpoint: POST /api/users/user_id/conversion-requests

Description: Create a new redemption request.

Test Cases:

9.2.1 Valid Redemption Request

Test Case:

1. Create Redemption Request

2. Headers:

```
1 {
2     "Authorization": "Bearer child_token"
3 }
```

3. Passed Inputs:

```
1 {
2     "coins_amount": 100,
3     "description": "Convert coins to cash for allowance"
4 }
```

4. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {
2   "id": "redemption_uuid",
3   "user_id": "child_uuid",
4   "coins_amount": 100,
5   "cash_amount": 10.0,
6   "description": "Convert coins to cash for allowance",
7   "status": "pending",
8   "created_at": "2025-01-01T00:00:00Z"
9 }
```

5. Result: Passed

Pytest Code:

```
1 def test_create_redemption_request(client):
2     """Test creating a redemption request."""
3     # Register child user
4     child_response = client.post('/api/auth/register', json={
5         'email': 'child@test.com',
6         'password': 'Child@123',
7         'name': 'Test_Child',
8         'role': 'younger_child'
9     })
10    child_token = child_response.json()['access_token']
11    child_id = child_response.json()['user']['id']
12
13    # Create redemption request
14    response = client.post(f'/api/users/{child_id}/conversion-requests', json={
15        'coins_amount': 100,
16        'description': 'Convert_coins_to_cash_for_allowance'
17    }, headers={'Authorization': f'Bearer_{child_token}'})
18
19    assert response.status_code == 200
20    data = response.json()
21    assert data['coins_amount'] == 100
22    assert data['status'] == 'pending'
23    assert 'cash_amount' in data
```

10 Health Check Endpoints

10.1 Health Check

Endpoint: GET /health

Description: Perform a health check on the application.

Test Cases:

10.1.1 Health Check

Test Case:

1. Health Check

2. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {  
2   "status": "OK"  
3 }
```

3. Result: Passed

Pytest Code:

```
1 def test_health_check(client):  
2     """Test health check endpoint."""  
3     response = client.get('/health')  
4  
5     assert response.status_code == 200  
6     data = response.json()  
7     assert data['status'] == 'OK'
```

10.2 API Status

Endpoint: GET /api/status

Description: Get backend status and environment configuration.

Test Cases:

10.2.1 API Status

Test Case:

1. Get API Status

2. Expected Output:

- HTTP Status Code: 200
- JSON Response:

```
1 {  
2   "status": "OK",  
3   "environment": "development",  
4   "database_type": "SQLite",  
5   "secret_key_configured": true,  
6   "algorithm": "HS256"  
7 }
```

3. **Result:** Passed

Pytest Code:

```
1 def test_api_status(client):
2     """Test API status endpoint."""
3     response = client.get('/api/status')
4
5     assert response.status_code == 200
6     data = response.json()
7     assert data['status'] == 'OK'
8     assert 'environment' in data
9     assert 'database_type' in data
10    assert 'secret_key_configured' in data
11    assert 'algorithm' in data
```

11 Error Handling Test Cases

11.1 Unauthorized Access

Test Case:

1. Access Protected Endpoint Without Token

2. **Headers:** None

3. **Expected Output:**

- HTTP Status Code: 401
- JSON Response:

```
1 {
2     "detail": "Not authenticated"
3 }
```

4. **Result:** Passed

Pytest Code:

```
1 def test_unauthorized_access(client):
2     """Test accessing protected endpoint without token."""
3     response = client.get('/api/users/me')
4
5     assert response.status_code == 401
6     data = response.json()
7     assert data['detail'] == 'Not authenticated'
```


11.2 Invalid Token

Test Case:

1. Access Protected Endpoint With Invalid Token

2. Headers:

```
1 {  
2   "Authorization": "Bearer invalid_token"  
3 }
```

3. Expected Output:

- HTTP Status Code: 401
- JSON Response:

```
1 {  
2   "detail": "Could not validate credentials"  
3 }
```

4. Result: Passed

Pytest Code:

```
1 def test_invalid_token(client):  
2     """Test accessing protected endpoint with invalid token."""  
3     response = client.get('/api/users/me', headers={  
4         'Authorization': 'Bearer_invalid_token'  
5     })  
6  
7     assert response.status_code == 401  
8     data = response.json()  
9     assert 'Could_not_validate_credentials' in data['detail']
```

11.3 Resource Not Found

Test Case:

1. Access Non-existent Resource

2. Headers:

```
1 {  
2   "Authorization": "Bearer valid_token"  
3 }
```

3. Expected Output:

- HTTP Status Code: 404
- JSON Response:

```
1 {  
2   "detail": "Not found"  
3 }
```

4. Result: Passed

Pytest Code:

```
1 def test_resource_not_found(client):  
2     """Test accessing non-existent resource."""  
3     # Register user to get valid token  
4     register_response = client.post('/api/auth/register', json={  
5         'email': 'test@test.com',  
6         'password': 'Test@123',  
7         'name': 'Test_User',  
8         'role': 'parent'  
9     })  
10    token = register_response.json()['access_token']  
11  
12    # Try to access non-existent resource  
13    response = client.get('/api/non-existent-endpoint', headers={  
14        'Authorization': f'Bearer_{token}'  
15    })  
16  
17    assert response.status_code == 404  
18    data = response.json()  
19    assert data['detail'] == 'Not_found'
```

12 Conclusion

This document provides comprehensive test cases for the CoinCraft FastAPI backend application. The test cases cover authentication, user management, goals, tasks, dashboard functionality, shop system, redemption system, and error handling. Each test case includes the expected inputs, outputs, and HTTP status codes to ensure proper API functionality.

The test cases are designed to validate:

- Proper authentication and authorization
- CRUD operations for all entities
- Role-based access control
- Data validation and error handling
- Integration between different system components