

Casos de Prueba para Postman - MinGO API

Esta guia contiene todos los casos de prueba para validar los endpoints de la API MinGO usando Postman.

Configuracion Inicial

Variables de Entorno

Configura las siguientes variables en Postman:

- `base_url` = `http://localhost:3000` - URL base de la API
- `access_token` = (vacio) - Token JWT de acceso
- `refresh_token` = (vacio) - Token JWT de refresco
- `user_id` = (vacio) - ID del usuario autenticado
- `child_id` = (vacio) - ID del hijo creado
- `class_id` = (vacio) - ID de la clase creada
- `class_code` = (vacio) - Código de la clase
- `verification_token` = (vacio) - Token de verificacion
- `reset_token` = (vacio) - Token de reset de password

Headers Comunes

Para endpoints protegidos, usar:

```
Authorization: Bearer {{access_token}}
Content-Type: application/json
```

1. AUTH - Autenticacion

1.1 Registro de Usuario

Endpoint: POST `{{base_url}}/auth/register`

Caso 1.1.1: Registro exitoso de PADRE

```
{
  "email": "padre@test.com",
  "password": "Test123!@",
  "name": "Juan Perez",
  "role": "PADRE"
}
```

Respuesta esperada: 201 Created

```
{
  "accessToken": "eyJhbG...",
  "refreshToken": "eyJhbG...",
```

```

"user": {
    "id": "uuid",
    "email": "padre@test.com",
    "name": "Juan Perez",
    "role": "PADRE",
    "isActive": true,
    "emailVerified": false,
    "createdAt": "2024-01-01T00:00:00.000Z"
}
}

```

Post-script:

```

if (pm.response.code === 201) {
    var json = pm.response.json();
    pm.environment.set("access_token", json.accessToken);
    pm.environment.set("refresh_token", json.refreshToken);
    pm.environment.set("user_id", json.user.id);
}

```

Caso 1.1.2: Registro exitoso de DOCENTE

```
{
    "email": "docente@test.com",
    "password": "Test123!@",
    "name": "Maria Garcia",
    "role": "DOCENTE"
}
```

Respuesta esperada: 201 Created

Caso 1.1.3: Error - Email ya registrado

```
{
    "email": "padre@test.com",
    "password": "Test123!@",
    "name": "Otro Usuario",
    "role": "PADRE"
}
```

Respuesta esperada: 400 Bad Request

```
{
    "statusCode": 400,
    "message": "Email already registered"
}
```

Caso 1.1.4: Error - Email invalido

```
{  
    "email": "email-invalido",  
    "password": "Test123!@",  
    "name": "Usuario",  
    "role": "PADRE"  
}
```

Respuesta esperada: 400 Bad Request

Caso 1.1.5: Error - Password muy corta

```
{  
    "email": "nuevo@test.com",  
    "password": "123",  
    "name": "Usuario",  
    "role": "PADRE"  
}
```

Respuesta esperada: 400 Bad Request

Caso 1.1.6: Error - Nombre muy corto

```
{  
    "email": "nuevo@test.com",  
    "password": "Test123!@",  
    "name": "A",  
    "role": "PADRE"  
}
```

Respuesta esperada: 400 Bad Request

Caso 1.1.7: Error - Rol invalido

```
{  
    "email": "nuevo@test.com",  
    "password": "Test123!@",  
    "name": "Usuario",  
    "role": "INVALID_ROLE"  
}
```

Respuesta esperada: 400 Bad Request

1.2 Login

Endpoint: POST {{base_url}}/auth/login

Caso 1.2.1: Login exitoso

```
{  
    "email": "padre@test.com",  
    "password": "Test123!@"  
}
```

Respuesta esperada: 200 OK

```
{  
    "accessToken": "eyJhbG... ",  
    "refreshToken": "eyJhbG... ",  
    "user": {  
        "id": "uuid",  
        "email": "padre@test.com",  
        "name": "Juan Perez",  
        "role": "PADRE"  
    }  
}
```

Post-script:

```
if (pm.response.code === 200) {  
    var json = pm.response.json();  
    pm.environment.set("access_token", json.accessToken);  
    pm.environment.set("refresh_token", json.refreshToken);  
    pm.environment.set("user_id", json.user.id);  
}
```

Caso 1.2.2: Error - Credenciales invalidas

```
{  
    "email": "padre@test.com",  
    "password": "wrongpassword"  
}
```

Respuesta esperada: 401 Unauthorized

```
{  
    "statusCode": 401,  
    "message": "Invalid credentials"  
}
```

Caso 1.2.3: Error - Usuario no existe

```
{  
    "email": "noexiste@test.com",  
    "password": "Test123!@"  
}
```

Respuesta esperada: 401 Unauthorized

1.3 Refresh Token

Endpoint: POST {{base_url}}/auth/refresh

Caso 1.3.1: Refresh exitoso

```
{  
  "refreshToken": "{{refresh_token}}"  
}
```

Respuesta esperada: 200 OK Post-script:

```
if (pm.response.code === 200) {  
  var json = pm.response.json();  
  pm.environment.set("access_token", json.accessToken);  
  pm.environment.set("refresh_token", json.refreshToken);  
}
```

Caso 1.3.2: Error - Token invalido

```
{  
  "refreshToken": "token-invalido"  
}
```

Respuesta esperada: 401 Unauthorized

1.4 Obtener Perfil

Endpoint: GET {{base_url}}/auth/profile **Headers:** Authorization: Bearer {{access_token}}

Caso 1.4.1: Perfil obtenido exitosamente **Respuesta esperada:** 200 OK

```
{  
  "id": "uuid",  
  "email": "padre@test.com",  
  "name": "Juan Perez",  
  "role": "PADRE",  
  "isActive": true,  
  "emailVerified": false,  
  "createdAt": "2024-01-01T00:00:00.000Z"  
}
```

Caso 1.4.2: Error - Sin token Headers: (sin Authorization) **Respuesta esperada:** 401 Unauthorized

Caso 1.4.3: Error - Token expirado/invalido Headers: Authorization: Bearer token-invalido **Respuesta esperada:** 401 Unauthorized

1.5 Verificacion de Email

Endpoint: POST {{base_url}}/auth/verify-email

Caso 1.5.1: Verificacion exitosa

```
{  
  "token": "{{verification_token}}"  
}
```

Respuesta esperada: 200 OK

```
{  
  "success": true,  
  "message": "Email verified successfully. Welcome to MinGO!"  
}
```

Caso 1.5.2: Error - Token invalido

```
{  
  "token": "token-invalido"  
}
```

Respuesta esperada: 400 Bad Request

```
{  
  "statusCode": 400,  
  "message": "Invalid or expired verification token"  
}
```

1.6 Reenviar Verificacion

Endpoint: POST {{base_url}}/auth/resend-verification

Caso 1.6.1: Reenvio exitoso

```
{  
  "email": "padre@test.com"  
}
```

Respuesta esperada: 200 OK

```
{  
    "success": true,  
    "message": "If an account exists with this email, a verification link has been sent."  
}
```

Caso 1.6.2: Email no existente (mismo mensaje por seguridad)

```
{  
    "email": "noexiste@test.com"  
}
```

Respuesta esperada: 200 OK

1.7 Olvide mi Contrasena

Endpoint: POST {{base_url}}/auth/forgot-password

Caso 1.7.1: Solicitud exitosa

```
{  
    "email": "padre@test.com"  
}
```

Respuesta esperada: 200 OK

```
{  
    "success": true,  
    "message": "If an account exists with this email, a password reset link has been sent."  
}
```

1.8 Restablecer Contrasena

Endpoint: POST {{base_url}}/auth/reset-password

Caso 1.8.1: Reset exitoso

```
{  
    "token": "{{reset_token}}",  
    "newPassword": "NewPass123!@"  
}
```

Respuesta esperada: 200 OK

```
{  
    "success": true,  
    "message": "Password has been reset successfully. You can now login with your new password."  
}
```

Caso 1.8.2: Error - Token invalido

```
{  
    "token": "token-invalido",  
    "newPassword": "NewPass123!@"  
}
```

Respuesta esperada: 400 Bad Request

Caso 1.8.3: Error - Password debil

```
{  
    "token": "{{reset_token}}",  
    "newPassword": "123"  
}
```

Respuesta esperada: 400 Bad Request

2. CHILDREN - Gestion de Hijos

Nota: Todos los endpoints requieren autenticacion y rol PADRE.

2.1 Crear Hijo

Endpoint: POST {{base_url}}/children Headers: Authorization:
Bearer {{access_token}}

Caso 2.1.1: Creacion exitosa

```
{  
    "name": "Pedrito Perez",  
    "birthDate": "2019-05-15",  
    "disabilityType": "AUDITIVA",  
    "notes": "Le gusta aprender jugando"  
}
```

Respuesta esperada: 201 Created

```
{  
    "id": "uuid",  
    "parentId": "uuid",  
    "name": "Pedrito Perez",  
    "birthDate": "2019-05-15T00:00:00.000Z",  
    "age": 5,  
    "ageCategory": "3-5 años",  
    "disabilityType": "AUDITIVA",  
    "notes": "Le gusta aprender jugando",  
    "createdAt": "...",  
    "updatedAt": "..."  
}
```

```
    "updatedAt": "..."  
}
```

Post-script:

```
if (pm.response.code === 201) {  
    var json = pm.response.json();  
    pm.environment.set("child_id", json.id);  
}
```

Caso 2.1.2: Creacion sin tipo de discapacidad

```
{  
    "name": "Maria Perez",  
    "birthDate": "2020-08-20"  
}
```

Respuesta esperada: 201 Created

Caso 2.1.3: Error - Nombre vacio

```
{  
    "name": "",  
    "birthDate": "2019-05-15"  
}
```

Respuesta esperada: 400 Bad Request

Caso 2.1.4: Error - Edad fuera de rango (muy pequeno)

```
{  
    "name": "Bebe",  
    "birthDate": "2024-01-01"  
}
```

Respuesta esperada: 400 Bad Request

Caso 2.1.5: Error - Edad fuera de rango (muy grande)

```
{  
    "name": "Adulto",  
    "birthDate": "2000-01-01"  
}
```

Respuesta esperada: 400 Bad Request

Caso 2.1.6: Error - Tipo de discapacidad invalido

```
{  
    "name": "Test",  
}
```

```
        "birthDate": "2019-05-15",
        "disabilityType": "INVALIDO"
    }
```

Respuesta esperada: 400 Bad Request

Caso 2.1.7: Error - Usuario no es PADRE Nota: Hacer login como DOCENTE primero

```
{
    "name": "Test",
    "birthDate": "2019-05-15"
}
```

Respuesta esperada: 403 Forbidden

2.2 Listar Hijos

Endpoint: GET {{base_url}}/children Headers: Authorization: Bearer {{access_token}}

Caso 2.2.1: Lista exitosa Respuesta esperada: 200 OK

```
{
    "children": [
        {
            "id": "uuid",
            "parentId": "uuid",
            "name": "Pedrito Perez",
            "birthDate": "2019-05-15T00:00:00.000Z",
            "age": 5,
            "ageCategory": "3-5 años",
            "disabilityType": "AUDITIVA"
        }
    ],
    "total": 1
}
```

Caso 2.2.2: Lista vacia (usuario sin hijos) Respuesta esperada: 200 OK

```
{
    "children": [],
    "total": 0
}
```

2.3 Obtener Hijo por ID

Endpoint: GET {{base_url}}/children/{{child_id}} Headers:
Authorization: Bearer {{access_token}}

Caso 2.3.1: Hijo encontrado Respuesta esperada: 200 OK

Caso 2.3.2: Error - Hijo no encontrado Endpoint: GET {{base_url}}/children/uuid-no-existente
Respuesta esperada: 404 Not Found

Caso 2.3.3: Error - Hijo de otro padre Nota: Intentar acceder al hijo de
otro usuario Respuesta esperada: 403 Forbidden

2.4 Actualizar Hijo

Endpoint: PUT {{base_url}}/children/{{child_id}} Headers:
Authorization: Bearer {{access_token}}

Caso 2.4.1: Actualizacion exitosa

```
{  
  "name": "Pedro Perez Garcia",  
  "notes": "Notas actualizadas"  
}
```

Respuesta esperada: 200 OK

Caso 2.4.2: Actualizar tipo de discapacidad

```
{  
  "disabilityType": "AMBAS"  
}
```

Respuesta esperada: 200 OK

Caso 2.4.3: Error - Nombre muy largo

```
{  
  "name": "A".repeat(101)  
}
```

Respuesta esperada: 400 Bad Request

2.5 Eliminar Hijo

Endpoint: DELETE {{base_url}}/children/{{child_id}} Headers:
Authorization: Bearer {{access_token}}

Caso 2.5.1: Eliminacion exitosa Respuesta esperada: 204 No Content

Caso 2.5.2: Error - Hijo no encontrado Respuesta esperada: 404 Not Found

3. PROGRESS - Progreso de Aprendizaje

3.1 Registrar Intento

Endpoint: POST {{base_url}}/progress/attempt Headers: Authorization:
Bearer {{access_token}}

Caso 3.1.1: Intento correcto

```
{
  "lessonId": "{{lesson_id}}",
  "activityId": "{{activity_id}}",
  "correct": true,
  "timeSpent": 30
}
```

Respuesta esperada: 201 Created

```
{
  "id": "uuid",
  "userId": "uuid",
  "lessonId": "uuid",
  "completed": false,
  "accuracy": 100,
  "totalAttempts": 1,
  "correctAttempts": 1,
  "timeSpent": 30
}
```

Caso 3.1.2: Intento incorrecto

```
{
  "lessonId": "{{lesson_id}}",
  "activityId": "{{activity_id}}",
  "correct": false,
  "timeSpent": 45
}
```

Respuesta esperada: 201 Created

Caso 3.1.3: Error - Tiempo negativo

```
{  
    "lessonId": "{{lesson_id}}",  
    "activityId": "{{activity_id}}",  
    "correct": true,  
    "timeSpent": -10  
}
```

Respuesta esperada: 400 Bad Request

3.2 Completar Lección

Endpoint: POST {{base_url}}/progress/complete
Headers: Authorization: Bearer {{access_token}}

Caso 3.2.1: Lección completada exitosamente

```
{  
    "lessonId": "{{lesson_id}}"  
}
```

Respuesta esperada: 200 OK

```
{  
    "progress": {  
        "id": "uuid",  
        "completed": true,  
        "completedAt": "2024-01-01T00:00:00.000Z"  
    },  
    "levelUnlocked": null  
}
```

Caso 3.2.2: Lección completada desbloquea nivel **Respuesta esperada:**
200 OK

```
{  
    "progress": { ... },  
    "levelUnlocked": {  
        "id": "uuid",  
        "levelSectionId": "uuid",  
        "levelName": "Intermedio",  
        "unlockedAt": "..."  
    }  
}
```

3.3 Obtener Progreso

Endpoint: GET {{base_url}}/progress Headers: Authorization: Bearer {{access_token}}

Caso 3.3.1: Obtener todo el progreso Respuesta esperada: 200 OK

Caso 3.3.2: Filtrar por lección Endpoint: GET {{base_url}}/progress?lessonId={{lesson_id}}
Respuesta esperada: 200 OK

3.4 Obtener Estadísticas

Endpoint: GET {{base_url}}/progress/stats Headers: Authorization:
Bearer {{access_token}}

Caso 3.4.1: Estadísticas del usuario Respuesta esperada: 200 OK

```
{  
  "userId": "uuid",  
  "totalLessonsStarted": 5,  
  "totalLessonsCompleted": 3,  
  "averageAccuracy": 85.5,  
  "totalTimeSpent": 1800,  
  "unlockedLevels": ["Principiante", "Intermedio"]  
}
```

4. CONTENT - Contenido Educativo

4.1 Obtener Categorías de Edad (Público)

Endpoint: GET {{base_url}}/content/age-categories

Caso 4.1.1: Listar categorías Respuesta esperada: 200 OK

```
[  
  {  
    "id": "uuid",  
    "name": "1-3 años",  
    "minAge": 1,  
    "maxAge": 3,  
    "orderIndex": 1  
},
```

```
[  
  {  
    "id": "uuid",  
    "name": "3-5 años",  
    "minAge": 3,  
    "maxAge": 5,  
    "orderIndex": 2  
  }  
]
```

4.2 Obtener Niveles

Endpoint: GET {{base_url}}/content/levels **Headers:** Authorization: Bearer {{access_token}}

Caso 4.2.1: Listar niveles con estado de desbloqueo **Respuesta esperada:** 200 OK

```
[  
  {  
    "id": "uuid",  
    "name": "Principiante",  
    "level": "PRINCIPIANTE",  
    "orderIndex": 1,  
    "isUnlocked": true  
  },  
  {  
    "id": "uuid",  
    "name": "Intermedio",  
    "level": "INTERMEDIO",  
    "orderIndex": 2,  
    "isUnlocked": false  
  }  
]
```

4.3 Obtener Categorias de Contenido (Publico)

Endpoint: GET {{base_url}}/content/categories

Caso 4.3.1: Listar categorias **Respuesta esperada:** 200 OK

```
[  
  {  
    "id": "uuid",  
    "name": "Alfabeto",  
    "parent": null  
  }  
]
```

```
        "description": "Aprende el alfabeto en LSE",
        "orderIndex": 1
    }
]
```

4.4 Obtener Modulos

Endpoint: GET {{base_url}}/content/modules Headers: Authorization:
Bearer {{access_token}}

Caso 4.4.1: Listar todos los modulos Respuesta esperada: 200 OK

Caso 4.4.2: Filtrar por nivel Endpoint: GET {{base_url}}/content/modules?levelSectionId={{level}}
Respuesta esperada: 200 OK

4.5 Obtener Lecciones de un Modulo

Endpoint: GET {{base_url}}/content/modules/{{module_id}}/lessons
Headers: Authorization: Bearer {{access_token}}

Caso 4.5.1: Listar lecciones Respuesta esperada: 200 OK

```
[
{
    "id": "uuid",
    "moduleId": "uuid",
    "title": "Leccion 1: Saludos",
    "orderIndex": 1,
    "duration": 15,
    "isActive": true,
    "activitiesCount": 5
}
```

4.6 Obtener Leccion por ID

Endpoint: GET {{base_url}}/content/lessons/{{lesson_id}} Headers:
Authorization: Bearer {{access_token}}

Caso 4.6.1: Leccion encontrada Respuesta esperada: 200 OK

Caso 4.6.2: Error - Leccion no encontrada **Respuesta esperada:** 404
Not Found

4.7 Obtener Actividades de una Leccion

Endpoint: GET {{base_url}}/content/lessons/{{lesson_id}}/activities
Headers: Authorization: Bearer {{access_token}}

Caso 4.7.1: Listar actividades **Respuesta esperada:** 200 OK

```
[  
  {  
    "id": "uuid",  
    "lessonId": "uuid",  
    "title": "Practica: Saludo formal",  
    "activityType": "VIDEO_PRACTICE",  
    "orderIndex": 1,  
    "points": 10,  
    "isActive": true  
  }  
]
```

5. CLASSES - Gestión de Clases

5.1 Crear Clase (DOCENTE/ADMIN)

Endpoint: POST {{base_url}}/classes **Headers:** Authorization: Bearer {{access_token}} **Nota:** Requiere login como DOCENTE

Caso 5.1.1: Creacion exitosa

```
{  
  "name": "Clase de LSE Basico",  
  "description": "Clase para principiantes"  
}
```

Respuesta esperada: 201 Created

```
{  
  "id": "uuid",  
  "teacherId": "uuid",  
  "teacherName": "Maria Garcia",  
  "name": "Clase de LSE Basico",  
  "code": "ABC123",  
  "description": "Clase para principiantes",  
}
```

```
        "isActive": true,  
        "studentsCount": 0,  
        "createdAt": "..."  
    }  
}
```

Post-script:

```
if (pm.response.code === 201) {  
    var json = pm.response.json();  
    pm.environment.set("class_id", json.id);  
    pm.environment.set("class_code", json.code);  
}  
}
```

Caso 5.1.2: Error - Usuario PADRE intenta crear Clase Nota: Login como PADRE
Respuesta esperada: 403 Forbidden

Caso 5.1.3: Error - Nombre muy corto

```
{  
    "name": "A"  
}
```

Respuesta esperada: 400 Bad Request

5.2 Listar Clases que Enseno

Endpoint: GET {{base_url}}/classes/teaching **Headers:** Authorization: Bearer {{access_token}}

Caso 5.2.1: Listar clases del docente **Respuesta esperada:** 200 OK

```
{  
    "classes": [  
        {  
            "id": "uuid",  
            "name": "Clase de LSE Basico",  
            "code": "ABC123",  
            "studentsCount": 5  
        }  
    ],  
    "total": 1  
}
```

5.3 Actualizar Clase

Endpoint: PUT {{base_url}}/classes/{{class_id}} Headers: Authorization: Bearer {{access_token}}

Caso 5.3.1: Actualizacion exitosa

```
{  
  "name": "Clase de LSE Basico - Actualizada",  
  "description": "Descripcion actualizada"  
}
```

Respuesta esperada: 200 OK

Caso 5.3.2: Error - Clase de otro docente Respuesta esperada: 403 Forbidden

5.4 Eliminar Clase

Endpoint: DELETE {{base_url}}/classes/{{class_id}} Headers: Authorization: Bearer {{access_token}}

Caso 5.4.1: Eliminacion exitosa Respuesta esperada: 204 No Content

5.5 Crear Asignacion

Endpoint: POST {{base_url}}/classes/{{class_id}}/assignments Headers: Authorization: Bearer {{access_token}}

Caso 5.5.1: Asignacion exitosa

```
{  
  "lessonId": "{{lesson_id}}",  
  "title": "Tarea: Practica de saludos",  
  "description": "Completar la leccion de saludos",  
  "dueDate": "2024-12-31T23:59:59.000Z"  
}
```

Respuesta esperada: 201 Created

Caso 5.5.2: Asignacion sin fecha limite

```
{  
  "lessonId": "{{lesson_id}}",  
  "title": "Practica opcional"  
}
```

Respuesta esperada: 201 Created

5.6 Unirse a Clase (Estudiante)

Endpoint: POST {{base_url}}/classes/join **Headers:** Authorization: Bearer {{access_token}} **Nota:** Login como PADRE

Caso 5.6.1: Union exitosa

```
{  
  "code": "{{class_code}}"  
}
```

Respuesta esperada: 201 Created

```
{  
  "id": "uuid",  
  "classId": "uuid",  
  "studentId": "uuid",  
  "studentName": "Juan Perez",  
  "enrolledAt": "..."  
}
```

Caso 5.6.2: Error - Código invalido

```
{  
  "code": "INVALID"  
}
```

Respuesta esperada: 404 Not Found

Caso 5.6.3: Error - Ya inscrito

```
{  
  "code": "{{class_code}}"  
}
```

Respuesta esperada: 400 Bad Request

5.7 Salir de Clase

Endpoint: DELETE {{base_url}}/classes/{{class_id}}/leave **Headers:** Authorization: Bearer {{access_token}}

Caso 5.7.1: Salida exitosa Respuesta esperada: 204 No Content

5.8 Listar Clases Inscritas

Endpoint: GET {{base_url}}/classes/enrolled Headers: Authorization: Bearer {{access_token}}

Caso 5.8.1: Listar clases Respuesta esperada: 200 OK

```
{  
  "classes": [...],  
  "total": 2  
}
```

5.9 Obtener Clase por ID

Endpoint: GET {{base_url}}/classes/{{class_id}} Headers: Authorization: Bearer {{access_token}}

Caso 5.9.1: Clase encontrada Respuesta esperada: 200 OK

Caso 5.9.2: Error - Clase no encontrada Respuesta esperada: 404 Not Found

5.10 Listar Asignaciones de Clase

Endpoint: GET {{base_url}}/classes/{{class_id}}/assignments Headers: Authorization: Bearer {{access_token}}

Caso 5.10.1: Listar asignaciones Respuesta esperada: 200 OK

```
[  
  {  
    "id": "uuid",  
    "classId": "uuid",  
    "lessonId": "uuid",  
    "lessonTitle": "Leccion 1: Saludos",  
    "title": "Tarea: Practica de saludos",  
    "dueDate": "2024-12-31T23:59:59.000Z",  
    "isOverdue": false,  
    "createdAt": "..."  
  }  
]
```

Resumen de Casos de Prueba

Resultados por modulo:

- **Auth:** 20 casos (10 exitosos, 10 errores)
- **Children:** 12 casos (6 exitosos, 6 errores)
- **Progress:** 8 casos (6 exitosos, 2 errores)
- **Content:** 8 casos (8 exitosos, 0 errores)
- **Classes:** 15 casos (10 exitosos, 5 errores)

TOTAL: 63 casos (40 exitosos, 23 errores)

Colección Postman

Para importar estos casos en Postman:

1. Crear nueva colección “MinGO API Tests”
2. Configurar variables de entorno
3. Crear carpetas por modulo (Auth, Children, Progress, Content, Classes)
4. Agregar requests siguiendo los casos documentados
5. Agregar tests automatizados usando los post-scripts proporcionados

Script de Pre-request Global

```
// Verificar si tenemos token valido
if (!pm.environment.get("access_token")) {
    console.log("No hay token de acceso configurado");
}
```

Script de Test Global para Errores

```
// Verificar respuestas de error tienen formato correcto
if (pm.response.code >= 400) {
    pm.test("Error response has correct format", function() {
        var json = pm.response.json();
        pm.expect(json).to.have.property("statusCode");
        pm.expect(json).to.have.property("message");
    });
}
```

PRUEBAS UNITARIAS - MinGO API

Esta sección documenta todas las pruebas unitarias implementadas para validar la lógica de negocio de la API.

Herramientas Utilizadas

- **Jest (^30.0.0)** - Framework de testing principal

- **ts-jest** (^29.2.5) - Transformador TypeScript para Jest
- **@nestjs/testing** (^11.0.1) - Utilidades de testing para NestJS
- **uuid** (^13.0.0) - Generacion de UUIDs para mocks

Configuracion de Jest

```
{
  "rootDir": ".",
  "testRegex": ".*\\.spec\\.ts$",
  "roots": ["<rootDir>/src", "<rootDir>/test"],
  "transform": {
    "^.+\\.(t|j)s$": "ts-jest"
  },
  "transformIgnorePatterns": ["node_modules/(?!uuid)/"],
  "collectCoverageFrom": [
    "src/**/*.(t|j)s",
    "!src/**/*.module.ts",
    '!src/main.ts'
  ],
  "testEnvironment": "node"
}
```

Comandos de Ejecucion

```
# Ejecutar todas las pruebas
npm test

# Ejecutar pruebas en modo watch
npm run test:watch

# Ejecutar pruebas con cobertura
npm run test:cov

# Ejecutar pruebas e2e
npm run test:e2e
```

Estructura de Carpetas de Tests

```
test/
  --- unit/
    --- domain/
      |   --- entities/
      |   |   --- ChildInfo.spec.ts
      |   |   --- User.spec.ts
      |   --- value-objects/
```

```

|     +-+ Email.spec.ts
|     +-+ Password.spec.ts
|     +-+ UUID.spec.ts
+--- application/
|     +-+ use-cases/
|         +-+ auth/
|             |     +-+ ForgotPasswordUseCase.spec.ts
|             |     +-+ LoginUseCase.spec.ts
|             |     +-+ RegisterUseCase.spec.ts
|             |     +-+ ResetPasswordUseCase.spec.ts
|             |     +-+ VerifyEmailUseCase.spec.ts
|             +-+ child/
|                 +-+ CreateChildUseCase.spec.ts
|                 +-+ DeleteChildUseCase.spec.ts
|                 +-+ GetChildrenUseCase.spec.ts
|                 +-+ UpdateChildUseCase.spec.ts
+--- infrastructure/
|     +-+ services/
|         +-+ BcryptPasswordHasher.spec.ts
|         +-+ JwtTokenService.spec.ts
|         +-+ SMTPEmailService.spec.ts
+--- presentation/
    +-+ controllers/
        +-+ AuthController.spec.ts

```

1. PRUEBAS DE VALUE OBJECTS (Capa de Dominio)

1.1 UUID Value Object

Archivo: test/unit/domain/value-objects/UUID.spec.ts

Tests (12 total):

1. `create valid` - Crea UUID desde string valido
2. `create sin valor` - Genera UUID v4 automatico
3. `create invalido` - Lanza Error formato invalido
4. `create vacio` - Genera nuevo UUID
5. `generate` - Genera UUID v4 valido
6. `generate unico` - UUIDs diferentes
7. `from valido` - Crea desde string
8. `from invalido` - Lanza Error
9. `equals mismo` - Retorna true
10. `equals diferente` - Retorna false
11. `equals null` - Retorna false
12. `toString` - Retorna valor string

Ejemplo de test:

```
it('should create UUID from valid string', () => {
  const uuid = UUID.create('550e8400-e29b-41d4-a716-446655440000');
  expect(uuid.value).toBe('550e8400-e29b-41d4-a716-446655440000');
});
```

1.2 Email Value Object

Archivo: `test/unit/domain/value-objects/Email.spec.ts`

Tests (15 total):

1. `create valido` - Email normalizado
2. `create lowercase` - Convierte a minusculas
3. `create espacios` - Error formato invalido
4. `create vacio` - Error formato invalido
5. `create solo espacios` - Error formato invalido
6. `create sin @` - Error formato invalido
7. `create sin dominio` - Error formato invalido
8. `create sin local` - Error formato invalido
9. `create sin TLD` - Error formato invalido
10. `create formatos validos` - Acepta multiples formatos
11. `equals mismo` - Retorna true
12. `equals case insensitive` - Retorna true
13. `equals diferente` - Retorna false
14. `equals null` - Retorna false
15. `toString` - Retorna el email

Ejemplo de test:

```
it('should convert email to lowercase', () => {
  const email = Email.create('TEST@EXAMPLE.COM');
  expect(email.value).toBe('test@example.com');
});
```

1.3 Password Value Object

Archivo: `test/unit/domain/value-objects/Password.spec.ts`

Tests (10 total):

1. `create valido` - Password creada exitosamente
2. `create muy corta` - Error de validacion
3. `create sin mayusculas` - Error de validacion
4. `create sin minusculas` - Error de validacion
5. `create sin numeros` - Error de validacion

6. `create sin especiales` - Error de validacion
7. `createHashed` - Crea desde hash existente
8. `value` - Retorna el valor/hash
9. `equals mismo` - Retorna true
10. `equals diferente` - Retorna false

Requisitos de password: - Minimo 8 caracteres - Al menos una mayuscula - Al menos una minuscula - Al menos un numero - Al menos un caracter especial (!@#\$%^&*)

2. PRUEBAS DE ENTIDADES (Capa de Dominio)

2.1 User Entity

Archivo: `test/unit/domain/entities/User.spec.ts`

Tests (18 total):

1. `create valido` - Usuario con props correctas
2. `create timestamps` - `createdAt/updatedAt` asignados
3. `create defaults` - `isActive=true, emailVerified=false`
4. `generar token verificacion` - Token 64 chars, expira 24h
5. `verificar token valido` - `emailVerified=true`
6. `verificar token invalido` - Retorna false
7. `verificar token expirado` - Retorna false
8. `generar token reset` - Token expira en 1h
9. `validar reset token valido` - Retorna true
10. `validar reset token invalido` - Retorna false
11. `validar reset token expirado` - Retorna false
12. `reset password valido` - Password cambiada
13. `reset password invalido` - Retorna false
14. `deactivate` - `isActive=false`
15. `activate` - `isActive=true`
16. `updateProfile` - Nombre actualizado
17. `acceso hijo propio` - Retorna true
18. `acceso hijo ajeno` - Retorna false

Ejemplo de test:

```
it('should generate email verification token', () => {
  const user = createMockUser();
  const token = user.generateEmailVerificationToken();

  expect(token).toHaveLength(64);
  expect(user.emailVerificationToken).toBe(token);
  expect(user.emailVerificationExpires).toBeDefined();
});
```

2.2 ChildInfo Entity

Archivo: test/unit/domain/entities/ChildInfo.spec.ts

Tests (22 total):

1. create valido - ChildInfo con props correctas
 2. create timestamps - createdAt asignado
 3. create nombre vacio - Error nombre vacio
 4. create nombre largo - Error excede 100 chars
 5. create edad < 1 - Error edad invalida
 6. create edad > 18 - Error edad invalida
 7. create sin discapacidad - Acepta undefined
 8. reconstitute - Entidad reconstituida
 9. calculo edad - Edad correcta
 10. edad antes cumpleanos - Calculo correcto
 11. categoria 1-3 - Retorna "1-3 años"
 12. categoria 3-5 - Retorna "3-5 años"
 13. categoria 5-8 - Retorna "5-8 años"
 14. categoria 8-12 - Retorna "8-12 años"
 15. categoria fuera rango - Retorna "Fuera de rango"
 16. update nombre - Nombre cambiado
 17. update birthDate - Fecha cambiada
 18. update discapacidad - Tipo cambiado
 19. update notas - Notas cambiadas
 20. update avatar - URL cambiada
 21. update nombre invalido - Lanza Error
 22. update multiple - Todos actualizados
-

3. PRUEBAS DE USE CASES (Capa de Aplicacion)

3.1 Auth Use Cases

RegisterUseCase Archivo: test/unit/application/use-cases/auth/RegisterUseCase.spec.ts

Tests (7 total):

1. registro exitoso - AuthResponseDto con tokens
2. email existe - BusinessException
3. hash password - hash() llamado
4. guardar usuario - save() llamado
5. generar tokens - Tokens generados
6. enviar verificacion - sendVerificationEmail() llamado
7. token verificacion - Usuario con token

Mocks utilizados:

```
mockUserRepository = {
  findByEmail: jest.fn(),
  existsByEmail: jest.fn(),
  save: jest.fn(),
};

mockPasswordHasher = {
  hash: jest.fn(),
};

mockTokenService = {
  generateAccessToken: jest.fn(),
  generateRefreshToken: jest.fn(),
};

mockEmailService = {
  sendVerificationEmail: jest.fn(),
};
```

LoginUseCase Archivo: test/unit/application/use-cases/auth/LoginUseCase.spec.ts

Tests (6 total):

1. login exitoso - AuthResponseDto con tokens
 2. usuario no existe - UnauthorizedException
 3. password incorrecta - UnauthorizedException
 4. usuario inactivo - UnauthorizedException
 5. verificar password - compare() llamado
 6. generar tokens - Tokens generados
-

VerifyEmailUseCase Archivo: test/unit/application/use-cases/auth/VerifyEmailUseCase.spec.ts

Tests (6 total):

1. verificacion exitosa - MessageResponseDto success
 2. token invalido - BusinessException
 3. token expirado - BusinessException
 4. actualizar usuario - update() llamado
 5. enviar bienvenida - sendWelcomeEmail() llamado
 6. email verificado - emailVerified=true
-

ForgotPasswordUseCase Archivo: test/unit/application/use-cases/auth/ForgotPasswordUseCase

Tests (6 total):

1. solicitud exitosa - MessageResponseDto generico
 2. usuario no existe - Mismo mensaje (seguridad)
 3. usuario inactivo - Mismo mensaje (seguridad)
 4. generar token - Token reset creado
 5. enviar email - sendPasswordResetEmail() llamado
 6. guardar token - update() llamado
-

ResetPasswordUseCase Archivo: test/unit/application/use-cases/auth/ResetPasswordUseCase.spec

Tests (7 total):

1. reset exitoso - MessageResponseDto success
 2. hashear password - hash() llamado
 3. limpiar token - Token=undefined
 4. token no existe - BusinessException
 5. token expirado - BusinessException
 6. token no coincide - BusinessException
 7. password debil - Error validacion
-

3.2 Child Use Cases

CreateChildUseCase Archivo: test/unit/application/use-cases/child/CreateChildUseCase.spec

Tests (7 total):

1. crear como PADRE - ChildResponseDto creado
 2. padre no existe - NotFoundException
 3. no es PADRE - BusinessException
 4. limite FREE - BusinessException limite
 5. bajo limite - Creacion exitosa
 6. PREMIUM sin limite - No verifica limite
 7. parentId correcto - parentId=userId
-

UpdateChildUseCase Archivo: test/unit/application/use-cases/child/UpdateChildUseCase.spec

Tests (5 total):

1. actualizacion exitosa - ChildResponseDto actualizado
 2. hijo no encontrado - NotFoundException
 3. hijo otro parent - ForbiddenException
 4. multiples campos - Todos actualizados
 5. guardar cambios - save() llamado
-

DeleteChildUseCase Archivo: test/unit/application/use-cases/child/DeleteChildUseCase.spec.ts

Tests (5 total):

1. eliminacion exitosa - void sin error
 2. hijo no encontrado - NotFoundException
 3. hijo otro padre - ForbiddenException
 4. no elimina si no existe - delete() no llamado
 5. no elimina sin permiso - delete() no llamado
-

GetChildrenUseCase Archivo: test/unit/application/use-cases/child/GetChildrenUseCase.spec.ts

Tests (4 total):

1. listar hijos - Array de ChildResponseDto
 2. sin hijos - Array vacio
 3. multiples hijos - Todos del parent
 4. solo propios - findByParent() correcto
-

4. PRUEBAS DE SERVICIOS (Capa de Infraestructura)

4.1 BcryptPasswordHasher

Archivo: test/unit/infrastructure/services/BcryptPasswordHasher.spec.ts

Tests (5 total):

1. hash - String diferente al original
 2. hash diferente - Hashes diferentes (salt)
 3. compare valido - Retorna true
 4. compare invalido - Retorna false
 5. compare hash - Coincide con el hash
-

4.2 JwtTokenService

Archivo: test/unit/infrastructure/services/JwtTokenService.spec.ts

Tests (6 total):

1. generar access token - Token JWT valido
2. generar refresh token - Token JWT valido
3. verificar token valido - Payload decodificado
4. verificar token invalido - Lanza error
5. verificar refresh valido - Payload decodificado
6. verificar payload - userId, email, role presentes

4.3 SMTPEmailService

Archivo: test/unit/infrastructure/services/SMTPEmailService.spec.ts

Tests (9 total):

1. constructor - createTransport() llamado
2. verificacion exitosa - sendMail() llamado
3. verificacion URL - URL con token
4. verificacion error - No lanza error
5. reset exitoso - sendMail() llamado
6. reset URL - URL correcta
7. reset expiracion - Contiene “1 hora”
8. bienvenida exitosa - sendMail() llamado
9. bienvenida contenido - Texto correcto

Mock de nodemailer:

```
jest.mock('nodemailer');

mockTransporter = {
  sendMail: jest.fn().mockResolvedValue({ messageId: 'test-id' }),
};

(nodemailer.createTransport as jest.Mock).mockReturnValue(mockTransporter);
```

5. PRUEBAS DE CONTROLADORES (Capa de Presentación)

5.1 AuthController

Archivo: test/unit/presentation/controllers/AuthController.spec.ts

Tests (8 total):

1. register - Llama RegisterUseCase
2. login - Llama LoginUseCase
3. refreshToken - Llama RefreshTokenUseCase
4. verifyEmail - Llama VerifyEmailUseCase
5. resendVerification - Llama ResendVerificationUseCase
6. forgotPassword - Llama ForgotPasswordUseCase
7. resetPassword - Llama ResetPasswordUseCase
8. getProfile - Llama GetProfileUseCase

Configuracion del test:

```
const module: TestingModule = await Test.createTestingModule({
  controllers: [AuthController],
```

```

providers: [
  { provide: RegisterUseCase, useValue: mockRegisterUseCase },
  { provide: LoginUseCase, useValue: mockLoginUseCase },
  // ... otros use cases
],
}) .compile();

```

Resumen de Pruebas Unitarias

Resultados por categoria:

- **Value Objects:** 3 archivos, 37 tests - PASS
- **Entidades:** 2 archivos, 40 tests - PASS
- **Use Cases Auth:** 5 archivos, 35 tests - PASS
- **Use Cases Child:** 4 archivos, 22 tests - PASS
- **Servicios:** 3 archivos, 23 tests - PASS
- **Controladores:** 1 archivo, 8 tests - PASS

TOTAL: 18 archivos, 180 tests - PASS

Resultado de Ejecucion

```

Test Suites: 18 passed, 18 total
Tests:       180 passed, 180 total
Snapshots:   0 total
Time:        12.726 s

```

Cobertura de Codigo

Para generar reporte de cobertura:

```
npm run test:cov
```

Archivos excluidos de cobertura: - `src/**/*.module.ts` - Modulos de NestJS - `src/main.ts` - Punto de entrada

Buenas Practicas Aplicadas

1. **Aislamiento:** Cada test es independiente usando `beforeEach`
2. **Mocking:** Todas las dependencias externas son mockeadas
3. **Nomenclatura:** Tests describen claramente el comportamiento esperado
4. **Organizacion:** Tests agrupados por funcionalidad con `describe`
5. **Assertions:** Uso de matchers apropiados de Jest
6. **DRY:** Funciones helper para crear mocks reutilizables
7. **AAA Pattern:** Arrange, Act, Assert en cada test