Registro Diario de Avances

**Fecha:** 11/10/2025

**Autor:**

**Danluis Enrique Romani Mora**

**Eduardo Chaparro Huaman**

**Fabrizcio Argandoña Montalvo**

**Versión:** 1.3

# 1. Introducción

En este documento se muestra el avance de tareas respectivas al Sprint 3 con lo que también se detallan las historias de usuarios pertenecientes y el esfuerzo realizado para completar cada una de ellas, con evidencias del código utilizado, burndown chart y el análisis respectivo al rendimiento del equipo.

# 2. Lineamientos de Diseño

Desarrollar una tercera versión funcional de la aplicación que permita a los usuarios:

* Validez del test de Inteligencias Múltiples
* Realizar test de inteligencias múltiples
* Cálculo automático del puntaje
* Resultado del tipo de inteligencia predominante

# 3. Historias de Usuario y Tareas

| Backlog ID | Historia de Usuario | Tiempo Estimado | Fecha de Inicio | Fecha de Finalización |
| --- | --- | --- | --- | --- |
| HU-2.6 | Resultado del tipo de inteligencia predominante | 2 | 17/10/25 |  |
| HU-2.7 | Visualización de resultados | 5 |  |  |
| HU-2.8 | Almacenamiento de resultados | 6 |  |  |
| HU-2.9 | Acceso a histórico de resultados | 5 |  |  |
| HU-4.1 | Visualización de resultados de inteligencias múltiples | 12 |  |  |
| HU-P01 | Integración del backend con el frontend | 15 |  |  |

# 4. Evidencias de Código

## 4.1 Historia de Usuario P01: Integración del backend con el frontend

Archivo /controller/AdminController.java

@RestController

@RequestMapping("/api/admin")

public class AdminController {

@Autowired

private AdminService adminService;

// ==================== ADMINISTRADORES ====================

@PostMapping("/administradores")

public ResponseEntity<AdministradorDTO> crearAdministrador(@RequestBody CrearAdministradorRequest request) {

return ResponseEntity.ok(adminService.crearAdministrador(request));

}

@PutMapping("/administradores/{id}")

public ResponseEntity<AdministradorDTO> actualizarAdministrador(@PathVariable Integer id, @RequestBody CrearAdministradorRequest request) {

return ResponseEntity.ok(adminService.actualizarAdministrador(id, request));

}

@DeleteMapping("/administradores/{id}")

public ResponseEntity<Void> eliminarAdministrador(@PathVariable Integer id) {

adminService.eliminarAdministrador(id);

return ResponseEntity.ok().build();

}

@GetMapping("/administradores")

public ResponseEntity<List<AdministradorDTO>> listarAdministradores() {

return ResponseEntity.ok(adminService.listarAdministradores());

}

// Docentes

@PostMapping("/docentes")

public ResponseEntity<DocenteDTO> crearDocente(@RequestBody CrearDocenteRequest request) {

return ResponseEntity.ok(adminService.crearDocente(request));

}

@PutMapping("/docentes/{id}")

public ResponseEntity<DocenteDTO> actualizarDocente(@PathVariable Integer id, @RequestBody CrearDocenteRequest request) {

return ResponseEntity.ok(adminService.actualizarDocente(id, request));

}

@DeleteMapping("/docentes/{id}")

public ResponseEntity<Void> eliminarDocente(@PathVariable Integer id) {

adminService.eliminarDocente(id);

return ResponseEntity.ok().build();

}

@GetMapping("/docentes")

public ResponseEntity<List<DocenteDTO>> listarDocentes() {

return ResponseEntity.ok(adminService.listarDocentes());

}

// Alumnos

@PostMapping("/alumnos")

public ResponseEntity<AlumnoDTO> crearAlumno(@RequestBody CrearAlumnoRequest request) {

return ResponseEntity.ok(adminService.crearAlumno(request));

}

@PutMapping("/alumnos/{id}")

public ResponseEntity<AlumnoDTO> actualizarAlumno(@PathVariable Integer id, @RequestBody CrearAlumnoRequest request) {

return ResponseEntity.ok(adminService.actualizarAlumno(id, request));

}

@DeleteMapping("/alumnos/{id}")

public ResponseEntity<Void> eliminarAlumno(@PathVariable Integer id) {

adminService.eliminarAlumno(id);

return ResponseEntity.ok().build();

}

@GetMapping("/alumnos")

public ResponseEntity<List<AlumnoDTO>> listarAlumnos() {

return ResponseEntity.ok(adminService.listarAlumnos());

}

// Logs

@GetMapping("/logs")

public ResponseEntity<List<LogAccesoDTO>> listarLogs() {

return ResponseEntity.ok(adminService.listarLogs());

}

}

Archivo /controller/AlumnoController.java

@RestController

@RequestMapping("/api/alumno")

public class AlumnoController {

@Autowired

private TestService testService;

@GetMapping("/{idAlumno}/resultados/ultimo")

public ResponseEntity<List<ResultadoDTO>> obtenerUltimoResultado(@PathVariable Integer idAlumno) {

return ResponseEntity.ok(testService.obtenerUltimoResultado(idAlumno));

}

@GetMapping("/{idAlumno}/resultados/historial")

public ResponseEntity<List<ResultadoDTO>> obtenerHistorialResultados(@PathVariable Integer idAlumno) {

return ResponseEntity.ok(testService.obtenerHistorialResultados(idAlumno));

}

}

Archivo /controller/AuthController.java

@RestController

@RequestMapping("/api/auth")

public class AuthController {

@Autowired

private AuthService authService;

@PostMapping("/login/admin")

public ResponseEntity<?> loginAdmin(@RequestBody LoginRequest request) {

AdministradorDTO admin = authService.loginAdmin(request);

if (admin != null) {

return ResponseEntity.ok(admin);

}

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Credenciales inválidas");

}

@PostMapping("/login/docente")

public ResponseEntity<?> loginDocente(@RequestBody LoginRequest request) {

DocenteDTO docente = authService.loginDocente(request);

if (docente != null) {

return ResponseEntity.ok(docente);

}

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Credenciales inválidas");

}

@PostMapping("/login/alumno")

public ResponseEntity<?> loginAlumno(@RequestBody LoginRequest request) {

AlumnoDTO alumno = authService.loginAlumno(request);

if (alumno != null) {

return ResponseEntity.ok(alumno);

}

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Credenciales inválidas");

}

}

Archivo /controller/DodenteController.java

@RestController

@RequestMapping("/api/docente")

public class DocenteController {

@Autowired

private TestService testService;

@GetMapping("/alumnos/{idAlumno}/resultados")

public ResponseEntity<List<ResultadoDTO>> obtenerResultadosAlumno(@PathVariable Integer idAlumno) {

return ResponseEntity.ok(testService.obtenerHistorialResultados(idAlumno));

}

}

Archivo /controller/CursoController.java

@RestController

@RequestMapping("/api/cursos")

public class CursoController {

@Autowired

private CursoService cursoService;

// Crear curso

@PostMapping

public ResponseEntity<CursoDTO> crearCurso(@RequestBody CrearCursoRequest request) {

return ResponseEntity.ok(cursoService.crearCurso(request));

}

// Actualizar curso

@PutMapping("/{id}")

public ResponseEntity<CursoDTO> actualizarCurso(@PathVariable Integer id, @RequestBody CrearCursoRequest request) {

return ResponseEntity.ok(cursoService.actualizarCurso(id, request));

}

// Eliminar curso

@DeleteMapping("/{id}")

public ResponseEntity<Void> eliminarCurso(@PathVariable Integer id) {

cursoService.eliminarCurso(id);

return ResponseEntity.ok().build();

}

// Listar cursos

@GetMapping

public ResponseEntity<List<CursoDTO>> listarCursos() {

return ResponseEntity.ok(cursoService.listarCursos());

}

// Listar cursos por docente

@GetMapping("/docente/{idDocente}")

public ResponseEntity<List<CursoDTO>> listarPorDocente(@PathVariable Integer idDocente) {

return ResponseEntity.ok(cursoService.listarCursosPorDocente(idDocente));

}

// Matricular alumno en curso

@PostMapping("/matricular")

public ResponseEntity<Void> matricularAlumno(@RequestBody MatricularAlumnoRequest request) {

cursoService.matricularAlumno(request);

return ResponseEntity.ok().build();

}

// Listar alumnos de un curso

@GetMapping("/{idCurso}/alumnos")

public ResponseEntity<List<String>> listarAlumnosPorCurso(@PathVariable Integer idCurso) {

return ResponseEntity.ok(cursoService.listarAlumnosPorCurso(idCurso));

}

}

Archivo /controller/TestController.java

@RestController

@RequestMapping("/api/test")

public class TestController {

@Autowired

private TestService testService;

@GetMapping("/preguntas")

public ResponseEntity<List<PreguntaDTO>> obtenerPreguntas() {

return ResponseEntity.ok(testService.obtenerPreguntas());

}

@PostMapping("/completar")

public ResponseEntity<String> completarTest(@RequestBody CompletarTestRequest request) {

testService.completarTest(request);

return ResponseEntity.ok("Test completado exitosamente");

}

}

Archivo /model/Administrador.java

@Entity

@Table(name = "administradores")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Administrador {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_admin")

private Integer idAdmin;

@Column(name = "nombre\_usuario", nullable = false, unique = true, length = 100)

private String nombreUsuario;

@Column(name = "contrasena", nullable = false)

private String contrasena;

}

Archivo /model/Alumno.java

@Entity

@Table(name = "alumnos")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Alumno {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_alumno")

private Integer idAlumno;

@Column(name = "nombre\_usuario", nullable = false, unique = true)

private String nombreUsuario;

@Column(name = "contrasena", nullable = false)

private String contrasena;

}

Archivo /model/Curso.java

@Entity

@Table(name = "cursos")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Curso {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_curso")

private Integer idCurso;

@Column(name = "nombre\_curso", nullable = false)

private String nombreCurso;

@Column(name = "descripcion")

private String descripcion;

// Relación con docente

@ManyToOne

@JoinColumn(name = "id\_docente\_fk", nullable = false)

private Docente docente;

}

Archivo /model/Docente.java

@Entity

@Table(name = "docentes")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Docente {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_docente")

private Integer idDocente;

@Column(name = "nombre\_usuario", nullable = false, unique = true)

private String nombreUsuario;

@Column(name = "contrasena", nullable = false)

private String contrasena;

}

Archivo /Service/AdminService.java

@Service

public class AdminService {

@Autowired

private DocenteRepository docenteRepository;

@Autowired

private AlumnoRepository alumnoRepository;

@Autowired

private LogAccesoRepository logAccesoRepository;

@Autowired

private BCryptPasswordEncoder bCryptPasswordEncoder;

@Autowired

private AdministradorRepository administradorRepository;

// Crear

public AdministradorDTO crearAdministrador(CrearAdministradorRequest request) {

Administrador admin = new Administrador();

admin.setNombreUsuario(request.getNombreUsuario());

admin.setContrasena(bCryptPasswordEncoder.encode(request.getContrasena()));

Administrador saved = administradorRepository.save(admin);

return new AdministradorDTO(saved.getIdAdmin(), saved.getNombreUsuario());

}

// Actualizar

public AdministradorDTO actualizarAdministrador(Integer id, CrearAdministradorRequest request) {

Administrador admin = administradorRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Administrador no encontrado"));

admin.setNombreUsuario(request.getNombreUsuario());

if (request.getContrasena() != null && !request.getContrasena().isEmpty()) {

admin.setContrasena(bCryptPasswordEncoder.encode(request.getContrasena()));

}

Administrador saved = administradorRepository.save(admin);

return new AdministradorDTO(saved.getIdAdmin(), saved.getNombreUsuario());

}

// Eliminar

public void eliminarAdministrador(Integer id) {

administradorRepository.deleteById(id);

}

// Listar

public List<AdministradorDTO> listarAdministradores() {

return administradorRepository.findAll()

.stream()

.map(a -> new AdministradorDTO(a.getIdAdmin(), a.getNombreUsuario()))

.collect(Collectors.toList());

}

// Docentes

public DocenteDTO crearDocente(CrearDocenteRequest request) {

Docente docente = new Docente();

docente.setNombreUsuario(request.getNombreUsuario());

docente.setContrasena(bCryptPasswordEncoder.encode(request.getContrasena()));

Docente saved = docenteRepository.save(docente);

return new DocenteDTO(saved.getIdDocente(), saved.getNombreUsuario());

}

public DocenteDTO actualizarDocente(Integer id, CrearDocenteRequest request) {

Docente docente = docenteRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Docente no encontrado"));

docente.setNombreUsuario(request.getNombreUsuario());

if (request.getContrasena() != null && !request.getContrasena().isEmpty()) {

docente.setContrasena(bCryptPasswordEncoder.encode(request.getContrasena()));

}

Docente saved = docenteRepository.save(docente);

return new DocenteDTO(saved.getIdDocente(), saved.getNombreUsuario());

}

public void eliminarDocente(Integer id) {

docenteRepository.deleteById(id);

}

public List<DocenteDTO> listarDocentes() {

return docenteRepository.findAll().stream()

.map(d -> new DocenteDTO(d.getIdDocente(), d.getNombreUsuario()))

.collect(Collectors.toList());

}

// Alumnos

public AlumnoDTO crearAlumno(CrearAlumnoRequest request) {

Alumno alumno = new Alumno();

alumno.setNombreUsuario(request.getNombreUsuario());

alumno.setContrasena(bCryptPasswordEncoder.encode(request.getContrasena()));

Alumno saved = alumnoRepository.save(alumno);

return new AlumnoDTO(saved.getIdAlumno(), saved.getNombreUsuario());

}

public AlumnoDTO actualizarAlumno(Integer id, CrearAlumnoRequest request) {

Alumno alumno = alumnoRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Alumno no encontrado"));

alumno.setNombreUsuario(request.getNombreUsuario());

if (request.getContrasena() != null && !request.getContrasena().isEmpty()) {

alumno.setContrasena(bCryptPasswordEncoder.encode(request.getContrasena()));

}

Alumno saved = alumnoRepository.save(alumno);

return new AlumnoDTO(saved.getIdAlumno(), saved.getNombreUsuario());

}

public void eliminarAlumno(Integer id) {

alumnoRepository.deleteById(id);

}

public List<AlumnoDTO> listarAlumnos() {

return alumnoRepository.findAll().stream()

.map(a -> new AlumnoDTO(a.getIdAlumno(), a.getNombreUsuario()))

.collect(Collectors.toList());

}

// Logs

public List<LogAccesoDTO> listarLogs() {

return logAccesoRepository.findAll().stream()

.map(log -> new LogAccesoDTO(

log.getIdLog(),

log.getIdUsuario(),

log.getTipoUsuario().name(),

log.getFechaHoraAcceso()

))

.collect(Collectors.toList());

}

}

Archivo /service/AuthServive.java

@Service

public class AuthService {

@Autowired

private AdministradorRepository administradorRepository;

@Autowired

private DocenteRepository docenteRepository;

@Autowired

private AlumnoRepository alumnoRepository;

@Autowired

private LogAccesoRepository logAccesoRepository;

@Autowired

private BCryptPasswordEncoder bCryptPasswordEncoder;

public AdministradorDTO loginAdmin(LoginRequest request) {

Optional<Administrador> adminOpt = administradorRepository.findByNombreUsuario(request.getNombreUsuario());

if (adminOpt.isPresent() && bCryptPasswordEncoder.matches(request.getContrasena(), adminOpt.get().getContrasena())) {

Administrador admin = adminOpt.get();

// Registrar log de acceso

LogAcceso log = new LogAcceso();

log.setIdUsuario(admin.getIdAdmin());

log.setTipoUsuario(LogAcceso.TipoUsuario.admin);

logAccesoRepository.save(log);

return new AdministradorDTO(admin.getIdAdmin(), admin.getNombreUsuario());

}

return null;

}

public DocenteDTO loginDocente(LoginRequest request) {

Optional<Docente> docenteOpt = docenteRepository.findByNombreUsuario(request.getNombreUsuario());

if (docenteOpt.isPresent() && bCryptPasswordEncoder.matches(request.getContrasena(), docenteOpt.get().getContrasena())) {

Docente docente = docenteOpt.get();

// Registrar log de acceso

LogAcceso log = new LogAcceso();

log.setIdUsuario(docente.getIdDocente());

log.setTipoUsuario(LogAcceso.TipoUsuario.docente);

logAccesoRepository.save(log);

return new DocenteDTO(docente.getIdDocente(), docente.getNombreUsuario());

}

return null;

}

public AlumnoDTO loginAlumno(LoginRequest request) {

Optional<Alumno> alumnoOpt = alumnoRepository.findByNombreUsuario(request.getNombreUsuario());

if (alumnoOpt.isPresent() && bCryptPasswordEncoder.matches(request.getContrasena(), alumnoOpt.get().getContrasena())) {

Alumno alumno = alumnoOpt.get();

// Registrar log de acceso

LogAcceso log = new LogAcceso();

log.setIdUsuario(alumno.getIdAlumno());

log.setTipoUsuario(LogAcceso.TipoUsuario.alumno);

logAccesoRepository.save(log);

return new AlumnoDTO(alumno.getIdAlumno(), alumno.getNombreUsuario());

}

return null;

}

}

Archivo /service/CursoService.java

@Service

public class CursoService {

@Autowired

private CursoRepository cursoRepository;

@Autowired

private DocenteRepository docenteRepository;

@Autowired

private AlumnoRepository alumnoRepository;

@Autowired

private AlumnoCursoRepository alumnoCursoRepository;

// Crear curso

public CursoDTO crearCurso(CrearCursoRequest request) {

if (request.getIdDocente() == null) {

throw new IllegalArgumentException("El idDocente no puede ser nulo");

}

Docente docente = docenteRepository.findById(request.getIdDocente())

.orElseThrow(() -> new RuntimeException("Docente no encontrado con ID: " + request.getIdDocente()));

Curso curso = new Curso();

curso.setNombreCurso(request.getNombreCurso());

curso.setDescripcion(request.getDescripcion());

curso.setDocente(docente);

Curso saved = cursoRepository.save(curso);

return new CursoDTO(saved.getIdCurso(), saved.getNombreCurso(), saved.getDescripcion(), docente.getNombreUsuario());

}

// Actualizar curso

public CursoDTO actualizarCurso(Integer id, CrearCursoRequest request) {

Curso curso = cursoRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Curso no encontrado"));

curso.setNombreCurso(request.getNombreCurso());

curso.setDescripcion(request.getDescripcion());

if (request.getIdDocente() != null) {

Docente docente = docenteRepository.findById(request.getIdDocente())

.orElseThrow(() -> new RuntimeException("Docente no encontrado"));

curso.setDocente(docente);

}

Curso saved = cursoRepository.save(curso);

return new CursoDTO(saved.getIdCurso(), saved.getNombreCurso(), saved.getDescripcion(), saved.getDocente().getNombreUsuario());

}

// Eliminar curso

public void eliminarCurso(Integer id) {

cursoRepository.deleteById(id);

}

// Listar todos los cursos

public List<CursoDTO> listarCursos() {

return cursoRepository.findAll().stream()

.map(c -> new CursoDTO(c.getIdCurso(), c.getNombreCurso(), c.getDescripcion(), c.getDocente().getNombreUsuario()))

.collect(Collectors.toList());

}

// Listar cursos por docente

public List<CursoDTO> listarCursosPorDocente(Integer idDocente) {

Docente docente = docenteRepository.findById(idDocente)

.orElseThrow(() -> new RuntimeException("Docente no encontrado"));

return cursoRepository.findByDocente(docente).stream()

.map(c -> new CursoDTO(c.getIdCurso(), c.getNombreCurso(), c.getDescripcion(), docente.getNombreUsuario()))

.collect(Collectors.toList());

}

// Matricular alumno

public void matricularAlumno(MatricularAlumnoRequest request) {

Alumno alumno = alumnoRepository.findById(request.getIdAlumnoFk())

.orElseThrow(() -> new RuntimeException("Alumno no encontrado"));

Curso curso = cursoRepository.findById(request.getIdCursoFk())

.orElseThrow(() -> new RuntimeException("Curso no encontrado"));

AlumnoCurso matricula = new AlumnoCurso(alumno, curso, null);

alumnoCursoRepository.save(matricula);

}

// Listar alumnos por curso

public List<String> listarAlumnosPorCurso(Integer idCurso) {

Curso curso = cursoRepository.findById(idCurso)

.orElseThrow(() -> new RuntimeException("Curso no encontrado"));

return alumnoCursoRepository.findByCurso(curso).stream()

.map(ac -> ac.getAlumno().getNombreUsuario())

.collect(Collectors.toList());

}

}

Archivo /service/TestService.java

@Service

public class TestService {

@Autowired

private PreguntaTestRepository preguntaTestRepository;

@Autowired

private AlumnoRepository alumnoRepository;

@Autowired

private IntentoTestRepository intentoTestRepository;

@Autowired

private RespuestaAlumnoRepository respuestaAlumnoRepository;

@Autowired

private ResultadoTestRepository resultadoTestRepository;

@Autowired

private TipoInteligenciaRepository tipoInteligenciaRepository;

public List<PreguntaDTO> obtenerPreguntas() {

return preguntaTestRepository.findAll().stream()

.map(p -> new PreguntaDTO(

p.getIdPregunta(),

p.getTipoInteligencia().getIdInteligencia(),

p.getTipoInteligencia().getNombre(),

p.getTextoPregunta()

))

.collect(Collectors.toList());

}

@Transactional

public void completarTest(CompletarTestRequest request) {

// Obtener alumno

Alumno alumno = alumnoRepository.findById(request.getIdAlumno())

.orElseThrow(() -> new RuntimeException("Alumno no encontrado"));

// Crear intento de test

IntentoTest intento = new IntentoTest();

intento.setAlumno(alumno);

IntentoTest intentoSaved = intentoTestRepository.save(intento);

// Guardar respuestas individuales

for (CompletarTestRequest.RespuestaRequest respuestaRequest : request.getRespuestas()) {

PreguntaTest pregunta = preguntaTestRepository.findById(respuestaRequest.getIdPregunta())

.orElseThrow(() -> new RuntimeException("Pregunta no encontrada"));

RespuestaAlumno respuesta = new RespuestaAlumno();

respuesta.setIntentoTest(intentoSaved);

respuesta.setPreguntaTest(pregunta);

respuesta.setPuntaje(respuestaRequest.getPuntaje().byteValue());

respuestaAlumnoRepository.save(respuesta);

}

// Calcular puntajes por tipo de inteligencia

Map<Integer, Float> puntajesPorInteligencia = calcularPuntajes(request.getRespuestas());

// Guardar resultados

for (Map.Entry<Integer, Float> entry : puntajesPorInteligencia.entrySet()) {

TipoInteligencia tipoInteligencia = tipoInteligenciaRepository.findById(entry.getKey())

.orElseThrow(() -> new RuntimeException("Tipo de inteligencia no encontrado"));

ResultadoTest resultado = new ResultadoTest();

resultado.setIntentoTest(intentoSaved);

resultado.setTipoInteligencia(tipoInteligencia);

resultado.setPuntajeCalculado(entry.getValue());

resultadoTestRepository.save(resultado);

}

}

private Map<Integer, Float> calcularPuntajes(List<CompletarTestRequest.RespuestaRequest> respuestas) {

Map<Integer, List<Integer>> puntajesPorInteligencia = new HashMap<>();

for (CompletarTestRequest.RespuestaRequest respuesta : respuestas) {

PreguntaTest pregunta = preguntaTestRepository.findById(respuesta.getIdPregunta())

.orElseThrow(() -> new RuntimeException("Pregunta no encontrada"));

Integer idInteligencia = pregunta.getTipoInteligencia().getIdInteligencia();

puntajesPorInteligencia.computeIfAbsent(idInteligencia, k -> new java.util.ArrayList<>())

.add(respuesta.getPuntaje());

}

Map<Integer, Float> promedios = new HashMap<>();

for (Map.Entry<Integer, List<Integer>> entry : puntajesPorInteligencia.entrySet()) {

double promedio = entry.getValue().stream()

.mapToInt(Integer::intValue)

.average()

.orElse(0.0);

promedios.put(entry.getKey(), (float) promedio);

}

return promedios;

}

public List<ResultadoDTO> obtenerUltimoResultado(Integer idAlumno) {

List<IntentoTest> intentos = intentoTestRepository.findByAlumno\_IdAlumnoOrderByFechaRealizacionDesc(idAlumno);

if (intentos.isEmpty()) {

return List.of();

}

IntentoTest ultimoIntento = intentos.get(0);

List<ResultadoTest> resultados = resultadoTestRepository.findByIntentoId(ultimoIntento.getIdIntento());

return resultados.stream()

.map(r -> new ResultadoDTO(

r.getIdResultado(),

r.getIntentoTest().getIdIntento(),

r.getIntentoTest().getFechaRealizacion(),

r.getTipoInteligencia().getIdInteligencia(),

r.getTipoInteligencia().getNombre(),

r.getPuntajeCalculado()

))

.collect(Collectors.toList());

}

public List<ResultadoDTO> obtenerHistorialResultados(Integer idAlumno) {

List<ResultadoTest> resultados = resultadoTestRepository.findByAlumnoId(idAlumno);

return resultados.stream()

.map(r -> new ResultadoDTO(

r.getIdResultado(),

r.getIntentoTest().getIdIntento(),

r.getIntentoTest().getFechaRealizacion(),

r.getTipoInteligencia().getIdInteligencia(),

r.getTipoInteligencia().getNombre(),

r.getPuntajeCalculado()

))

.collect(Collectors.toList());

}

}

## 4.2 Historia de Usuario 2.6: Resultado del tipo de inteligencia predominante.

Archivo

## 4.3 Historia de Usuario 2.7: Visualización de resultados.

Archivo

## 4.4 Historia de Usuario 2.8: Almacenamiento de resultados.

Archivo

## 

## 4.5 Historia de Usuario 2.9: Acceso al histórico de resultados

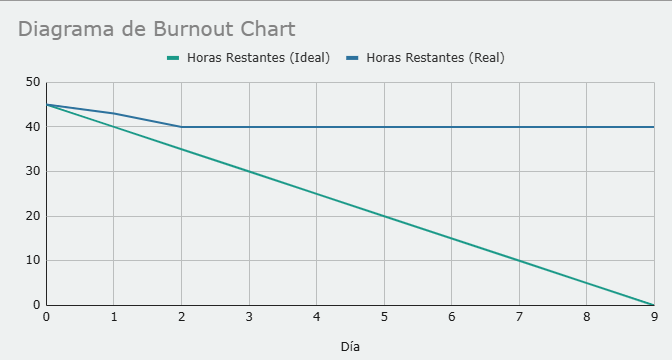
Archivo

## 4.5 Historia de Usuario 4.1:

Archivo

# 5. Burndown Chart

| Backlog ID | Historia de Usuario | Tiempo Estimado | 18/10/2025 | 20/10/2025 | 21/10/2025 | 22/10/2025 | 23/10/2025 | 24/10/2025 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Dia1** | **Dia2** | **Dia3** | **Dia4** | **Dia5** | **Dia6** |
| HU- P01 | Integración del backend con el frontend | 15 | 5 | 15 | 15 | 15 | 15 | 15 |
| HU-2.6 | Resultado del tipo de inteligencia predominante | 2 |  |  |  |  |  |  |
| HU-2.7 | Visualización de resultados | 5 |  |  |  |  |  |  |
| HU-2.8 | Almacenamiento de resultados | 6 |  |  |  |  |  |  |
| HU-2.9 | Acceso a histórico de resultados | 5 |  |  |  |  |  |  |
| HU-4.1 | Visualización de resultados de inteligencias múltiples | 12 |  |  |  |  |  |  |
| Tiempo de trabajo | | 45 | 40 | ? | ? | ? | ? | ? |



# 6. Análisis del Sprint

* No hay avance real entre las historias de usuario propuestas.
* Todo el esfuerzo esta enfocado en la etapa de integración del backend con el frontend.
* Hay avances respecto a la integración de las clases y de los endpoints que quedan pendientes.
* Hubo cambios relevantes en los módulos de model, controller y service
* Todo el grupo esta comprometido en la solución de este problema que surgió

# 7. Conclusiones

Nos encontramos con un problema en la integración respecto al funcionamiento de los controladores, para la integración con el frontend con lo que tomamos como un checkpoint el la ultima tarea del anterior spring para tener un proceso de desarrollo bastante limpio.

Respecto a lo que se esta refactorizando tenemos los modelos generados asi como los controladores principales del docente, alumno y administrador. Por ultimo se esta modificando los servicios que afectaban en el manejo de la información enviada por la base de datos.

Todo el equipo esta comprometido a cumplir con las tareas designadas, con ello con lo que concluimos que no se termino el spring con lo que estamos obligados a iterar nuevamente el spring hasta completar satisfactoriamente con las historias de usuario

Entre algunas medidas que tomamos esta en incrementar el tiempo de desarrollo diario pasando de unas 5 horas por persona a entre 7 y 11 horas por persona, asi como tambien decidimos dejar de tomar el domingo como dia de descanso para poder cumplir asi con los objetivos propuestos.