

# EcuAsist 2026

## Sistema de Control de Asistencia Escolar

<b>Version:</b>	2026.02
<b>Arquitectura:</b>	PHP MVC + PDO + MySQL
<b>Documento:</b>	Codigo Fuente Completo + Indice
<b>Generado:</b>	2026-02-20

# INDICE DE CONTENIDOS

## 1. ESTRUCTURA DE CARPETAS

## 2. CONFIG (2 archivos)

2.1 config/config.php

2.2 config/database.php

## 3. PUBLIC (2 archivos)

3.1 public/index.php

3.2 public/.htaccess

## 4. CONTROLLERS (18 archivos)

4.1 AcademicController.php

4.2 AssignmentController.php

4.3 AttendanceController.php

4.4 AuthController.php

4.5 BackupController.php

4.6 DashboardController.php

4.7 ExportController.php

4.8 InstitutionController.php

4.9 JustificationController.php

4.10 NotificationController.php

4.11 ProfileController.php

4.12 ReportController.php

4.13 RepresentativeController.php

4.14 ScheduleController.php

4.15 SearchController.php

4.16 StatsController.php

4.17 TutorController.php

4.18 UserController.php

## 5. MODELS (14 archivos)

5.1 Attendance.php

5.2 ClassSchedule.php

5.3 Course.php

5.4 Institution.php

5.5 InstitutionShift.php

5.6 Justification.php

5.7 Notification.php

5.8 Representative.php

5.9 Role.php

5.10 SchoolYear.php

5.11	Shift.php
5.12	Subject.php
5.13	TeacherAssignment.php
5.14	User.php
<b>6.</b>	<b>HELPERS (4 archivos)</b>
6.1	Backup.php
6.2	Logger.php
6.3	Mailer.php
6.4	Security.php

# 1. ESTRUCTURA DE CARPETAS

```
ArchImp/
backups/
config/
config.php
database.php
controllers/
AcademicController.php AssignmentController.php
AttendanceController.php AuthController.php
BackupController.php DashboardController.php
ExportController.php InstitutionController.php
JustificationController.php NotificationController.php
ProfileController.php ReportController.php
RepresentativeController.php ScheduleController.php
SearchController.php StatsController.php
TutorController.php UserController.php
helpers/
Backup.php Logger.php Mailer.php Security.php
models/
Attendance.php ClassSchedule.php Course.php
Institution.php InstitutionShift.php Justification.php
Notification.php Representative.php Role.php
SchoolYear.php Shift.php Subject.php
TeacherAssignment.php User.php
public/
.htaccess css/global.css index.php
uploads/institution/
vendor/ [Composer: TCPDF, PhpSpreadsheet]
views/
academic/ assignments/ attendance/ auth/
backup/ dashboard/ institution/ justifications/
notifications/ partials/ profile/ reports/
representatives/ schedule/ stats/ tutor/ users/
```

## 2. MODULO CONFIG

### config/config.php

PHP

```
1 <?php
2 // Zona horaria Ecuador
3 date_default_timezone_set('America/Guayaquil');
4
5 // Definir constantes ANTES de todo
6 define('BASE_PATH', __DIR__ . '/../');
7 define('BASE_URL', 'http://localhost/ecuasistencia2026');
8 define('EDIT_ATTENDANCE_HOURS', 48);
9
10 // Configuración de sesión ANTES de session_start()
11 ini_set('session.gc_maxlifetime', 86400); // 24 horas
12 session_set_cookie_params([
13 'lifetime' => 86400, // 24 horas
14 'path' => '/',
15 'domain' => '',
16 'secure' => false, // true si usas HTTPS
17 'httponly' => true, // Protección XSS
18 'samesite' => 'Lax' // Protección CSRF
19 ]);
20
21 // Iniciar sesión DESPUÉS de configurar
22 session_start();
23
24 // Timeout de inactividad: 30 minutos
25 $inactive_timeout = 1800; // 30 minutos en segundos
26 if (isset($_SESSION['last_activity']) &&
27 (time() - $_SESSION['last_activity'] > $inactive_timeout)) {
28 session_unset();
29 session_destroy();
30 header('Location: ' . BASE_URL . '/public/?action=login&timeout=1');
31 exit;
32 }
33 $_SESSION['last_activity'] = time();
34
35 // SMTP Config
36 define('SMTP_HOST', 'smtp.gmail.com');
37 define('SMTP_PORT', 587);
38 define('SMTP_USER', 'viktorengel@gmail.com');
39 define('SMTP_PASS', 'Orktvi.5/*83gM');
40 define('SMTP_FROM', 'noreply@ecuasist.edu.ec');
41 define('SMTP_NAME', 'EcuAsist2026');
42
43 require_once BASE_PATH . '/config/database.php';
44 require_once BASE_PATH . '/helpers/Security.php';
```

### config/database.php

PHP

```

1 <?php
2 class Database {
3 private $host = 'localhost';
4 private $db = 'ecuasistencia2026_db';
5 private $user = 'root';
6 private $pass = '';
7 private $charset = 'utf8mb4';
8 private $pdo;
9
10 public function connect() {
11 if ($this->pdo === null) {
12 $dsn = "mysql:host={$this->host};dbname={$this->db};charset={$this->charset}";
13 $options = [
14 PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
15 PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
16 PDO::ATTR_EMULATE_PREPARES => false,
17 ];
18
19 try {
20 $this->pdo = new PDO($dsn, $this->user, $this->pass, $options);
21 } catch (PDOException $e) {
22 die('Error de conexión: ' . $e->getMessage());
23 }
24 }
25 return $this->pdo;
26 }
27 }

```

## 3. MODULO PUBLIC

public/index.php

PHP

```
1 <?php
2
3 require_once '../config/config.php';
4
5 $action = $_GET['action'] ?? 'login';
6
7 switch ($action) {
8     case 'register':
9         require_once BASE_PATH . '/controllers/AuthController.php';
10        $auth = new AuthController();
11        $auth->register();
12        break;
13
14     case 'login':
15         require_once BASE_PATH . '/controllers/AuthController.php';
16        $auth = new AuthController();
17        $auth->login();
18        break;
19
20     case 'logout':
21         require_once BASE_PATH . '/controllers/AuthController.php';
22        $auth = new AuthController();
23        $auth->logout();
24        break;
25
26     case 'forgot':
27         require_once BASE_PATH . '/controllers/AuthController.php';
28        $auth = new AuthController();
29        $auth->forgotPassword();
30        break;
31
32     case 'reset':
33         require_once BASE_PATH . '/controllers/AuthController.php';
34        $auth = new AuthController();
35        $auth->resetPassword();
36        break;
37
38     case 'dashboard':
39         require_once BASE_PATH . '/controllers/DashboardController.php';
40        $dash = new DashboardController();
41        $dash->index();
42        break;
43
44     case 'users':
45         require_once BASE_PATH . '/controllers/UserController.php';
46        $userCtrl = new UserController();
47        $userCtrl->index();
48        break;
49
50     case 'assign_role':
51         require_once BASE_PATH . '/controllers/UserController.php';
52        $userCtrl = new UserController();
53        $userCtrl->assignRole();
54        break;
55
56     case 'attendance_register':
57         require_once BASE_PATH . '/controllers/AttendanceController.php';
58        $attCtrl = new AttendanceController();
59        $attCtrl->register();
60        break;
```

```

61
62 case 'get_students':
63 require_once BASE_PATH . '/controllers/AttendanceController.php';
64 $attCtrl = new AttendanceController();
65 $attCtrl->getStudents();
66 break;
67
68 case 'attendance_view':
69 require_once BASE_PATH . '/controllers/AttendanceController.php';
70 $attCtrl = new AttendanceController();
71 $attCtrl->view();
72 break;
73
74 case 'my_attendance':
75 require_once BASE_PATH . '/controllers/AttendanceController.php';
76 $attCtrl = new AttendanceController();
77 $attCtrl->myAttendance();
78 break;
79
80 case 'academic':
81 require_once BASE_PATH . '/controllers/AcademicController.php';
82 $acadCtrl = new AcademicController();
83 $acadCtrl->index();
84 break;
85
86 case 'create_course':
87 require_once BASE_PATH . '/controllers/AcademicController.php';
88 $acadCtrl = new AcademicController();
89 $acadCtrl->createCourse();
90 break;
91
92 case 'create_subject':
93 require_once BASE_PATH . '/controllers/AcademicController.php';
94 $acadCtrl = new AcademicController();
95 $acadCtrl->createSubject();
96 break;
97
98 case 'enroll_students':
99 require_once BASE_PATH . '/controllers/AcademicController.php';
100 $acadCtrl = new AcademicController();
101 $acadCtrl->enrollStudents();
102 break;
103
104 case 'view_course_students':
105 require_once BASE_PATH . '/controllers/AcademicController.php';
106 $acadCtrl = new AcademicController();
107 $acadCtrl->viewCourseStudents();
108 break;
109
110 case 'reports':
111 require_once BASE_PATH . '/controllers/ReportController.php';
112 $reportCtrl = new ReportController();
113 $reportCtrl->index();
114 break;
115
116 case 'generate_pdf':
117 require_once BASE_PATH . '/controllers/ReportController.php';
118 $reportCtrl = new ReportController();
119 $reportCtrl->generatePDF();
120 break;

```



```

121
122 case 'generate_excel':
123     require_once BASE_PATH . '/controllers/ReportController.php';
124     $reportCtrl = new ReportController();
125     $reportCtrl->generateExcel();
126     break;
127
128 case 'manage_representatives':
129     require_once BASE_PATH . '/controllers/RepresentativeController.php';
130     $repCtrl = new RepresentativeController();
131     $repCtrl->manageRepresentatives();
132     break;
133
134 case 'my_children':
135     require_once BASE_PATH . '/controllers/RepresentativeController.php';
136     $repCtrl = new RepresentativeController();
137     $repCtrl->myChildren();
138     break;
139
140 case 'child_attendance':
141     require_once BASE_PATH . '/controllers/RepresentativeController.php';
142     $repCtrl = new RepresentativeController();
143     $repCtrl->childAttendance();
144     break;
145
146 case 'assignments':
147     require_once BASE_PATH . '/controllers/AssignmentController.php';
148     $assignCtrl = new AssignmentController();
149     $assignCtrl->index();
150     break;
151
152 case 'create_assignment':
153     require_once BASE_PATH . '/controllers/AssignmentController.php';
154     $assignCtrl = new AssignmentController();
155     $assignCtrl->assign();
156     break;
157
158 case 'set_tutor':
159     require_once BASE_PATH . '/controllers/AssignmentController.php';
160     $assignCtrl = new AssignmentController();
161     $assignCtrl->setTutor();
162     break;
163
164 case 'remove_tutor':
165     require_once BASE_PATH . '/controllers/AssignmentController.php';
166     $controller = new AssignmentController();
167     $controller->removeTutor();
168     break;
169
170 case 'get_course_teachers':
171     require_once BASE_PATH . '/controllers/AssignmentController.php';
172     $controller = new AssignmentController();
173     $controller->getCourseTeachers();
174     break;
175
176 case 'remove_assignment':
177     require_once BASE_PATH . '/controllers/AssignmentController.php';
178     $assignCtrl = new AssignmentController();
179     $assignCtrl->remove();
180     break;

```

```

181
182 case 'view_course_assignments':
183     require_once BASE_PATH . '/controllers/AssignmentController.php';
184     $assignCtrl = new AssignmentController();
185     $assignCtrl->viewByCourse();
186     break;
187
188 case 'profile':
189     require_once BASE_PATH . '/controllers/ProfileController.php';
190     $profileCtrl = new ProfileController();
191     $profileCtrl->view();
192     break;
193
194 case 'edit_profile':
195     require_once BASE_PATH . '/controllers/ProfileController.php';
196     $profileCtrl = new ProfileController();
197     $profileCtrl->edit();
198     break;
199
200 case 'change_password':
201     require_once BASE_PATH . '/controllers/ProfileController.php';
202     $profileCtrl = new ProfileController();
203     $profileCtrl->changePassword();
204     break;
205
206 case 'search_students':
207     require_once BASE_PATH . '/controllers/SearchController.php';
208     $searchCtrl = new SearchController();
209     $searchCtrl->searchStudents();
210     break;
211
212 case 'export_student_list':
213     require_once BASE_PATH . '/controllers/ExportController.php';
214     $exportCtrl = new ExportController();
215     $exportCtrl->exportStudentList();
216     break;
217
218 case 'stats':
219     require_once BASE_PATH . '/controllers/StatsController.php';
220     $statsCtrl = new StatsController();
221     $statsCtrl->index();
222     break;
223
224 case 'attendance_calendar':
225     require_once BASE_PATH . '/controllers/AttendanceController.php';
226     $attCtrl = new AttendanceController();
227     $attCtrl->calendar();
228     break;
229
230 case 'submit_justification':
231     require_once BASE_PATH . '/controllers/JustificationController.php';
232     $justCtrl = new JustificationController();
233     $justCtrl->submit();
234     break;
235
236 case 'my_justifications':
237     require_once BASE_PATH . '/controllers/JustificationController.php';
238     $justCtrl = new JustificationController();
239     $justCtrl->myJustifications();
240     break;

```

```

241
242 case 'pending_justifications':
243     require_once BASE_PATH . '/controllers/JustificationController.php';
244     $justCtrl = new JustificationController();
245     $justCtrl->pending();
246     break;
247
248 case 'reviewed_justifications':
249     require_once BASE_PATH . '/controllers/JustificationController.php';
250     $controller = new JustificationController();
251     $controller->reviewed();
252     break;
253
254 case 'review_justification':
255     require_once BASE_PATH . '/controllers/JustificationController.php';
256     $justCtrl = new JustificationController();
257     $justCtrl->review();
258     break;
259
260 case 'backups':
261     require_once BASE_PATH . '/controllers/BackupController.php';
262     $backupCtrl = new BackupController();
263     $backupCtrl->index();
264     break;
265
266 case 'create_backup':
267     require_once BASE_PATH . '/controllers/BackupController.php';
268     $backupCtrl = new BackupController();
269     $backupCtrl->create();
270     break;
271
272 case 'download_backup':
273     require_once BASE_PATH . '/controllers/BackupController.php';
274     $backupCtrl = new BackupController();
275     $backupCtrl->download();
276     break;
277
278 case 'cleanup_backups':
279     require_once BASE_PATH . '/controllers/BackupController.php';
280     $backupCtrl = new BackupController();
281     $backupCtrl->cleanup();
282     break;
283
284 case 'remove_role':
285     require_once BASE_PATH . '/controllers/UserController.php';
286     $userCtrl = new UserController();
287     $userCtrl->removeRole();
288     break;
289
290 case 'get_course_subjects':
291     require_once BASE_PATH . '/controllers/AttendanceController.php';
292     $controller = new AttendanceController();
293     $controller->getCourseSubjects();
294     break;
295
296 case 'get_teacher_course_subjects':
297     require_once BASE_PATH . '/controllers/AttendanceController.php';
298     $controller = new AttendanceController();
299     $controller->getTeacherCourseSubjects();
300     break;

```

```

301
302 case 'schedules':
303     require_once BASE_PATH . '/controllers/ScheduleController.php';
304     $controller = new ScheduleController();
305     $controller->index();
306     break;
307
308 case 'manage_schedule':
309     require_once BASE_PATH . '/controllers/ScheduleController.php';
310     $controller = new ScheduleController();
311     $controller->manageCourse();
312     break;
313
314 case 'delete_schedule_class':
315     require_once BASE_PATH . '/controllers/ScheduleController.php';
316     $controller = new ScheduleController();
317     $controller->deleteClass();
318     break;
319
320 case 'get_schedule_info':
321     require_once BASE_PATH . '/controllers/AttendanceController.php';
322     $controller = new AttendanceController();
323     $controller->getScheduleInfo();
324     break;
325
326 case 'get_existing_attendance':
327     require_once BASE_PATH . '/controllers/AttendanceController.php';
328     $controller = new AttendanceController();
329     $controller->getExistingAttendance();
330     break;
331
332 case 'tutor_management':
333     require_once BASE_PATH . '/controllers/AssignmentController.php';
334     $controller = new AssignmentController();
335     $controller->tutorManagement();
336     break;
337
338 case 'check_course_tutor':
339     require_once BASE_PATH . '/controllers/AssignmentController.php';
340     $controller = new AssignmentController();
341     $controller->checkCourseTutor();
342     break;
343
344 case 'get_course_subjects_schedule':
345     require_once BASE_PATH . '/controllers/ScheduleController.php';
346     $controller = new ScheduleController();
347     $controller->getCourseSubjectsSchedule();
348     break;
349
350 case 'check_schedule_conflict':
351     require_once BASE_PATH . '/controllers/ScheduleController.php';
352     $controller = new ScheduleController();
353     $controller->checkScheduleConflict();
354     break;
355
356 case 'institution':
357     require_once BASE_PATH . '/controllers/InstitutionController.php';
358     $controller = new InstitutionController();
359     $controller->index();
360     break;

```

```

361
362 case 'update_institution':
363     require_once BASE_PATH . '/controllers/InstitutionController.php';
364     $controller = new InstitutionController();
365     $controller->update();
366     break;
367
368 case 'assign_institution_shift':
369     require_once BASE_PATH . '/controllers/InstitutionController.php';
370     $controller = new InstitutionController();
371     $controller->assignShift();
372     break;
373
374 case 'remove_institution_shift':
375     require_once BASE_PATH . '/controllers/InstitutionController.php';
376     $controller = new InstitutionController();
377     $controller->removeShift();
378     break;
379
380 case 'toggle_institution_shift':
381     require_once BASE_PATH . '/controllers/InstitutionController.php';
382     (new InstitutionController())->toggleShift();
383     break;
384
385 case 'create_user':
386     require_once BASE_PATH . '/controllers/UserController.php';
387     $controller = new UserController();
388     $controller->create();
389     break;
390
391 case 'edit_user':
392     require_once BASE_PATH . '/controllers/UserController.php';
393     $controller = new UserController();
394     $controller->edit();
395     break;
396
397 case 'delete_user':
398     require_once BASE_PATH . '/controllers/UserController.php';
399     $controller = new UserController();
400     $controller->delete();
401     break;
402
403 case 'create_school_year':
404     require_once BASE_PATH . '/controllers/AcademicController.php';
405     $controller = new AcademicController();
406     $controller->createSchoolYear();
407     break;
408
409 case 'edit_school_year':
410     require_once BASE_PATH . '/controllers/AcademicController.php';
411     $controller = new AcademicController();
412     $controller->editSchoolYear();
413     break;
414
415 case 'delete_school_year':
416     require_once BASE_PATH . '/controllers/AcademicController.php';
417     $controller = new AcademicController();
418     $controller->deleteSchoolYear();
419     break;
420

```

```

421 case 'activate_school_year':
422     require_once BASE_PATH . '/controllers/AcademicController.php';
423     $controller = new AcademicController();
424     $controller->activateSchoolYear();
425     break;
426
427 case 'deactivate_school_year':
428     require_once BASE_PATH . '/controllers/AcademicController.php';
429     $controller = new AcademicController();
430     $controller->deactivateSchoolYear();
431     break;
432
433 case 'edit_course':
434     require_once BASE_PATH . '/controllers/AcademicController.php';
435     $controller = new AcademicController();
436     $controller->editCourse();
437     break;
438
439 case 'delete_course':
440     require_once BASE_PATH . '/controllers/AcademicController.php';
441     $controller = new AcademicController();
442     $controller->deleteCourse();
443     break;
444
445 case 'edit_subject':
446     require_once BASE_PATH . '/controllers/AcademicController.php';
447     $controller = new AcademicController();
448     $controller->editSubject();
449     break;
450
451 case 'delete_subject':
452     require_once BASE_PATH . '/controllers/AcademicController.php';
453     $controller = new AcademicController();
454     $controller->deleteSubject();
455     break;
456
457 case 'unenroll_student':
458     require_once BASE_PATH . '/controllers/AcademicController.php';
459     $controller = new AcademicController();
460     $controller->unenrollStudent();
461     break;
462
463 case 'delete_backup':
464     require_once BASE_PATH . '/controllers/BackupController.php';
465     $controller = new BackupController();
466     $controller->delete();
467     break;
468
469 case 'remove_representative':
470     require_once BASE_PATH . '/controllers/RepresentativeController.php';
471     $controller = new RepresentativeController();
472     $controller->removeRelation();
473     break;
474
475 case 'notifications':
476     require_once BASE_PATH . '/controllers/NotificationController.php';
477     (new NotificationController())->index();
478     break;
479
480 case 'notifications_mark_read':

```

```

481 require_once BASE_PATH . '/controllers/NotificationController.php';
482 (new NotificationController())->markRead();
483 break;
484
485 case 'notifications_mark_all':
486 require_once BASE_PATH . '/controllers/NotificationController.php';
487 (new NotificationController())->markAllRead();
488 break;
489
490 case 'notifications_delete':
491 require_once BASE_PATH . '/controllers/NotificationController.php';
492 (new NotificationController())->delete();
493 break;
494
495 case 'notifications_delete_read':
496 require_once BASE_PATH . '/controllers/NotificationController.php';
497 (new NotificationController())->deleteRead();
498 break;
499
500 case 'notifications_unread_json':
501 require_once BASE_PATH . '/controllers/NotificationController.php';
502 (new NotificationController())->getUnread();
503 break;
504
505 case 'tutor_course_attendance':
506 require_once BASE_PATH . '/controllers/TutorController.php';
507 (new TutorController())->courseAttendance();
508 break;
509
510 case 'tutor_course_attendance_ajax':
511 require_once BASE_PATH . '/controllers/TutorController.php';
512 (new TutorController())->ajax();
513 break;
514
515 default:
516 header('Location: ?action=login');
517 }

```

## public/.htaccess

Apache

```

1 <IfModule mod_rewrite.c>
2 RewriteEngine On
3 RewriteBase /ecuasistencia2026/public/
4
5 RewriteCond %{REQUEST_FILENAME} !-f
6 RewriteCond %{REQUEST_FILENAME} !-d
7 RewriteRule ^(.*)$ index.php?action=$1 [QSA,L]
8 </IfModule>

```

## 4. MODULO CONTROLLERS (18 archivos)

### 4.1 — AcademicController.php

controllers/AcademicController.php

PHP

```
1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/Course.php';
4 require_once BASE_PATH . '/models/Subject.php';
5 require_once BASE_PATH . '/models/SchoolYear.php';
6 require_once BASE_PATH . '/models/Shift.php';
7 require_once BASE_PATH . '/models/User.php';
8
9 class AcademicController {
10 private $courseModel;
11 private $subjectModel;
12 private $schoolYearModel;
13 private $shiftModel;
14 private $userModel;
15
16 public function __construct() {
17 Security::requireLogin();
18 if (!Security::hasRole('autoridad')) {
19 die('Acceso denegado');
20 }
21
22 $db = new Database();
23 $this->courseModel = new Course($db);
24 $this->subjectModel = new Subject($db);
25 $this->schoolYearModel = new SchoolYear($db);
26 $this->shiftModel = new Shift($db);
27 $this->userModel = new User($db);
28 }
29
30 public function index() {
31 $courses = $this->courseModel->getAll();
32 $subjects = $this->subjectModel->getAll();
33 $schoolYears = $this->schoolYearModel->getAll();
34 $shifts = $this->shiftModel->getAll();
35
36 include BASE_PATH . '/views/academic/index.php';
37 }
38
39 public function createCourse() {
40 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
41 $activeYear = $this->schoolYearModel->getActive();
42
43 if (!$activeYear) {
44 header('Location: ?action=academic&error=no_active_year');
45 exit;
46 }
47
48 $data = [
49 ':institution_id' => $_SESSION['institution_id'],
50 ':school_year_id' => $activeYear['id'],
51 ':name' => html_entity_decode(Security::sanitize($_POST['name']), ENT_QUOTES, 'UTF-8'),
52 ':grade_level' => html_entity_decode(Security::sanitize($_POST['grade_level']), ENT_QUOTES, 'UTF-8'),
53 ':parallel' => Security::sanitize($_POST['parallel']),
54 ':shift_id' => (int)$_POST['shift_id']
55 ];
56
57 $this->courseModel->create($data);
58 header('Location: ?action=academic&course_success=1');
59 exit;
60 }
```



```

61 }
62
63 public function createSubject() {
64 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
65 $data = [
66 ':institution_id' => $_SESSION['institution_id'],
67 ':name' => Security::sanitize($_POST['name']),
68 ':code' => Security::sanitize($_POST['code'])
69 ];
70
71 $this->subjectModel->create($data);
72 header('Location: ?action=academic&subject_success=1');
73 exit;
74 }
75 }
76
77 // =====
78 // CRUD CURSOS
79 // =====
80
81 public function editCourse() {
82 $courseId = (int)$_GET['id'];
83 $course = $this->courseModel->findById($courseId);
84
85 if (!$course || $course['institution_id'] != $_SESSION['institution_id']) {
86 header('Location: ?action=academic&error=course_not_found');
87 exit;
88 }
89
90 if ($_SERVER['REQUEST_METHOD'] === 'GET') {
91 $shifts = $this->shiftModel->getAll();
92 include BASE_PATH . '/views/academic/course_edit.php';
93 return;
94 }
95
96 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
97 $data = [
98 'id' => $courseId,
99 'name' => html_entity_decode(Security::sanitize($_POST['name']), ENT_QUOTES, 'UTF-8'),
100 'grade_level' => html_entity_decode(Security::sanitize($_POST['grade_level']), ENT_QUOTES, 'UTF-8'),
101 'parallel' => Security::sanitize($_POST['parallel']),
102 'shift_id' => (int)$_POST['shift_id']
103 ];
104
105 if ($this->courseModel->update($data)) {
106 header('Location: ?action=academic&course_updated=1');
107 exit;
108 } else {
109 $errors[] = "Error al actualizar el curso";
110 $shifts = $this->shiftModel->getAll();
111 include BASE_PATH . '/views/academic/course_edit.php';
112 }
113 }
114 }
115
116 public function deleteCourse() {
117 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
118 $courseId = (int)$_POST['course_id'];
119 $course = $this->courseModel->findById($courseId);
120

```

```

121 if (!$course || $course['institution_id'] != $_SESSION['institution_id']) {
122 header('Location: ?action=academic&error=course_not_found');
123 exit;
124 }
125
126 // Verificar si tiene estudiantes matriculados
127 $students = $this->courseModel->getEnrolledStudents($courseId);
128 if (count($students) > 0) {
129 header('Location: ?action=academic&error=course_has_students');
130 exit;
131 }
132
133 // Verificar si tiene asignaciones docentes
134 $db = new Database();
135 $stmt = $db->connect()->prepare("SELECT COUNT(*) as count FROM teacher_assignments WHERE course_id = :id");
136 $stmt->execute([':id' => $courseId]);
137 $result = $stmt->fetch();
138
139 if ($result['count'] > 0) {
140 header('Location: ?action=academic&error=course_has_assignments');
141 exit;
142 }
143
144 if ($this->courseModel->delete($courseId)) {
145 header('Location: ?action=academic&course_deleted=1');
146 exit;
147 } else {
148 header('Location: ?action=academic&error=delete_failed');
149 exit;
150 }
151 }
152 }
153
154 // =====
155 // CRUD ASIGNATURAS
156 // =====
157
158 public function editSubject() {
159 $subjectId = (int)$_GET['id'];
160 $subject = $this->subjectModel->findById($subjectId);
161
162 if (!$subject || $subject['institution_id'] != $_SESSION['institution_id']) {
163 header('Location: ?action=academic&error=subject_not_found');
164 exit;
165 }
166
167 if ($_SERVER['REQUEST_METHOD'] === 'GET') {
168 include BASE_PATH . '/views/academic/subject_edit.php';
169 return;
170 }
171
172 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
173 $data = [
174 'id' => $subjectId,
175 'name' => Security::sanitize($_POST['name']),
176 'code' => Security::sanitize($_POST['code'])
177 ];
178
179 if ($this->subjectModel->update($data)) {
180 header('Location: ?action=academic&subject_updated=1');

```

```

181 exit;
182 } else {
183 $errors[] = "Error al actualizar la asignatura";
184 include BASE_PATH . '/views/academic/subject_edit.php';
185 }
186 }
187 }
188
189 public function deleteSubject() {
190 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
191 $subjectId = (int)$_POST['subject_id'];
192 $subject = $this->subjectModel->findById($subjectId);
193
194 if (!$subject || $subject['institution_id'] != $_SESSION['institution_id']) {
195 header('Location: ?action=academic&error=subject_not_found');
196 exit;
197 }
198
199 // Verificar si tiene asignaciones docentes
200 $db = new Database();
201 $stmt = $db->connect()->prepare("SELECT COUNT(*) as count FROM teacher_assignments WHERE subject_id = :id");
202 $stmt->execute([':id' => $subjectId]);
203 $result = $stmt->fetch();
204
205 if ($result['count'] > 0) {
206 header('Location: ?action=academic&error=subject_has_assignments');
207 exit;
208 }
209
210 if ($this->subjectModel->delete($subjectId)) {
211 header('Location: ?action=academic&subject_deleted=1');
212 exit;
213 } else {
214 header('Location: ?action=academic&error=delete_failed');
215 exit;
216 }
217 }
218 }
219
220 public function enrollStudents() {
221 $activeYear = $this->schoolYearModel->getActive();
222 $courses = $this->courseModel->getAll();
223 $availableStudents = $this->userModel->getStudentsNotEnrolled($activeYear['id']);
224 $allStudents = $this->userModel->getByRole('estudiante');
225
226 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
227 $courseId = (int)$_POST['course_id'];
228 $studentIds = $_POST['student_ids'] ?? [];
229
230 $enrolled = 0;
231 $errors = 0;
232
233 foreach ($studentIds as $studentId) {
234 if ($this->courseModel->enrollStudent($courseId, (int)$studentId, $activeYear['id'])) {
235 $enrolled++;
236 } else {
237 $errors++;
238 }
239 }
240

```

```

241 header('Location: ?action=enroll_students&enrolled=' . $enrolled . '&errors=' . $errors);
242 exit;
243 }
244
245 include BASE_PATH . '/views/academic/enroll.php';
246 }
247
248 public function unenrollStudent() {
249 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
250 $studentId = (int)$_POST['student_id'];
251 $activeYear = $this->schoolYearModel->getActive();
252
253 if (!$activeYear) {
254 header('Location: ?action=enroll_students&error=no_active_year');
255 exit;
256 }
257
258 // Verificar que el estudiante existe y está matriculado
259 $course = $this->userModel->getStudentCourse($studentId, $activeYear['id']);
260
261 if (!$course) {
262 header('Location: ?action=enroll_students&error=not_enrolled');
263 exit;
264 }
265
266 // Retirar estudiante
267 if ($this->courseModel->unenrollStudent($studentId, $activeYear['id'])) {
268 header('Location: ?action=enroll_students&unenrolled=1');
269 exit;
270 } else {
271 header('Location: ?action=enroll_students&error=unenroll_failed');
272 exit;
273 }
274 }
275 }
276
277 public function viewCourseStudents() {
278 $courseId = (int)($_GET['course_id'] ?? 0);
279
280 if (!$courseId) {
281 header('Location: ?action=academic');
282 exit;
283 }
284
285 $course = $this->courseModel->getAll();
286 $course = array_filter($course, fn($c) => $c['id'] == $courseId);
287 $course = reset($course);
288
289 $students = $this->courseModel->getEnrolledStudents($courseId);
290
291 include BASE_PATH . '/views/academic/course_students.php';
292 }
293
294 // =====
295 // CRUD AÑOS LECTIVOS
296 // =====
297
298 public function createSchoolYear() {
299 if ($_SERVER['REQUEST_METHOD'] === 'GET') {
300 include BASE_PATH . '/views/academic/school_year_create.php';

```

```

301 return;
302 }
303
304 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
305     $errors = [];
306
307     // Validaciones
308     if (empty($_POST['name'])) {
309         $errors[] = "El nombre es obligatorio";
310     }
311     if (empty($_POST['start_date'])) {
312         $errors[] = "La fecha de inicio es obligatoria";
313     }
314     if (empty($_POST['end_date'])) {
315         $errors[] = "La fecha de fin es obligatoria";
316     }
317
318     if (!empty($_POST['start_date']) && !empty($_POST['end_date'])) {
319         if (strtotime($_POST['end_date']) <= strtotime($_POST['start_date'])) {
320             $errors[] = "La fecha de fin debe ser posterior a la fecha de inicio";
321         }
322     }
323     // Verificar solapamiento de fechas
324     if ($this->schoolYearModel->checkOverlap($_POST['start_date'], $_POST['end_date'])) {
325         $errors[] = "Ya existe un año lectivo con fechas que se solapan";
326     }
327 }
328
329 if (!empty($errors)) {
330     include BASE_PATH . '/views/academic/school_year_create.php';
331     return;
332 }
333
334 // Crear año lectivo
335 $data = [
336     'institution_id' => $_SESSION['institution_id'],
337     'name' => Security::sanitize($_POST['name']),
338     'start_date' => $_POST['start_date'],
339     'end_date' => $_POST['end_date'],
340     'is_active' => isset($_POST['is_active']) ? 1 : 0
341 ];
342
343 // Si se marca como activo, desactivar los demás ANTES de insertar
344 if ($data['is_active']) {
345     $this->schoolYearModel->activate(0);
346 }
347
348 // create() ahora devuelve el ID insertado (o 0 si falla)
349 $newId = $this->schoolYearModel->create($data);
350
351 if ($newId > 0) {
352     // Activar el nuevo año lectivo con el ID real
353     if ($data['is_active']) {
354         $this->schoolYearModel->activate($newId);
355     }
356     header('Location: ?action=academic&sy_created=1');
357     exit;
358 } else {
359     $errors[] = "Error al crear el año lectivo";
360     include BASE_PATH . '/views/academic/school_year_create.php';

```

```

361 }
362 }
363 }
364
365 public function editSchoolYear() {
366     $yearId = (int)$_GET['id'];
367     $year = $this->schoolYearModel->findById($yearId);
368
369     if (!$year) {
370         header('Location: ?action=academic&error=year_not_found');
371         exit;
372     }
373
374     // Verificar institución
375     if ($year['institution_id'] != $_SESSION['institution_id']) {
376         die('Acceso denegado');
377     }
378
379     if ($_SERVER['REQUEST_METHOD'] === 'GET') {
380         include BASE_PATH . '/views/academic/school_year_edit.php';
381         return;
382     }
383
384     if ($_SERVER['REQUEST_METHOD'] === 'POST') {
385         $errors = [];
386
387         // Validaciones
388         if (empty($_POST['name'])) {
389             $errors[] = "El nombre es obligatorio";
390         }
391         if (empty($_POST['start_date'])) {
392             $errors[] = "La fecha de inicio es obligatoria";
393         }
394         if (empty($_POST['end_date'])) {
395             $errors[] = "La fecha de fin es obligatoria";
396         }
397
398         if (!empty($_POST['start_date']) && !empty($_POST['end_date'])) {
399             if (strtotime($_POST['end_date']) <= strtotime($_POST['start_date'])) {
400                 $errors[] = "La fecha de fin debe ser posterior a la fecha de inicio";
401             }
402         }
403
404         // Verificar solapamiento (excluyendo el actual)
405         if ($this->schoolYearModel->checkOverlap($_POST['start_date'], $_POST['end_date'], $yearId)) {
406             $errors[] = "Ya existe otro año lectivo con fechas que se solapan";
407         }
408
409         if (!empty($errors)) {
410             include BASE_PATH . '/views/academic/school_year_edit.php';
411             return;
412         }
413
414         // Actualizar año lectivo
415         $data = [
416             'id' => $yearId,
417             'name' => Security::sanitize($_POST['name']),
418             'start_date' => $_POST['start_date'],
419             'end_date' => $_POST['end_date']
420         ];

```

```

421
422 if ($this->schoolYearModel->update($data)) {
423 header('Location: ?action=academic&sy_updated=1');
424 exit;
425 } else {
426 $errors[] = "Error al actualizar el año lectivo";
427 include BASE_PATH . '/views/academic/school_year_edit.php';
428 }
429 }
430 }
431
432 public function deleteSchoolYear() {
433 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
434 $yearId = (int)$_POST['year_id'];
435 $year = $this->schoolYearModel->findById($yearId);
436
437 if (!$year) {
438 header('Location: ?action=academic&error=year_not_found');
439 exit;
440 }
441
442 // Verificar institución
443 if ($year['institution_id'] != $_SESSION['institution_id']) {
444 die('Acceso denegado');
445 }
446
447 // No permitir eliminar año activo
448 if ($year['is_active'] == 1) {
449 header('Location: ?action=academic&error=cannot_delete_active');
450 exit;
451 }
452
453 if ($this->schoolYearModel->delete($yearId)) {
454 header('Location: ?action=academic&sy_deleted=1');
455 exit;
456 } else {
457 header('Location: ?action=academic&error=has_courses');
458 exit;
459 }
460 }
461 }
462
463 public function activateSchoolYear() {
464 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
465 $yearId = (int)$_POST['year_id'];
466 $year = $this->schoolYearModel->findById($yearId);
467
468 if (!$year) {
469 header('Location: ?action=academic&error=year_not_found');
470 exit;
471 }
472
473 // Verificar institución
474 if ($year['institution_id'] != $_SESSION['institution_id']) {
475 die('Acceso denegado');
476 }
477
478 if ($this->schoolYearModel->activate($yearId)) {
479 header('Location: ?action=academic&sy_activated=1');
480 exit;

```

```

481 } else {
482 header('Location: ?action=academic&error=activate_failed');
483 exit;
484 }
485 }
486 }
487
488 public function deactivateSchoolYear() {
489 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
490 $yearId = (int)$_POST['year_id'];
491 $year = $this->schoolYearModel->findById($yearId);
492
493 if (!$year) {
494 header('Location: ?action=academic&error=year_not_found');
495 exit;
496 }
497
498 // Verificar institución
499 if ($year['institution_id'] != $_SESSION['institution_id']) {
500 die('Acceso denegado');
501 }
502
503 if ($this->schoolYearModel->deactivate($yearId)) {
504 header('Location: ?action=academic&sy_deactivated=1');
505 exit;
506 } else {
507 header('Location: ?action=academic&error=deactivate_failed');
508 exit;
509 }
510 }
511 }
512 }

```

## 4.2 — AssignmentController.php

controllers/AssignmentController.php

PHP



```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/TeacherAssignment.php';
4 require_once BASE_PATH . '/models/Course.php';
5 require_once BASE_PATH . '/models/Subject.php';
6 require_once BASE_PATH . '/models/SchoolYear.php';
7 require_once BASE_PATH . '/models/User.php';
8
9 class AssignmentController {
10 private $assignmentModel;
11 private $courseModel;
12 private $subjectModel;
13 private $schoolYearModel;
14 private $userModel;
15
16 public function __construct() {
17 $db = new Database();
18 $this->assignmentModel = new TeacherAssignment($db);
19 $this->courseModel = new Course($db);
20 $this->subjectModel = new Subject($db);
21 $this->schoolYearModel = new SchoolYear($db);
22 $this->userModel = new User($db);
23 }
24
25 public function index() {
26 Security::requireLogin();
27 if (!Security::hasRole('autoridad')) {
28 die('Acceso denegado');
29 }
30
31 $courses = $this->courseModel->getAll();
32 $subjects = $this->subjectModel->getAll();
33 $teachers = $this->userModel->getByRole('docente');
34 $activeYear = $this->schoolYearModel->getActive();
35 $assignments = $this->assignmentModel->getAll();
36
37 include BASE_PATH . '/views/assignments/index.php';
38 }
39
40 public function assign() {
41 Security::requireLogin();
42 if (!Security::hasRole('autoridad')) {
43 die('Acceso denegado');
44 }
45
46 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
47 $activeYear = $this->schoolYearModel->getActive();
48
49 $data = [
50 ':teacher_id' => (int)$_POST['teacher_id'],
51 ':course_id' => (int)$_POST['course_id'],
52 ':subject_id' => (int)$_POST['subject_id'],
53 ':school_year_id' => $activeYear['id'],
54 ':is_tutor' => 0
55 ];
56
57 $result = $this->assignmentModel->assign($data);
58
59 if ($result['success']) {
60 header('Location: ?action=assignments&success=1');

```

```

61 } else {
62 header('Location: ?action=assignments&error=' . urlencode($result['message']));
63 }
64 exit;
65 }
66 }
67
68 public function setTutor() {
69 Security::requireLogin();
70 if (!Security::hasRole('autoridad')) {
71 die('Acceso denegado');
72 }
73
74 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
75 $activeYear = $this->schoolYearModel->getActive();
76 $courseId = (int)$_POST['course_id'];
77 $teacherId = (int)$_POST['teacher_id'];
78
79 $result = $this->assignmentModel->setTutor($courseId, $teacherId, $activeYear['id']);
80
81 if ($result['success']) {
82 header('Location: ?action=tutor_management&tutor_success=1');
83 } else {
84 header('Location: ?action=tutor_management&tutor_error=' . urlencode($result['message']));
85 }
86 exit;
87 }
88 }
89
90 public function removeTutor() {
91 Security::requireLogin();
92 if (!Security::hasRole('autoridad')) {
93 die('Acceso denegado');
94 }
95
96 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
97 $activeYear = $this->schoolYearModel->getActive();
98 $courseId = (int)$_POST['course_id'];
99
100 $db = new Database();
101 $sql = "UPDATE teacher_assignments
102 SET is_tutor = 0
103 WHERE course_id = :course_id
104 AND school_year_id = :school_year_id";
105
106 $stmt = $db->connect()->prepare($sql);
107 $stmt->execute([
108 ':course_id' => $courseId,
109 ':school_year_id' => $activeYear['id']
110 ]);
111
112 header('Location: ?action=tutor_management&tutor_removed=1');
113 exit;
114 }
115 }
116
117 public function remove() {
118 Security::requireLogin();
119 if (!Security::hasRole('autoridad')) {
120 die('Acceso denegado');

```

```

121 }
122
123 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
124     $assignmentId = (int)$_POST['assignment_id'];
125     $result = $this->assignmentModel->remove($assignmentId);
126
127     if ($result['success']) {
128         header('Location: ?action=assignments&removed=1');
129     } else {
130         header('Location: ?action=assignments&error=' . urlencode($result['message']));
131     }
132     exit;
133 }
134 }
135
136 public function viewByCourse() {
137     Security::requireLogin();
138     if (!Security::hasRole('autoridad')) {
139         die('Acceso denegado');
140     }
141
142     $courseId = (int)($_GET['course_id'] ?? 0);
143
144     if (!$courseId) {
145         header('Location: ?action=assignments');
146         exit;
147     }
148
149     $courses = $this->courseModel->getAll();
150     $course = array_filter($courses, fn($c) => $c['id'] == $courseId);
151     $course = reset($course);
152
153     $activeYear = $this->schoolYearModel->getActive();
154     $assignments = $this->assignmentModel->getByCourse($courseId);
155     $tutor = $this->assignmentModel->getTutorByCourse($courseId, $activeYear['id']);
156
157     include BASE_PATH . '/views/assignments/view_course.php';
158 }
159
160 public function getCourseTeachers() {
161     if (!isset($_SESSION['user_id'])) {
162         header('Content-Type: application/json');
163         echo json_encode(['error' => 'No autenticado']);
164         exit;
165     }
166
167     $courseId = (int)($_GET['course_id'] ?? 0);
168
169     if (!$courseId) {
170         header('Content-Type: application/json');
171         echo json_encode([]);
172         exit;
173     }
174
175     // Obtener año lectivo activo
176     $db = new Database();
177     $sqlYear = "SELECT id FROM school_years WHERE is_active = 1 LIMIT 1";
178     $stmtYear = $db->connect()->query($sqlYear);
179     $activeYear = $stmtYear->fetch();
180

```

```

181 if (!$activeYear) {
182     header('Content-Type: application/json');
183     echo json_encode([]);
184     exit;
185 }
186
187 // Solo docentes del curso que:
188 // 1. NO son tutores de otro curso
189 // 2. NO son tutores del curso actual
190 $sql = "SELECT DISTINCT ta.teacher_id,
191 CONCAT(u.last_name, ' ', u.first_name) as teacher_name
192 FROM teacher_assignments ta
193 INNER JOIN users u ON ta.teacher_id = u.id
194 WHERE ta.course_id = :course_id
195 AND ta.teacher_id NOT IN (
196 SELECT teacher_id
197 FROM teacher_assignments
198 WHERE is_tutor = 1
199 AND school_year_id = :school_year_id
200 )
201 ORDER BY u.last_name, u.first_name";
202
203 $stmt = $db->connect()->prepare($sql);
204 $stmt->execute([
205     ':course_id' => $courseId,
206     ':school_year_id' => $activeYear['id']
207 ]);
208 $teachers = $stmt->fetchAll(PDO::FETCH_ASSOC);
209
210 header('Content-Type: application/json');
211 echo json_encode($teachers);
212 exit;
213 }
214
215 public function tutorManagement() {
216     $courses = $this->courseModel->getAll();
217     $assignments = $this->assignmentModel->getAll();
218
219     include BASE_PATH . '/views/assignments/tutor.php';
220 }
221
222 public function checkCourseTutor() {
223     if (!isset($_SESSION['user_id'])) {
224         header('Content-Type: application/json');
225         echo json_encode(null);
226         exit;
227     }
228
229     $courseId = (int)($_GET['course_id'] ?? 0);
230
231     if (!$courseId) {
232         header('Content-Type: application/json');
233         echo json_encode(null);
234         exit;
235     }
236
237     $db = new Database();
238     require_once BASE_PATH . '/models/SchoolYear.php';
239     $yearModel = new SchoolYear($db);
240     $activeYear = $yearModel->getActive();

```

```
241
242 $sql = "SELECT CONCAT(u.last_name, ' ', u.first_name) as tutor_name
243 FROM teacher_assignments ta
244 INNER JOIN users u ON ta.teacher_id = u.id
245 WHERE ta.course_id = :course_id
246 AND ta.school_year_id = :school_year_id
247 AND ta.is_tutor = 1
248 LIMIT 1";
249
250 $stmt = $db->connect()->prepare($sql);
251 $stmt->execute([
252 ':course_id' => $courseId,
253 ':school_year_id' => $activeYear['id']
254 ]);
255
256 $tutor = $stmt->fetch(PDO::FETCH_ASSOC);
257
258 header('Content-Type: application/json');
259 echo json_encode($tutor);
260 exit;
261 }
262 }
```

## 4.3 — AttendanceController.php

controllers/AttendanceController.php

PHP

```

1 <?php
2 // ARCHIVO COMPLETO - Reemplaza TODO el archivo
3 // controllers/AttendanceController.php
4
5 require_once BASE_PATH . '/config/config.php';
6 require_once BASE_PATH . '/models/Attendance.php';
7 require_once BASE_PATH . '/models/Course.php';
8 require_once BASE_PATH . '/models/Subject.php';
9 require_once BASE_PATH . '/models/SchoolYear.php';
10 require_once BASE_PATH . '/models/Shift.php';
11 require_once BASE_PATH . '/models/Notification.php';
12
13 class AttendanceController {
14     private $attendanceModel;
15     private $courseModel;
16     private $subjectModel;
17     private $schoolYearModel;
18     private $shiftModel;
19     private $notificationModel;
20
21     public function __construct() {
22         Security::requireLogin();
23         $db = new Database();
24
25         $this->attendanceModel = new Attendance($db);
26         $this->courseModel = new Course($db);
27         $this->subjectModel = new Subject($db);
28         $this->schoolYearModel = new SchoolYear($db);
29         $this->shiftModel = new Shift($db);
30         $this->notificationModel = new Notification($db);
31     }
32
33     private function isWithin48BusinessHours($date) {
34         $targetDate = new DateTime($date);
35         $today = new DateTime();
36         $today->setTime(0, 0, 0);
37
38         if ($targetDate->format('Y-m-d') === $today->format('Y-m-d')) {
39             return true;
40         }
41         if ($targetDate > $today) {
42             return false;
43         }
44
45         $businessHours = 0;
46         $current = clone $targetDate;
47
48         while ($current < $today) {
49             $current->modify('+1 day');
50             $dayOfWeek = (int)$current->format('N');
51             if ($dayOfWeek < 6) {
52                 $businessHours += 24;
53             }
54         }
55
56         return $businessHours <= EDIT_ATTENDANCE_HOURS;
57     }
58
59     private function getMaxEditDate() {
60         $today = new DateTime();

```

```

61 $businessHoursNeeded = EDIT_ATTENDANCE_HOURS;
62 $current = clone $today;
63
64 while ($businessHoursNeeded > 0) {
65     $current->modify('+1 day');
66     $dayOfWeek = (int)$current->format('N');
67     if ($dayOfWeek < 6) {
68         $businessHoursNeeded -= 24;
69     }
70 }
71
72 return $current->format('Y-m-d');
73 }
74
75 public function register() {
76     if (!Security::hasRole(['docente', 'autoridad'])) {
77         die('Acceso denegado');
78     }
79
80     if ($_SERVER['REQUEST_METHOD'] === 'POST') {
81         $activeYear = $this->schoolYearModel->getActive();
82         $date = $_POST['date'];
83         $today = date('Y-m-d');
84
85         if (strtotime($date) > strtotime($today)) {
86             header('Location: ?action=attendance_register&error=future');
87             exit;
88         }
89
90         if (!$this->isWithin48BusinessHours($date)) {
91             header('Location: ?action=attendance_register&error=toolate');
92             exit;
93         }
94
95         $scheduleId = (int)$_POST['schedule_id'];
96
97         // Obtener datos del horario incluyendo nombre de asignatura
98         $db = new Database();
99         $pdo = $db->connect();
100         $sql = "SELECT cs.*, s.name as subject_name
101 FROM class_schedule cs
102 INNER JOIN subjects s ON cs.subject_id = s.id
103 WHERE cs.id = :id";
104 $stmt = $pdo->prepare($sql);
105 $stmt->execute([':id' => $scheduleId]);
106 $scheduleData = $stmt->fetch();
107
108 $subjectName = $scheduleData['subject_name'] ?? 'clase';
109 $dateFormatted = date('d/m/Y', strtotime($date));
110
111 foreach ($_POST as $key => $value) {
112     if (strpos($key, 'status_') === 0) {
113         $studentId = (int)str_replace('status_', '', $key);
114         $status = Security::sanitize($value);
115         $observation = Security::sanitize($_POST['obs_' . $studentId] ?? '');
116
117         $data = [
118             ':student_id' => $studentId,
119             ':course_id' => $scheduleData['course_id'],
120             ':subject_id' => $scheduleData['subject_id'],

```

```

121 ':teacher_id' => $_SESSION['user_id'],
122 ':school_year_id' => $activeYear['id'],
123 ':shift_id' => 1,
124 ':date' => $date,
125 ':hour_period' => $scheduleData['period_number'] . 'ra hora',
126 ':status' => $status,
127 ':observation' => $observation
128 ];
129
130 $this->attendanceModel->create($data);
131
132 // Notificar al estudiante si fue marcado ausente
133 // El enlace lleva directo a su asistencia para que pueda justificar
134 if ($status === 'ausente') {
135 $this->notificationModel->create(
136 $studentId,
137 '■ Ausencia registrada',
138 "Se registró una ausencia el {$dateFormatted} en {$subjectName}. Puedes justificarla.",
139 'ausente',
140 '?action=my_attendance'
141 );
142 }
143
144 // Notificar al estudiante si fue marcado con tardanza
145 if ($status === 'tardanza') {
146 $this->notificationModel->create(
147 $studentId,
148 '■ Tardanza registrada',
149 "Se registró una tardanza el {$dateFormatted} en {$subjectName}.",
150 'warning',
151 '?action=my_attendance'
152 );
153 }
154 }
155 }
156
157 header('Location: ?action=attendance_register&success=1');
158 exit;
159 }
160
161 // Obtener clases del docente para hoy
162 $activeYear = $this->schoolYearModel->getActive();
163
164 $db = new Database();
165 require_once BASE_PATH . '/models/ClassSchedule.php';
166 $scheduleModel = new ClassSchedule($db);
167
168 $todayClasses = $scheduleModel->getTeacherScheduleToday($_SESSION['user_id'], $activeYear['id']);
169 $minDate = $this->calculateMinDate();
170 $maxEditDate = $this->getMaxEditDate();
171
172 include BASE_PATH . '/views/attendance/register.php';
173 }
174
175 private function getTeacherCourses($teacherId) {
176 $sql = "SELECT DISTINCT c.*, sh.name as shift_name
177 FROM courses c
178 INNER JOIN teacher_assignments ta ON c.id = ta.course_id
179 INNER JOIN shifts sh ON c.shift_id = sh.id
180 WHERE ta.teacher_id = :teacher_id";

```



```

181 ORDER BY c.name";
182
183 $db = new Database();
184 $stmt = $db->connect()->prepare($sql);
185 $stmt->execute([':teacher_id' => $teacherId]);
186 return $stmt->fetchAll();
187 }
188
189 private function calculateMinDate() {
190 $date = new DateTime();
191 $hoursBack = 0;
192
193 while ($hoursBack < 48) {
194 $date->modify('-1 day');
195 $dayOfWeek = $date->format('N');
196 if ($dayOfWeek >= 1 && $dayOfWeek <= 5) {
197 $hoursBack += 24;
198 }
199 }
200
201 return $date->format('Y-m-d');
202 }
203
204 public function getStudents() {
205 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
206 $courseId = (int)$_POST['course_id'];
207 $students = $this->courseModel->getStudents($courseId);
208 header('Content-Type: application/json');
209 echo json_encode($students);
210 exit;
211 }
212 }
213
214 public function view() {
215 if (!Security::hasRole(['docente', 'inspector', 'autoridad'])) {
216 die('Acceso denegado');
217 }
218
219 $courses = $this->courseModel->getAll();
220 $attendances = [];
221
222 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
223 $courseId = (int)$_POST['course_id'];
224 $date = $_POST['date'];
225 $attendances = $this->attendanceModel->getByCourse($courseId, $date);
226 }
227
228 include BASE_PATH . '/views/attendance/view.php';
229 }
230
231 public function myAttendance() {
232 if (!Security::hasRole('estudiante')) {
233 die('Acceso denegado');
234 }
235
236 $studentId = $_SESSION['user_id'];
237 $attendances = $this->attendanceModel->getByStudent($studentId);
238
239 include BASE_PATH . '/views/attendance/my_attendance.php';
240 }

```

```

241
242 public function calendar() {
243     if (!Security::hasRole(['docente', 'autoridad', 'inspector'])) {
244         die('Acceso denegado');
245     }
246
247     $courses = $this->courseModel->getAll();
248
249     if (isset($_GET['course_id'])) {
250         $courseId = (int)$_GET['course_id'];
251         $month = $_GET['month'] ?? date('Y-m');
252
253         list($year, $monthNum) = explode('-', $month);
254
255         $monthNames = [
256             '01' => 'Enero', '02' => 'Febrero', '03' => 'Marzo',
257             '04' => 'Abril', '05' => 'Mayo', '06' => 'Junio',
258             '07' => 'Julio', '08' => 'Agosto', '09' => 'Septiembre',
259             '10' => 'Octubre', '11' => 'Noviembre', '12' => 'Diciembre'
260         ];
261
262         $monthName = $monthNames[$monthNum];
263         $prevMonth = date('Y-m', strtotime($month . '-01 -1 month'));
264         $nextMonth = date('Y-m', strtotime($month . '-01 +1 month'));
265         $firstDay = date('w', strtotime($year . '-' . $monthNum . '-01'));
266         $daysInMonth = date('t', strtotime($year . '-' . $monthNum . '-01'));
267
268         $calendarDays = [];
269
270         for ($i = 0; $i < $firstDay; $i++) {
271             $calendarDays[] = ['day' => '', 'classes' => ''];
272         }
273
274         for ($day = 1; $day <= $daysInMonth; $day++) {
275             $date = $year . '-' . $monthNum . '-' . str_pad($day, 2, '0', STR_PAD_LEFT);
276             $dayOfWeek = date('w', strtotime($date));
277             $classes = [];
278
279             if ($date == date('Y-m-d')) $classes[] = 'today';
280             if ($dayOfWeek == 0 || $dayOfWeek == 6) $classes[] = 'weekend';
281
282             $attendance = $this->attendanceModel->getDayStats($courseId, $date);
283
284             if ($attendance && $attendance['total'] > 0) {
285                 $classes[] = 'has-attendance';
286             }
287
288             $calendarDays[] = [
289                 'day' => $day,
290                 'classes' => implode(' ', $classes),
291                 'attendance' => $attendance
292             ];
293         }
294
295         include BASE_PATH . '/views/attendance/calendar.php';
296     } else {
297         include BASE_PATH . '/views/attendance/calendar.php';
298     }
299 }
300

```

```

301 public function getCourseSubjects() {
302 if (!isset($_SESSION['user_id'])) {
303 header('Content-Type: application/json');
304 echo json_encode(['error' => 'No autenticado']);
305 exit;
306 }
307
308 $courseId = (int)($_GET['course_id'] ?? 0);
309
310 if (!$courseId) {
311 header('Content-Type: application/json');
312 echo json_encode([]);
313 exit;
314 }
315
316 $sql = "SELECT DISTINCT ta.subject_id, s.name as subject_name
317 FROM teacher_assignments ta
318 INNER JOIN subjects s ON ta.subject_id = s.id
319 WHERE ta.course_id = :course_id
320 ORDER BY s.name";
321
322 $db = new Database();
323 $stmt = $db->connect()->prepare($sql);
324 $stmt->execute([':course_id' => $courseId]);
325 $subjects = $stmt->fetchAll(PDO::FETCH_ASSOC);
326
327 header('Content-Type: application/json');
328 echo json_encode($subjects);
329 exit;
330 }
331
332 public function getTeacherCourseSubjects() {
333 if (!isset($_SESSION['user_id'])) {
334 header('Content-Type: application/json');
335 echo json_encode(['error' => 'No autenticado']);
336 exit;
337 }
338
339 $courseId = (int)($_GET['course_id'] ?? 0);
340
341 if (!$courseId) {
342 header('Content-Type: application/json');
343 echo json_encode([]);
344 exit;
345 }
346
347 if (Security::hasRole('autoridad')) {
348 $sql = "SELECT DISTINCT ta.subject_id, s.name as subject_name
349 FROM teacher_assignments ta
350 INNER JOIN subjects s ON ta.subject_id = s.id
351 WHERE ta.course_id = :course_id
352 ORDER BY s.name";
353 $params = [':course_id' => $courseId];
354 } else {
355 $sql = "SELECT DISTINCT ta.subject_id, s.name as subject_name
356 FROM teacher_assignments ta
357 INNER JOIN subjects s ON ta.subject_id = s.id
358 WHERE ta.course_id = :course_id
359 AND ta.teacher_id = :teacher_id
360 ORDER BY s.name";

```

```

361 $params = ['course_id' => $courseId, 'teacher_id' => $_SESSION['user_id']];
362 }
363
364 $db = new Database();
365 $stmt = $db->connect()->prepare($sql);
366 $stmt->execute($params);
367 $subjects = $stmt->fetchAll(PDO::FETCH_ASSOC);
368
369 header('Content-Type: application/json');
370 echo json_encode($subjects);
371 exit;
372 }
373
374 public function getScheduleInfo() {
375 if (!isset($_SESSION['user_id'])) {
376 header('Content-Type: application/json');
377 echo json_encode(['error' => 'No autenticado']);
378 exit;
379 }
380
381 $scheduleId = (int)($_GET['schedule_id'] ?? 0);
382
383 if (!$scheduleId) {
384 header('Content-Type: application/json');
385 echo json_encode([]);
386 exit;
387 }
388
389 $sql = "SELECT * FROM class_schedule WHERE id = :id";
390 $db = new Database();
391 $stmt = $db->connect()->prepare($sql);
392 $stmt->execute(['id' => $scheduleId]);
393 $schedule = $stmt->fetch(PDO::FETCH_ASSOC);
394
395 header('Content-Type: application/json');
396 echo json_encode($schedule);
397 exit;
398 }
399
400 public function getExistingAttendance() {
401 header('Content-Type: application/json');
402
403 if (!isset($_SESSION['user_id'])) {
404 echo json_encode([]);
405 exit;
406 }
407
408 $scheduleId = (int)($_GET['schedule_id'] ?? 0);
409 $date = $_GET['date'] ?? date('Y-m-d');
410
411 if (!$scheduleId) {
412 echo json_encode([]);
413 exit;
414 }
415
416 $db = new Database();
417 $pdo = $db->connect();
418 $sql = "SELECT * FROM class_schedule WHERE id = :id";
419 $stmt = $pdo->prepare($sql);
420 $stmt->execute(['id' => $scheduleId]);

```

```

421 $schedule = $stmt->fetch();
422
423 if (!$schedule) {
424     echo json_encode([]);
425     exit;
426 }
427
428 $sql2 = "SELECT student_id, status, observation
429 FROM attendances
430 WHERE course_id = :course_id
431 AND subject_id = :subject_id
432 AND date = :date
433 AND hour_period = :hour_period";
434 $stmt2 = $pdo->prepare($sql2);
435 $stmt2->execute([
436     ':course_id' => $schedule['course_id'],
437     ':subject_id' => $schedule['subject_id'],
438     ':date' => $date,
439     ':hour_period' => $schedule['period_number'] . 'ra hora'
440 ]);
441
442 echo json_encode($stmt2->fetchAll(PDO::FETCH_ASSOC));
443 exit;
444 }
445 }

```

## 4.4 — AuthController.php

controllers/AuthController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/User.php';
4 require_once BASE_PATH . '/helpers/Mailer.php';
5
6 class AuthController {
7     private $userModel;
8
9     public function __construct() {
10         $db = new Database();
11         $this->userModel = new User($db);
12     }
13
14     public function register() {
15         if ($_SERVER['REQUEST_METHOD'] === 'POST') {
16             if (!Security::validateToken($_POST['csrf_token'] ?? '')) {
17                 die('Token inválido');
18             }
19
20             $data = [
21                 'institution_id' => 1,
22                 'username' => Security::sanitize($_POST['username']),
23                 'email' => Security::sanitize($_POST['email']),
24                 'password' => $_POST['password'],
25                 'first_name' => Security::sanitize($_POST['first_name']),
26                 'last_name' => Security::sanitize($_POST['last_name']),
27                 'dni' => Security::sanitize($_POST['dni'] ?? ''),
28                 'phone' => Security::sanitize($_POST['phone'] ?? '')
29             ];
30
31             if ($this->userModel->create($data)) {
32                 header('Location: ' . BASE_URL . '/public/index.php?action=login&registered=1');
33                 exit;
34             }
35         }
36
37         include BASE_PATH . '/views/auth/register.php';
38     }
39
40     public function login() {
41         if ($_SERVER['REQUEST_METHOD'] === 'POST') {
42             if (!Security::validateToken($_POST['csrf_token'] ?? '')) {
43                 die('Token inválido');
44             }
45
46             $emailOrUsername = Security::sanitize($_POST['email']);
47             $password = $_POST['password'];
48
49             // Buscar por email O username
50             $user = $this->userModel->findByEmailOrUsername($emailOrUsername);
51
52             if ($user && Security::verifyPassword($password, $user['password'])) {
53                 $_SESSION['user_id'] = $user['id'];
54                 $_SESSION['username'] = $user['username'];
55                 $_SESSION['first_name'] = $user['first_name'];
56                 $_SESSION['last_name'] = $user['last_name'];
57                 $_SESSION['institution_id'] = $user['institution_id'];
58                 $_SESSION['roles'] = $this->userModel->getUserRoles($user['id']);
59                 $_SESSION['is_superuser'] = !empty($user['is_superuser']);
60

```

```

61 header('Location: ' . BASE_URL . '/public/index.php?action=dashboard');
62 exit;
63 } else {
64 $error = 'Credenciales incorrectas';
65 }
66 }
67
68 include BASE_PATH . '/views/auth/login.php';
69 }
70
71 public function logout() {
72 session_destroy();
73 header('Location: ' . BASE_URL . '/public/index.php?action=login');
74 exit;
75 }
76
77 public function forgotPassword() {
78 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
79 $email = Security::sanitize($_POST['email']);
80 $user = $this->userModel->findByEmail($email);
81
82 if ($user) {
83 $token = bin2hex(random_bytes(32));
84 $this->userModel->setResetToken($email, $token);
85
86 $resetLink = BASE_URL . "/public/index.php?action=reset&token={$token}";
87 $body = "Haga clic aquí para restablecer su contraseña: <a href='{$resetLink}'>{$resetLink}</a>";
88
89 Mailer::send($email, 'Restablecer contraseña', $body);
90 }
91
92 $message = 'Si el correo existe, recibirá un enlace de recuperación';
93 }
94
95 include BASE_PATH . '/views/auth/forgot.php';
96 }
97
98 public function resetPassword() {
99 $token = $_GET['token'] ?? '';
100
101 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
102 $user = $this->userModel->validateResetToken($token);
103
104 if ($user && $_POST['password'] === $_POST['password_confirm']) {
105 $this->userModel->resetPassword($user['id'], $_POST['password']);
106 header('Location: ' . BASE_URL . '/public/index.php?action=login&reset=1');
107 exit;
108 }
109 }
110
111 include BASE_PATH . '/views/auth/reset.php';
112 }
113 }

```

## 4.5 — BackupController.php

controllers/BackupController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/helpers/Backup.php';
4
5 class BackupController {
6     private $backup;
7
8     public function __construct() {
9         Security::requireLogin();
10        if (!Security::hasRole('autoridad')) {
11            die('Acceso denegado');
12        }
13
14        $this->backup = new Backup();
15    }
16
17    public function index() {
18        $backups = $this->backup->getBackups();
19        include BASE_PATH . '/views/backup/index.php';
20    }
21
22    public function create() {
23        $filename = $this->backup->createBackup();
24
25        if ($filename) {
26            header('Location: ?action=backups&success=1&file=' . $filename);
27        } else {
28            header('Location: ?action=backups&error=1');
29        }
30        exit;
31    }
32
33    public function download() {
34        $filename = $_GET['file'] ?? '';
35        $this->backup->downloadBackup($filename);
36    }
37
38    public function cleanup() {
39        $deleted = $this->backup->deleteOldBackups(30);
40        header('Location: ?action=backups&cleanup=' . $deleted);
41        exit;
42    }
43
44    public function delete() {
45        $filename = $_GET['file'] ?? '';
46
47        if ($this->backup->deleteBackup($filename)) {
48            header('Location: ?action=backups&deleted=1');
49        } else {
50            header('Location: ?action=backups&error_delete=1');
51        }
52        exit;
53    }
54 }

```

## 4.6 — DashboardController.php

controllers/DashboardController.php

PHP



```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/Attendance.php';
4 require_once BASE_PATH . '/models/User.php';
5 require_once BASE_PATH . '/models/Course.php';
6
7 class DashboardController {
8     private $attendanceModel;
9     private $userModel;
10    private $courseModel;
11
12    public function __construct() {
13        Security::requireLogin();
14
15        $db = new Database();
16        $this->attendanceModel = new Attendance($db);
17        $this->userModel = new User($db);
18        $this->courseModel = new Course($db);
19    }
20
21    public function index() {
22        $stats = $this->getStats();
23        include BASE_PATH . '/views/dashboard/index.php';
24    }
25
26    private function getStats() {
27        $stats = [];
28
29        if (Security::hasRole('autoridad')) {
30            $stats['total_students'] = $this->getTotalByRole('estudiante');
31            $stats['total_teachers'] = $this->getTotalByRole('docente');
32            $stats['total_courses'] = $this->getTotalCourses();
33            $stats['today_attendance'] = $this->getTodayAttendanceStats();
34        }
35
36        if (Security::hasRole('docente')) {
37            $stats['my_courses'] = $this->getTeacherCourses($_SESSION['user_id']);
38            $stats['my_students'] = $this->getTeacherStudentsCount($_SESSION['user_id']);
39            $stats['tutor_course'] = $this->getTutorCourse($_SESSION['user_id']);
40        }
41
42        if (Security::hasRole('estudiante')) {
43            $stats['my_attendance'] = $this->getStudentAttendanceStats($_SESSION['user_id']);
44        }
45
46        if (Security::hasRole('representante')) {
47            $stats['my_children'] = $this->getRepresentativeChildrenCount($_SESSION['user_id']);
48        }
49
50        return $stats;
51    }
52
53    private function getTotalByRole($role) {
54        $sql = "SELECT COUNT(DISTINCT u.id) as total
55        FROM users u
56        INNER JOIN user_roles ur ON u.id = ur.user_id
57        INNER JOIN roles r ON ur.role_id = r.id
58        WHERE r.name = :role AND u.institution_id = :institution_id";
59
60        $db = new Database();

```

```

61 $stmt = $db->connect()->prepare($sql);
62 $stmt->execute([
63 ':role' => $role,
64 ':institution_id' => $_SESSION['institution_id']
65 ]);
66
67 return $stmt->fetch()['total'];
68 }
69
70 private function getTotalCourses() {
71 $sql = "SELECT COUNT(*) as total FROM courses WHERE institution_id = :institution_id";
72
73 $db = new Database();
74 $stmt = $db->connect()->prepare($sql);
75 $stmt->execute([':institution_id' => $_SESSION['institution_id']]);
76
77 return $stmt->fetch()['total'];
78 }
79
80 private function getTodayAttendanceStats() {
81 $sql = "SELECT
82 COUNT(*) as total,
83 SUM(CASE WHEN status = 'presente' THEN 1 ELSE 0 END) as presente,
84 SUM(CASE WHEN status = 'ausente' THEN 1 ELSE 0 END) as ausente
85 FROM attendances
86 WHERE date = CURDATE()";
87
88 $db = new Database();
89 $stmt = $db->connect()->query($sql);
90 return $stmt->fetch();
91 }
92
93 private function getTeacherCourses($teacherId) {
94 $sql = "SELECT COUNT(DISTINCT course_id) as total
95 FROM teacher_assignments
96 WHERE teacher_id = :teacher_id";
97
98 $db = new Database();
99 $stmt = $db->connect()->prepare($sql);
100 $stmt->execute([':teacher_id' => $teacherId]);
101
102 return $stmt->fetch()['total'];
103 }
104
105 private function getTeacherStudentsCount($teacherId) {
106 $sql = "SELECT COUNT(DISTINCT cs.student_id) as total
107 FROM teacher_assignments ta
108 INNER JOIN course_students cs ON ta.course_id = cs.course_id
109 WHERE ta.teacher_id = :teacher_id";
110
111 $db = new Database();
112 $stmt = $db->connect()->prepare($sql);
113 $stmt->execute([':teacher_id' => $teacherId]);
114
115 return $stmt->fetch()['total'];
116 }
117
118 private function getStudentAttendanceStats($studentId) {
119 $sql = "SELECT
120 COUNT(*) as total,

```

```

121 SUM(CASE WHEN status = 'presente' THEN 1 ELSE 0 END) as presente,
122 SUM(CASE WHEN status = 'ausente' THEN 1 ELSE 0 END) as ausente,
123 SUM(CASE WHEN status = 'tardanza' THEN 1 ELSE 0 END) as tardanza
124 FROM attendances
125 WHERE student_id = :student_id
126 AND date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)";
127
128 $db = new Database();
129 $stmt = $db->connect()->prepare($sql);
130 $stmt->execute([':student_id' => $studentId]);
131
132 return $stmt->fetch();
133 }
134
135 private function getRepresentativeChildrenCount($representativeId) {
136 $sql = "SELECT COUNT(*) as total
137 FROM representatives
138 WHERE representative_id = :representative_id";
139
140 $db = new Database();
141 $stmt = $db->connect()->prepare($sql);
142 $stmt->execute([':representative_id' => $representativeId]);
143
144 return $stmt->fetch()['total'];
145 }
146
147 private function getTutorCourse($teacherId) {
148 $sql = "SELECT c.name as course_name
149 FROM teacher_assignments ta
150 INNER JOIN courses c ON ta.course_id = c.id
151 INNER JOIN school_years sy ON ta.school_year_id = sy.id
152 WHERE ta.teacher_id = :teacher_id
153 AND ta.is_tutor = 1
154 AND sy.is_active = 1
155 LIMIT 1";
156
157 $db = new Database();
158 $stmt = $db->connect()->prepare($sql);
159 $stmt->execute([':teacher_id' => $teacherId]);
160 $row = $stmt->fetch();
161 return $row ? $row['course_name'] : null;
162 }
163 }

```

## 4.7 — ExportController.php

controllers/ExportController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/vendor/autoload.php';
4 require_once BASE_PATH . '/models/Course.php';
5
6 use PhpOffice\PhpSpreadsheet\Spreadsheet;
7 use PhpOffice\PhpSpreadsheet\Writer\Xlsx;
8 use PhpOffice\PhpSpreadsheet\Style\Fill;
9 use PhpOffice\PhpSpreadsheet\Style\Alignment;
10
11 class ExportController {
12     private $courseModel;
13
14     public function __construct() {
15         Security::requireLogin();
16         if (!Security::hasRole(['autoridad', 'docente'])) {
17             die('Acceso denegado');
18         }
19
20         $db = new Database();
21         $this->courseModel = new Course($db);
22     }
23
24     public function exportStudentList() {
25         $courseId = (int)$_GET['course_id'];
26         $students = $this->courseModel->getEnrolledStudents($courseId);
27
28         $courses = $this->courseModel->getAll();
29         $course = array_filter($courses, fn($c) => $c['id'] == $courseId);
30         $course = reset($course);
31
32         $spreadsheet = new Spreadsheet();
33         $sheet = $spreadsheet->getActiveSheet();
34
35         $sheet->setTitle('Nómina');
36
37         // Encabezado
38         $sheet->mergeCells('A1:E1');
39         $sheet->setCellValue('A1', 'NÓMINA DE ESTUDIANTES');
40         $sheet->getStyle('A1')->getFont()->setBold(true)->setSize(16);
41         $sheet->getStyle('A1')->getAlignment()->setHorizontal(Alignment::HORIZONTAL_CENTER);
42
43         $sheet->setCellValue('A3', 'Curso:');
44         $sheet->setCellValue('B3', $course['name'] . ' - ' . $course['shift_name']);
45         $sheet->getStyle('A3')->getFont()->setBold(true);
46
47         $sheet->setCellValue('A4', 'Total:');
48         $sheet->setCellValue('B4', count($students) . ' estudiantes');
49         $sheet->getStyle('A4')->getFont()->setBold(true);
50
51         // Cabecera tabla
52         $headers = ['#', 'Apellidos', 'Nombres', 'Cédula', 'Fecha Matrícula'];
53         $col = 'A';
54         foreach ($headers as $header) {
55             $sheet->setCellValue($col . '6', $header);
56             $sheet->getStyle($col . '6')->getFont()->setBold(true);
57             $sheet->getStyle($col . '6')->getFill()
58             ->setFillType(Fill::FILL_SOLID)
59             ->getStartColor()->setRGB('007bff');
60             $sheet->getStyle($col . '6')->getFont()->getColor()->setRGB('FFFFFF');

```

```

61 $col++;
62 }
63
64 // Datos
65 $row = 7;
66 $num = 1;
67 foreach ($students as $student) {
68 $sheet->setCellValue('A' . $row, $num++);
69 $sheet->setCellValue('B' . $row, $student['last_name']);
70 $sheet->setCellValue('C' . $row, $student['first_name']);
71 $sheet->setCellValue('D' . $row, $student['dni'] ?? '-');
72 $sheet->setCellValue('E' . $row, date('d/m/Y', strtotime($student['enrollment_date'])));
73 $row++;
74 }
75
76 // Ajustar anchos
77 $sheet->getColumnDimension('A')->setWidth(5);
78 $sheet->getColumnDimension('B')->setWidth(20);
79 $sheet->getColumnDimension('C')->setWidth(20);
80 $sheet->getColumnDimension('D')->setWidth(15);
81 $sheet->getColumnDimension('E')->setWidth(15);
82
83 $writer = new Xlsx($spreadsheet);
84
85 header('Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
86 header('Content-Disposition: attachment;filename="nomina_' . $course['name'] . '_' . date('YmdHis') .
    '.xlsx"');
87 header('Cache-Control: max-age=0');
88
89 $writer->save('php://output');
90 exit;
91 }
92 }

```

## 4.8 — InstitutionController.php

controllers/InstitutionController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/Institution.php';
4 require_once BASE_PATH . '/models/InstitutionShift.php';
5 require_once BASE_PATH . '/models/Shift.php';
6
7 class InstitutionController {
8     private $institutionModel;
9     private $institutionShiftModel;
10    private $shiftModel;
11
12    public function __construct() {
13        Security::requireLogin();
14        if (!Security::hasRole('autoridad')) {
15            die('Acceso denegado');
16        }
17
18        $db = new Database();
19        $this->institutionModel = new Institution($db);
20        $this->institutionShiftModel = new InstitutionShift($db);
21        $this->shiftModel = new Shift($db);
22    }
23
24    public function index() {
25        $institution = $this->institutionModel->getById(1);
26        $allShifts = $this->shiftModel->getAll();
27        $assignedShiftIds = $this->institutionShiftModel->getInstitutionShiftIds(1);
28
29        include BASE_PATH . '/views/institution/index.php';
30    }
31
32    public function update() {
33        if ($_SERVER['REQUEST_METHOD'] !== 'POST') return;
34
35        $current = $this->institutionModel->getById(1);
36        $logoPath = $current['logo_path']; // Mantener logo actual por defecto
37
38        if (isset($_FILES['logo']) && $_FILES['logo']['error'] == 0) {
39            $uploadDir = BASE_PATH . '/uploads/institution/';
40            if (!file_exists($uploadDir)) {
41                mkdir($uploadDir, 0755, true);
42            }
43
44            $ext = strtolower(pathinfo($_FILES['logo']['name'], PATHINFO_EXTENSION));
45            $allowed = ['jpg', 'jpeg', 'png', 'gif', 'webp'];
46            $maxSize = 2 * 1024 * 1024; // 2MB
47
48            if (in_array($ext, $allowed) && $_FILES['logo']['size'] <= $maxSize) {
49                $filename = 'logo_' . time() . '.' . $ext;
50                $uploadPath = $uploadDir . $filename;
51
52                if (move_uploaded_file($_FILES['logo']['tmp_name'], $uploadPath)) {
53                    // Eliminar logo anterior
54                    if ($current['logo_path'] && file_exists(BASE_PATH . '/' . $current['logo_path'])) {
55                        unlink(BASE_PATH . '/' . $current['logo_path']);
56                    }
57                    $logoPath = 'uploads/institution/' . $filename;
58                }
59            }
60        }

```

```

61
62 $data = [
63 'name' => Security::sanitize($_POST['name']),
64 'address' => Security::sanitize($_POST['address']),
65 'province' => Security::sanitize($_POST['province']),
66 'city' => Security::sanitize($_POST['city']),
67 'phone' => Security::sanitize($_POST['phone']),
68 'email' => Security::sanitize($_POST['email']),
69 'director_name'=> Security::sanitize($_POST['director_name']),
70 'amie_code' => Security::sanitize($_POST['amie_code']),
71 'website' => Security::sanitize($_POST['website']),
72 'logo_path' => $logoPath,
73 ];
74
75 $this->institutionModel->update(1, $data);
76 header('Location: ?action=institution&success=1');
77 exit;
78 }
79
80 // Toggle: si está asignada la quita, si no la asigna - responde JSON para AJAX
81 public function toggleShift() {
82 header('Content-Type: application/json');
83
84 if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
85 echo json_encode(['error' => 'Método no permitido']); exit;
86 }
87
88 $shiftId = (int)$_POST['shift_id'];
89 $assignedShiftIds = $this->institutionShiftModel->getInstitutionShiftIds(1);
90
91 if (in_array($shiftId, $assignedShiftIds)) {
92 $this->institutionShiftModel->remove(1, $shiftId);
93 echo json_encode(['action' => 'removed', 'shift_id' => $shiftId]);
94 } else {
95 $this->institutionShiftModel->assign([
96 ':institution_id' => 1,
97 ':shift_id' => $shiftId,
98 ]);
99 echo json_encode(['action' => 'assigned', 'shift_id' => $shiftId]);
100 }
101 exit;
102 }
103
104 // Mantener compatibilidad con rutas antiguas
105 public function assignShift() {
106 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
107 $this->institutionShiftModel->assign([
108 ':institution_id' => 1,
109 ':shift_id' => (int)$_POST['shift_id'],
110 ]);
111 header('Location: ?action=institution&success=1');
112 exit;
113 }
114 }
115
116 public function removeShift() {
117 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
118 $this->institutionShiftModel->remove(1, (int)$_POST['shift_id']);
119 header('Location: ?action=institution&success=1');
120 exit;
121 }
122 }
123 }

```

## 4.9 — JustificationController.php

```
1 <?php
2 // ARCHIVO COMPLETO - Reemplaza TODO el archivo
3 // controllers/JustificationController.php
4
5 require_once BASE_PATH . '/config/config.php';
6 require_once BASE_PATH . '/models/Justification.php';
7 require_once BASE_PATH . '/models/Attendance.php';
8 require_once BASE_PATH . '/models/Attendance.php';
9 require_once BASE_PATH . '/models/Notification.php';
10
11 class JustificationController {
12     private $justificationModel;
13     private $attendanceModel;
14     private $notificationModel;
15
16     public function __construct() {
17         Security::requireLogin();
18
19         $db = new Database();
20         $this->justificationModel = new Justification($db);
21         $this->attendanceModel = new Attendance($db);
22         $this->notificationModel = new Notification($db);
23     }
24
25     public function submit() {
26         if (!Security::hasRole(['estudiante', 'representante'])) {
27             die('Acceso denegado');
28         }
29
30         $studentId = Security::hasRole('estudiante')
31             ? $_SESSION['user_id']
32             : (isset($_GET['student_id']) ? (int)$_GET['student_id'] : null);
33
34         if ($_SERVER['REQUEST_METHOD'] === 'POST') {
35
36             $studentId = Security::hasRole('estudiante')
37                 ? $_SESSION['user_id']
38                 : (int)$_POST['student_id'];
39
40             // Ausencias seleccionadas
41             $attendanceIds = $_POST['attendance_ids'] ?? [];
42             if (empty($attendanceIds)) {
43                 header('Location: ?action=submit_justification&error=no_absences');
44                 exit;
45             }
46
47             // Obtener fechas de las ausencias seleccionadas para calcular días laborables
48             $db = new Database();
49             $pdo = $db->connect();
50             $in = implode(',', array_map('intval', $attendanceIds));
51             $rows = $pdo->query(
52                 "SELECT date FROM attendances WHERE id IN ($in) AND student_id = $studentId ORDER BY date"
53             )->fetchAll();
54
55             if (empty($rows)) {
56                 header('Location: ?action=submit_justification&error=no_absences');
57                 exit;
58             }
59
60             $dates = array_column($rows, 'date');
```



```

61 $dateFrom = min($dates);
62 $dateTo = max($dates);
63 $workingDays = count($attendanceIds); // cada ausencia = 1 día (pueden ser no consecutivos)
64 $canApprove = Justification::resolveApprover($workingDays);
65
66 // Subir documento
67 $documentPath = null;
68 if (isset($_FILES['document']) && $_FILES['document']['error'] == 0) {
69 $uploadDir = BASE_PATH . '/uploads/justifications/';
70 if (!file_exists($uploadDir)) mkdir($uploadDir, 0755, true);
71 $ext = strtolower(pathinfo($_FILES['document']['name'], PATHINFO_EXTENSION));
72 $allowed = ['pdf','jpg','jpeg','png'];
73 if (in_array($ext, $allowed) && $_FILES['document']['size'] <= 5 * 1024 * 1024) {
74 $filename = uniqid() . '.' . $ext;
75 if (move_uploaded_file($_FILES['document']['tmp_name'], $uploadDir . $filename)) {
76 $documentPath = 'uploads/justifications/' . $filename;
77 }
78 }
79 }
80
81 // Motivo
82 $reasonType = Security::sanitize($_POST['reason_type'] ?? '');
83 $reasonText = Security::sanitize($_POST['reason'] ?? '');
84 $reasonFinal = ($reasonType === 'Otro')
85 ? $reasonText
86 : ($reasonType . ($reasonText ? ': ' . $reasonText : ''));
87
88 $data = [
89 ':student_id' => $studentId,
90 ':submitted_by' => $_SESSION['user_id'],
91 ':date_from' => $dateFrom,
92 ':date_to' => $dateTo,
93 ':working_days' => $workingDays,
94 ':reason_type' => $reasonType,
95 ':reason' => $reasonFinal,
96 ':document_path' => $documentPath,
97 ':can_approve' => $canApprove,
98 ];
99
100 $this->justificationModel->createForAttendances($attendanceIds, $data);
101
102 // Notificar
103 if ($canApprove === 'tutor') {
104 $stmt = $pdo->prepare(
105 "SELECT ta.teacher_id FROM teacher_assignments ta
106 INNER JOIN course_students cs ON ta.course_id = cs.course_id
107 WHERE cs.student_id = :sid AND ta.is_tutor = 1 LIMIT 1"
108 );
109 $stmt->execute([':sid' => $studentId]);
110 $tutorId = $stmt->fetchColumn();
111 if ($tutorId) {
112 $this->notificationModel->create(
113 $tutorId,
114 '■ Justificación pendiente (tutor)',
115 "Un estudiante de tu curso necesita justificación por $workingDays día(s).",
116 'info',
117 '?action=tutor_pending_justifications'
118 );
119 }
120 } else {

```

```

121 $this->notifyReviewers(
122 '■ Nueva justificación pendiente',
123 "Justificación de $workingDays día(s) requiere revisión.",
124 'info',
125 '?action=pending_justifications'
126 );
127 }
128
129 header('Location: ?action=my_justifications&success=1');
130 exit;
131 }
132
133 // GET: cargar ausencias no justificadas
134 $absences = $this->attendanceModel->getUnjustifiedAbsences($studentId ?? $_SESSION['user_id']);
135
136 // Si representante, cargar sus hijos
137 $myChildren = [];
138 if (Security::hasRole('representante')) {
139     $db = new Database();
140     $stmt = $db->connect()->prepare(
141 "SELECT u.id, CONCAT(u.last_name, ' ',u.first_name) as full_name
142 FROM users u INNER JOIN representatives r ON u.id = r.student_id
143 WHERE r.representative_id = :rid ORDER BY u.last_name"
144 );
145 $stmt->execute([':rid' => $_SESSION['user_id']]);
146 $myChildren = $stmt->fetchAll();
147 }
148
149 include BASE_PATH . '/views/justifications/submit.php';
150 }
151
152
153 public function pendingForTutor() {
154 if (!Security::hasRole('docente')) die('Acceso denegado');
155
156 $db = new Database();
157 $attModel = new Attendance($db);
158 $courseId = $attModel->getTutorCourseId($_SESSION['user_id']);
159
160 if (!$courseId) {
161 header('Location: ?action=dashboard&error=not_tutor'); exit;
162 }
163
164 $justifications = $this->justificationModel->getPendingForTutor($courseId);
165 include BASE_PATH . '/views/justifications/pending.php';
166 }
167
168 public function myJustifications() {
169 if (!Security::hasRole(['estudiante', 'representante'])) {
170 die('Acceso denegado');
171 }
172 $studentId = $_SESSION['user_id'];
173 $justifications = $this->justificationModel->getByStudent($studentId);
174 include BASE_PATH . '/views/justifications/my_list.php';
175 }
176
177 public function pending() {
178 if (!Security::hasRole(['autoridad', 'inspector'])) {
179 die('Acceso denegado');
180 }

```

```

181 $justifications = $this->justificationModel->getPending();
182 include BASE_PATH . '/views/justifications/pending.php';
183 }
184
185 public function review() {
186 if (!Security::hasRole(['autoridad', 'inspector'])) {
187 die('Acceso denegado');
188 }
189
190 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
191 $justificationId = (int)$_POST['justification_id'];
192 $action = $_POST['review_action'];
193 $notes = Security::sanitize($_POST['notes'] ?? '');
194
195 // Obtener datos ANTES de actualizar para poder notificar
196 $justification = $this->justificationModel->getById($justificationId);
197
198 if ($action === 'approve') {
199 $this->justificationModel->approveRange($justificationId, $_SESSION['user_id'], $notes);
200
201 if ($justification) {
202 // Notificar al estudiante – enlace a sus justificaciones
203 $this->notificationModel->create(
204 $justification['student_id'],
205 '■ Justificación aprobada',
206 'Tu justificación fue aprobada.' . ($notes ? " Nota: $notes" : ''),
207 'success',
208 '?action=my_justifications'
209 );
210 // Notificar al representante si fue distinto del estudiante
211 if ($justification['submitted_by'] != $justification['student_id']) {
212 $this->notificationModel->create(
213 $justification['submitted_by'],
214 '■ Justificación aprobada',
215 'La justificación que enviaste fue aprobada.' . ($notes ? " Nota: $notes" : ''),
216 'success',
217 '?action=my_justifications'
218 );
219 }
220 }
221 } else {
222 $this->justificationModel->reject($justificationId, $_SESSION['user_id'], $notes);
223
224 if ($justification) {
225 // Notificar al estudiante – puede enviar una nueva justificación
226 $this->notificationModel->create(
227 $justification['student_id'],
228 '■ Justificación rechazada',
229 'Tu justificación fue rechazada.' . ($notes ? " Motivo: $notes" : ''),
230 'danger',
231 '?action=my_justifications'
232 );
233 if ($justification['submitted_by'] != $justification['student_id']) {
234 $this->notificationModel->create(
235 $justification['submitted_by'],
236 '■ Justificación rechazada',
237 'La justificación que enviaste fue rechazada.' . ($notes ? " Motivo: $notes" : ''),
238 'danger',
239 '?action=my_justifications'
240 );

```

```

241 }
242 }
243 }
244
245 header('Location: ?action=pending_justifications&success=1');
246 exit;
247 }
248 }
249
250 public function reviewed() {
251 if (!Security::hasRole(['autoridad', 'inspector'])) {
252 die('Acceso denegado');
253 }
254 $filter = $_GET['filter'] ?? 'all';
255 $justifications = $this->justificationModel->getReviewed();
256
257 if ($filter !== 'all') {
258 $justifications = array_values(array_filter($justifications, function($j) use ($filter) {
259 return $j['status'] === $filter;
260 }));
261 }
262
263 include BASE_PATH . '/views/justifications/reviewed.php';
264 }
265
266 // Privado: notificar a todos los revisores
267 private function _notifyReviewers($title, $message, $type = 'info', $link = null) {
268 $db = new Database();
269 $pdo = $db->connect();
270
271 $sql = "SELECT DISTINCT u.id FROM users u
272 INNER JOIN user_roles ur ON u.id = ur.user_id
273 INNER JOIN roles r ON ur.role_id = r.id
274 WHERE r.name IN ('autoridad','inspector') AND u.id != :sender";
275 $stmt = $pdo->prepare($sql);
276 $stmt->execute([':sender' => $_SESSION['user_id']]);
277 $reviewers = $stmt->fetchAll(PDO::FETCH_COLUMN);
278
279 foreach ($reviewers as $reviewerId) {
280 $this->notificationModel->create($reviewerId, $title, $message, $type, $link);
281 }
282 }
283 }

```

## 4.10 — NotificationController.php

controllers/NotificationController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/Notification.php';
4
5 class NotificationController {
6     private $notificationModel;
7
8     public function __construct() {
9         Security::requireLogin();
10        $db = new Database();
11        $this->notificationModel = new Notification($db);
12    }
13
14    // Vista principal: todas las notificaciones del usuario
15    public function index() {
16        $userId = $_SESSION['user_id'];
17        $perPage = 15;
18        $page = max(1, (int)($_GET['page'] ?? 1));
19        $offset = ($page - 1) * $perPage;
20
21        $notifications = $this->notificationModel->getByUser($userId, $perPage, $offset);
22        $total = $this->notificationModel->countByUser($userId);
23        $totalPages = (int)ceil($total / $perPage);
24        $unreadCount = $this->notificationModel->getUnreadCount($userId);
25
26        include BASE_PATH . '/views/notifications/index.php';
27    }
28
29    // AJAX: marcar una como leída y devolver nuevo conteo
30    public function markRead() {
31        $userId = $_SESSION['user_id'];
32        $id = (int)($_GET['id'] ?? 0);
33
34        if ($id) {
35            $this->notificationModel->markAsRead($id, $userId);
36        }
37
38        // Responder JSON para el dropdown del navbar
39        header('Content-Type: application/json');
40        echo json_encode([
41            'success' => true,
42            'unread' => $this->notificationModel->getUnreadCount($userId)
43        ]);
44        exit;
45    }
46
47    // Marcar todas como leídas (POST desde vista)
48    public function markAllRead() {
49        $userId = $_SESSION['user_id'];
50        $this->notificationModel->markAllAsRead($userId);
51        header('Location: ?action=notifications&success=read');
52        exit;
53    }
54
55    // Eliminar una notificación (POST desde vista)
56    public function delete() {
57        $userId = $_SESSION['user_id'];
58        $id = (int)($_POST['notification_id'] ?? 0);
59
60        if ($id) {

```

```

61 $this->notificationModel->delete($id, $userId);
62 }
63
64 header('Location: ?action=notifications&success=deleted');
65 exit;
66 }
67
68 // Eliminar todas las leídas
69 public function deleteRead() {
70 $userId = $_SESSION['user_id'];
71 $this->notificationModel->deleteRead($userId);
72 header('Location: ?action=notifications&success=cleaned');
73 exit;
74 }
75
76 // AJAX: obtener no leídas para el dropdown del navbar
77 public function getUnread() {
78 $userId = $_SESSION['user_id'];
79 $notifications = $this->notificationModel->getUnread($userId, 8);
80 $count = $this->notificationModel->getUnreadCount($userId);
81
82 header('Content-Type: application/json');
83 echo json_encode([
84 'notifications' => $notifications,
85 'count' => $count
86 ]);
87 exit;
88 }
89 }

```

## 4.11 — ProfileController.php

controllers/ProfileController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/User.php';
4
5 class ProfileController {
6     private $userModel;
7
8     public function __construct() {
9         Security::requireLogin();
10
11         $db = new Database();
12         $this->userModel = new User($db);
13     }
14
15     public function view() {
16         $user = $this->userModel->findById($_SESSION['user_id']);
17         $roles = $this->userModel->getUserRoles($_SESSION['user_id']);
18
19         include BASE_PATH . '/views/profile/view.php';
20     }
21
22     public function edit() {
23         $user = $this->userModel->findById($_SESSION['user_id']);
24
25         if ($_SERVER['REQUEST_METHOD'] === 'POST') {
26             $data = [
27                 'first_name' => Security::sanitize($_POST['first_name']),
28                 'last_name' => Security::sanitize($_POST['last_name']),
29                 'phone' => Security::sanitize($_POST['phone']),
30                 'email' => Security::sanitize($_POST['email'])
31             ];
32
33             if ($this->userModel->updateProfile($_SESSION['user_id'], $data)) {
34                 header('Location: ?action=profile&success=1');
35                 exit;
36             }
37         }
38
39         include BASE_PATH . '/views/profile/edit.php';
40     }
41
42     public function changePassword() {
43         if ($_SERVER['REQUEST_METHOD'] === 'POST') {
44             $currentPassword = $_POST['current_password'];
45             $newPassword = $_POST['new_password'];
46             $confirmPassword = $_POST['confirm_password'];
47
48             $user = $this->userModel->findById($_SESSION['user_id']);
49
50             if (!Security::verifyPassword($currentPassword, $user['password'])) {
51                 $error = 'Contraseña actual incorrecta';
52             } elseif ($newPassword !== $confirmPassword) {
53                 $error = 'Las contraseñas no coinciden';
54             } elseif (strlen($newPassword) < 6) {
55                 $error = 'La contraseña debe tener al menos 6 caracteres';
56             } else {
57                 if ($this->userModel->updatePassword($_SESSION['user_id'], $newPassword)) {
58                     header('Location: ?action=profile&password_changed=1');
59                     exit;
60                 }
61             }
62         }
63
64         include BASE_PATH . '/views/profile/change_password.php';
65     }
66 }

```

## 4.12 — ReportController.php

controllers/ReportController.php

PHP

```
1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/Attendance.php';
4 require_once BASE_PATH . '/models/Course.php';
5 require_once BASE_PATH . '/models/Subject.php';
6 require_once BASE_PATH . '/models/SchoolYear.php';
7 require_once BASE_PATH . '/models/User.php';
8 require_once BASE_PATH . '/models/Institution.php';
9 require_once BASE_PATH . '/vendor/autoload.php';
10
11 use PhpOffice\PhpSpreadsheet\Spreadsheet;
12 use PhpOffice\PhpSpreadsheet\Writer\Xlsx;
13 use PhpOffice\PhpSpreadsheet\Style\Fill;
14 use PhpOffice\PhpSpreadsheet\Style\Border;
15 use PhpOffice\PhpSpreadsheet\Style\Alignment;
16
17 class ReportController {
18     private $attendanceModel;
19     private $courseModel;
20     private $subjectModel;
21     private $schoolYearModel;
22     private $userModel;
23     private $institutionModel;
24
25     public function __construct() {
26         Security::requireLogin();
27         if (!Security::hasRole(['autoridad', 'inspector', 'docente'])) {
28             die('Acceso denegado');
29         }
30
31         $db = new Database();
32         $this->attendanceModel = new Attendance($db);
33         $this->courseModel = new Course($db);
34         $this->subjectModel = new Subject($db);
35         $this->schoolYearModel = new SchoolYear($db);
36         $this->userModel = new User($db);
37         $this->institutionModel = new Institution($db);
38     }
39
40     public function index() {
41         $courses = $this->courseModel->getAll();
42         $subjects = $this->subjectModel->getAll();
43
44         $data = [];
45         $course = null;
46         $startDate = null;
47         $endDate = null;
48
49         if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['preview'])) {
50             $courseId = (int)$_POST['course_id'];
51             $startDate = $_POST['start_date'];
52             $endDate = $_POST['end_date'];
53
54             $data = $this->attendanceModel->getReportData($courseId, $startDate, $endDate);
55             $course = $this->getCourseInfo($courseId);
56         }
57
58         include BASE_PATH . '/views/reports/index.php';
59     }
60 }
```



```

61 public function generatePDF() {
62     $courseId = (int)$_POST['course_id'];
63     $startDate = $_POST['start_date'];
64     $endDate = $_POST['end_date'];
65
66     $data = $this->attendanceModel->getReportData($courseId, $startDate, $endDate);
67     $course = $this->getCourseInfo($courseId);
68     $institution = $this->institutionModel->getById($_SESSION['institution_id']);
69
70     $pdf = new \TCPDF('L', 'mm', 'A4', true, 'UTF-8');
71
72     $pdf->SetCreator('EcuAsist 2026');
73     $pdf->SetAuthor('Sistema de Asistencia');
74     $pdf->SetTitle('Reporte de Asistencia');
75
76     $pdf->setPrintHeader(false);
77     $pdf->setPrintFooter(false);
78     $pdf->SetMargins(10, 10, 10);
79     $pdf->SetAutoPageBreak(true, 10);
80
81     $pdf->AddPage();
82
83     $pdf->SetFont('helvetica', 'B', 16);
84     $pdf->Cell(0, 10, 'REPORTE DE ASISTENCIA', 0, 1, 'C');
85
86     $pdf->SetFont('helvetica', '', 10);
87     $pdf->Cell(0, 5, $institution['name'] ?? 'Institución Educativa', 0, 1, 'C');
88     $pdf->Ln(5);
89
90     $pdf->SetFont('helvetica', 'B', 11);
91     $pdf->Cell(50, 6, 'Curso:', 0, 0);
92     $pdf->SetFont('helvetica', '', 11);
93     $pdf->Cell(0, 6, html_entity_decode($course['name'], ENT_QUOTES, 'UTF-8'), 0, 1);
94
95     $pdf->SetFont('helvetica', 'B', 11);
96     $pdf->Cell(50, 6, 'Período:', 0, 0);
97     $pdf->SetFont('helvetica', '', 11);
98     $pdf->Cell(0, 6, date('d/m/Y', strtotime($startDate)) . ' al ' . date('d/m/Y', strtotime($endDate)), 0, 1);
99     $pdf->Ln(3);
100
101     $html = '<table border="1" cellpadding="4">
102     <thead>
103     <tr style="background-color: #007bff; color: white; font-weight: bold;">
104     <th width="16.66666%">Fecha</th>
105     <th width="16.66666%">Estudiante</th>
106     <th width="16.66666%">Asignatura</th>
107     <th width="16.66666%">Hora</th>
108     <th width="16.66666%">Estado</th>
109     <th width="16.66666%">Observación</th>
110     </tr>
111     </thead>
112     <tbody>';
113
114     foreach ($data as $row) {
115         $statusColor = $this->getStatusColor($row['status']);
116         $html .= '<tr>
117         <td>' . date('d/m/Y', strtotime($row['date'])) . '</td>
118         <td>' . $row['student_name'] . '</td>
119         <td>' . $row['subject_name'] . '</td>
120         <td>' . $row['hour_period'] . '</td>

```

```

121 <td style="background-color: ' . $statusColor . ';"> . ucfirst($row['status']) . '</td>
122 <td>' . ($row['observation'] ? '-' : '') . '</td>
123 </tr>';
124 }
125
126 $html .= '</tbody></table>';
127
128 $pdf->SetFont('helvetica','',9);
129
130 $pdf->writeHTML($html, true, false, true, false, '');
131
132 // Generar nombre de archivo con curso
133 $courseName = $this->sanitizeFilename($course['name']);
134 $filename = 'reporte_asistencia_' . $courseName . '_' . date('YmdHis') . '.pdf';
135
136 $pdf->Output($filename, 'D');
137 exit;
138 }
139
140 public function generateExcel() {
141 $courseId = (int)$_POST['course_id'];
142 $startDate = $_POST['start_date'];
143 $endDate = $_POST['end_date'];
144
145 $data = $this->attendanceModel->getReportData($courseId, $startDate, $endDate);
146 $course = $this->getCourseInfo($courseId);
147 $institution = $this->institutionModel->getById($_SESSION['institution_id']);
148
149 $spreadsheet = new Spreadsheet();
150 $sheet = $spreadsheet->getActiveSheet();
151
152 $sheet->setTitle('Asistencia');
153
154 // Encabezado
155 $sheet->mergeCells('A1:F1');
156 $sheet->setCellValue('A1', 'REPORTE DE ASISTENCIA');
157 $sheet->getStyle('A1')->getFont()->setBold(true)->setSize(16);
158 $sheet->getStyle('A1')->getAlignment()->setHorizontal(Alignment::HORIZONTAL_CENTER);
159
160 $sheet->mergeCells('A2:F2');
161 $sheet->setCellValue('A2', $institution['name'] ?? 'Institución Educativa');
162 $sheet->getStyle('A2')->getAlignment()->setHorizontal(Alignment::HORIZONTAL_CENTER);
163
164 $sheet->setCellValue('A4', 'Curso:');
165 $sheet->setCellValue('B4', html_entity_decode($course['name'], ENT_QUOTES, 'UTF-8'));
166 $sheet->getStyle('A4')->getFont()->setBold(true);
167
168 $sheet->setCellValue('A5', 'Período:');
169 $sheet->setCellValue('B5', date('d/m/Y', strtotime($startDate)) . ' al ' . date('d/m/Y',
    strtotime($endDate)));
170 $sheet->getStyle('A5')->getFont()->setBold(true);
171
172 // Cabecera de tabla
173 $row = 7;
174 $headers = ['Fecha', 'Estudiante', 'Asignatura', 'Hora', 'Estado', 'Observación'];
175 $col = 'A';
176
177 foreach ($headers as $header) {
178 $sheet->setCellValue($col . $row, $header);
179 $sheet->getStyle($col . $row)->getFont()->setBold(true);
180 $sheet->getStyle($col . $row)->getFill()

```

```

181 ->setFillType(Fill::FILL_SOLID)
182 ->getStartColor()->setRGB('007bff');
183 $sheet->getStyle($col . $row)->getFont()->getColor()->setRGB('FFFFFF');
184 $col++;
185 }
186
187 // Datos
188 $row = 8;
189 foreach ($data as $record) {
190 $sheet->setCellValue('A' . $row, date('d/m/Y', strtotime($record['date'])));
191 $sheet->setCellValue('B' . $row, $record['student_name']);
192 $sheet->setCellValue('C' . $row, $record['subject_name']);
193 $sheet->setCellValue('D' . $row, $record['hour_period']);
194 $sheet->setCellValue('E' . $row, ucfirst($record['status']));
195 $sheet->setCellValue('F' . $row, $record['observation'] ?: '-');
196
197 $statusColor = $this->getStatusColorHex($record['status']);
198 $sheet->getStyle('E' . $row)->getFill()
199 ->setFillType(Fill::FILL_SOLID)
200 ->getStartColor()->setRGB($statusColor);
201
202 $row++;
203 }
204
205 // Ajustar anchos
206 $sheet->getColumnDimension('A')->setWidth(12);
207 $sheet->getColumnDimension('B')->setWidth(25);
208 $sheet->getColumnDimension('C')->setWidth(20);
209 $sheet->getColumnDimension('D')->setWidth(15);
210 $sheet->getColumnDimension('E')->setWidth(12);
211 $sheet->getColumnDimension('F')->setWidth(25);
212
213 // Bordes
214 $styleArray = [
215 'borders' => [
216 'allBorders' => [
217 'borderStyle' => Border::BORDER_THIN,
218 ],
219 ],
220 ];
221 $sheet->getStyle('A7:F' . ($row - 1))->applyFromArray($styleArray);
222
223 $writer = new Xlsx($spreadsheet);
224
225 // Generar nombre de archivo con curso
226 $courseName = $this->sanitizeFilename($course['name']);
227 $filename = 'reporte_asistencia_' . $courseName . '_' . date('YmdHis') . '.xlsx';
228
229 header('Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
230 header('Content-Disposition: attachment;filename="' . $filename . '"');
231 header('Cache-Control: max-age=0');
232
233 $writer->save('php://output');
234 exit;
235 }
236
237 private function getCourseInfo($courseId) {
238 $courses = $this->courseModel->getAll();
239 foreach ($courses as $course) {
240 if ($course['id'] == $courseId) {

```

```

241 return $course;
242 }
243 }
244 return null;
245 }
246
247 private function getStatusColor($status) {
248     switch ($status) {
249         case 'presente': return '#d4edda';
250         case 'ausente': return '#f8d7da';
251         case 'tardanza': return '#fff3cd';
252         case 'justificado': return '#d1ecf1';
253         default: return '#ffffff';
254     }
255 }
256
257 private function getStatusColorHex($status) {
258     switch ($status) {
259         case 'presente': return '28a745';
260         case 'ausente': return 'dc3545';
261         case 'tardanza': return 'ffc107';
262         case 'justificado': return '17a2b8';
263         default: return 'FFFFFF';
264     }
265 }
266
267 private function sanitizeFilename($filename) {
268     // Decodificar HTML entities
269     $filename = html_entity_decode($filename, ENT_QUOTES, 'UTF-8');
270
271     // Reemplazar caracteres no permitidos en nombres de archivo
272     $filename = str_replace(['"', '/', '\\', ':', '*', '?', '<', '>', '|'], '', $filename);
273
274     // Reemplazar espacios con guiones bajos
275     $filename = str_replace(' ', '_', $filename);
276
277     // Reemplazar guiones múltiples
278     $filename = preg_replace('/-+/', '-', $filename);
279
280     // Limitar longitud (máximo 50 caracteres)
281     if (strlen($filename) > 50) {
282         $filename = substr($filename, 0, 50);
283     }
284
285     // Limpiar caracteres al inicio y final
286     $filename = trim($filename, '-_');
287
288     return $filename;
289 }
290 }

```

## 4.13 — RepresentativeController.php

controllers/RepresentativeController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/Representative.php';
4 require_once BASE_PATH . '/models/User.php';
5 require_once BASE_PATH . '/models/Attendance.php';
6
7 class RepresentativeController {
8     private $representativeModel;
9     private $userModel;
10    private $attendanceModel;
11
12    public function __construct() {
13        Security::requireLogin();
14    }
15
16    $db = new Database();
17    $this->representativeModel = new Representative($db);
18    $this->userModel = new User($db);
19    $this->attendanceModel = new Attendance($db);
20
21    public function manageRepresentatives() {
22        if (!Security::hasRole('autoridad')) {
23            die('Acceso denegado');
24        }
25
26        $representatives = $this->userModel->getByRole('representante');
27        $students = $this->userModel->getByRole('estudiante');
28
29        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
30            $repId = (int)$_POST['representative_id'];
31            $studentId = (int)$_POST['student_id'];
32            $relationship = Security::sanitize($_POST['relationship']);
33            $isPrimary = isset($_POST['is_primary']) ? 1 : 0;
34
35            $this->representativeModel->assignStudent($repId, $studentId, $relationship, $isPrimary);
36            header('Location: ?action=manage_representatives&success=1');
37            exit;
38        }
39
40        include BASE_PATH . '/views/representatives/manage.php';
41    }
42
43    public function myChildren() {
44        if (!Security::hasRole('representante')) {
45            die('Acceso denegado');
46        }
47
48        $children = $this->representativeModel->getStudentsByRepresentative($_SESSION['user_id']);
49
50        include BASE_PATH . '/views/representatives/my_children.php';
51    }
52
53    public function childAttendance() {
54        if (!Security::hasRole('representante')) {
55            die('Acceso denegado');
56        }
57
58        $studentId = (int)($_GET['student_id'] ?? 0);
59
60        // Verificar que el estudiante sea hijo del representante

```

```

61 $children = $this->representativeModel->getStudentsByRepresentative($_SESSION['user_id']);
62 $authorized = false;
63 $student = null;
64
65 foreach ($children as $child) {
66 if ($child['id'] == $studentId) {
67 $authorized = true;
68 $student = $child;
69 break;
70 }
71 }
72
73 if (!$authorized) {
74 die('No autorizado para ver este estudiante');
75 }
76
77 $attendances = $this->attendanceModel->getByStudent($studentId);
78
79 include BASE_PATH . '/views/representatives/child_attendance.php';
80 }
81
82 public function removeRelation() {
83 if (!Security::hasRole('autoridad')) {
84 die('Acceso denegado');
85 }
86
87 $repId = (int)($_GET['rep_id'] ?? 0);
88 $studentId = (int)($_GET['student_id'] ?? 0);
89
90 if ($repId && $studentId) {
91 if ($this->representativeModel->removeStudent($repId, $studentId)) {
92 header('Location: ?action=manage_representatives&removed=1');
93 exit;
94 }
95 }
96
97 header('Location: ?action=manage_representatives&error=1');
98 exit;
99 }
100 }

```

## 4.14 — ScheduleController.php

controllers/ScheduleController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/ClassSchedule.php';
4 require_once BASE_PATH . '/models/Course.php';
5 require_once BASE_PATH . '/models/Subject.php';
6 require_once BASE_PATH . '/models/User.php';
7 require_once BASE_PATH . '/models/SchoolYear.php';
8
9 class ScheduleController {
10 private $scheduleModel;
11 private $courseModel;
12 private $subjectModel;
13 private $userModel;
14 private $schoolYearModel;
15
16 public function __construct() {
17 Security::requireLogin();
18 if (!Security::hasRole('autoridad')) {
19 die('Acceso denegado');
20 }
21
22 $db = new Database();
23 $this->scheduleModel = new ClassSchedule($db);
24 $this->courseModel = new Course($db);
25 $this->subjectModel = new Subject($db);
26 $this->userModel = new User($db);
27 $this->schoolYearModel = new SchoolYear($db);
28 }
29
30 public function index() {
31 $courses = $this->courseModel->getAll();
32 include BASE_PATH . '/views/schedule/index.php';
33 }
34
35 public function manageCourse() {
36 $courseId = (int)($_GET['course_id'] ?? 0);
37
38 if (!$courseId) {
39 header('Location: ?action=schedules');
40 exit;
41 }
42
43 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
44 $activeYear = $this->schoolYearModel->getActive();
45
46 $data = [
47 ':course_id' => $courseId,
48 ':subject_id' => (int)$_POST['subject_id'],
49 ':teacher_id' => (int)$_POST['teacher_id'],
50 ':school_year_id' => $activeYear['id'],
51 ':day_of_week' => $_POST['day_of_week'],
52 ':period_number' => (int)$_POST['period_number']
53 ];
54
55 $result = $this->scheduleModel->create($data);
56
57 if ($result['success']) {
58 header('Location: ?action=manage_schedule&course_id=' . $courseId . '&success=1');
59 } else {
60 header('Location: ?action=manage_schedule&course_id=' . $courseId . '&error=' .
urlencode($result['message']));

```

```

61 }
62 exit;
63 }
64
65 $activeYear = $this->schoolYearModel->getActive();
66 $courses = $this->courseModel->getAll();
67 $course = array_filter($courses, fn($c) => $c['id'] == $courseId);
68 $course = reset($course);
69
70 $schedule = $this->scheduleModel->getByCourse($courseId, $activeYear['id']);
71 $subjects = $this->subjectModel->getAll();
72 $teachers = $this->userModel->getByRole('docente');
73
74 include BASE_PATH . '/views/schedule/manage.php';
75 }
76
77 public function deleteClass() {
78 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
79 $scheduleId = (int)$_POST['schedule_id'];
80 $courseId = (int)$_POST['course_id'];
81
82 $this->scheduleModel->delete($scheduleId);
83 header('Location: ?action=manage_schedule&course_id=' . $courseId . '&deleted=1');
84 exit;
85 }
86 }
87
88 public function getCourseSubjectsSchedule() {
89 if (!isset($_SESSION['user_id'])) {
90 header('Content-Type: application/json');
91 echo json_encode([]);
92 exit;
93 }
94
95 $courseId = (int)($_GET['course_id'] ?? 0);
96
97 if (!$courseId) {
98 header('Content-Type: application/json');
99 echo json_encode([]);
100 exit;
101 }
102
103 // Obtener asignaturas con sus docentes asignados al curso
104 $sql = "SELECT DISTINCT ta.subject_id, s.name as subject_name,
105 ta.teacher_id, CONCAT(u.last_name, ' ', u.first_name) as teacher_name
106 FROM teacher_assignments ta
107 INNER JOIN subjects s ON ta.subject_id = s.id
108 INNER JOIN users u ON ta.teacher_id = u.id
109 WHERE ta.course_id = :course_id
110 ORDER BY s.name";
111
112 $db = new Database();
113 $stmt = $db->connect()->prepare($sql);
114 $stmt->execute([':course_id' => $courseId]);
115 $subjects = $stmt->fetchAll(PDO::FETCH_ASSOC);
116
117 header('Content-Type: application/json');
118 echo json_encode($subjects);
119 exit;
120 }

```



```

121
122 public function checkScheduleConflict() {
123     if (!isset($_SESSION['user_id'])) {
124         header('Content-Type: application/json');
125         echo json_encode(['exists' => false]);
126         exit;
127     }
128
129     $courseId = (int)($_GET['course_id'] ?? 0);
130     $day = $_GET['day'] ?? '';
131     $period = (int)($_GET['period'] ?? 0);
132
133     if (!$courseId || !$day || !$period) {
134         header('Content-Type: application/json');
135         echo json_encode(['exists' => false]);
136         exit;
137     }
138
139     $db = new Database();
140     $activeYear = $this->schoolYearModel->getActive();
141
142     $sql = "SELECT s.name as subject_name, CONCAT(u.last_name, ' ', u.first_name) as teacher_name
143     FROM class_schedule cs
144     INNER JOIN subjects s ON cs.subject_id = s.id
145     INNER JOIN users u ON cs.teacher_id = u.id
146     WHERE cs.course_id = :course_id
147     AND cs.day_of_week = :day
148     AND cs.period_number = :period
149     AND cs.school_year_id = :school_year_id
150     LIMIT 1";
151
152     $stmt = $db->connect()->prepare($sql);
153     $stmt->execute([
154         ':course_id' => $courseId,
155         ':day' => $day,
156         ':period' => $period,
157         ':school_year_id' => $activeYear['id']
158     ]);
159
160     $existing = $stmt->fetch(PDO::FETCH_ASSOC);
161
162     $result = [
163         'exists' => $existing ? true : false,
164         'subject_name' => $existing['subject_name'] ?? null,
165         'teacher_name' => $existing['teacher_name'] ?? null
166     ];
167
168     header('Content-Type: application/json');
169     echo json_encode($result);
170     exit;
171 }
172 }

```

## 4.15 — SearchController.php

controllers/SearchController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/User.php';
4 require_once BASE_PATH . '/models/Course.php';
5 require_once BASE_PATH . '/models/Attendance.php';
6
7 class SearchController {
8     private $userModel;
9     private $courseModel;
10    private $attendanceModel;
11
12    public function __construct() {
13        Security::requireLogin();
14
15        $db = new Database();
16        $this->userModel = new User($db);
17        $this->courseModel = new Course($db);
18        $this->attendanceModel = new Attendance($db);
19    }
20
21    public function searchStudents() {
22        $query = $_GET['q'] ?? '';
23
24        if (strlen($query) < 2) {
25            echo json_encode([]);
26            exit;
27        }
28
29        $sql = "SELECT u.id, u.first_name, u.last_name, u.dni, u.email,
30 c.name as course_name
31 FROM users u
32 INNER JOIN user_roles ur ON u.id = ur.user_id
33 INNER JOIN roles r ON ur.role_id = r.id
34 LEFT JOIN course_students cs ON u.id = cs.student_id
35 LEFT JOIN courses c ON cs.course_id = c.id
36 WHERE r.name = 'estudiante'
37 AND u.institution_id = :institution_id
38 AND (u.first_name LIKE :query
39 OR u.last_name LIKE :query
40 OR u.dni LIKE :query
41 OR u.email LIKE :query)
42 LIMIT 10";
43
44        $db = new Database();
45        $stmt = $db->connect()->prepare($sql);
46        $stmt->execute([
47            ':institution_id' => $_SESSION['institution_id'],
48            ':query' => '%' . $query . '%'
49        ]);
50
51        header('Content-Type: application/json');
52        echo json_encode($stmt->fetchAll());
53        exit;
54    }
55
56    public function advancedAttendanceReport() {
57        if (!Security::hasRole(['autoridad', 'inspector'])) {
58            die('Acceso denegado');
59        }
60    }

```

```
61 $filters = [  
62 'course_id' => $_POST['course_id'] ?? null,  
63 'student_id' => $_POST['student_id'] ?? null,  
64 'subject_id' => $_POST['subject_id'] ?? null,  
65 'status' => $_POST['status'] ?? null,  
66 'start_date' => $_POST['start_date'] ?? null,  
67 'end_date' => $_POST['end_date'] ?? null  
68 ];  
69  
70 $results = $this->attendanceModel->getFiltered($filters);  
71  
72 include BASE_PATH . '/views/reports/advanced.php';  
73 }  
74 }
```

## 4.16 — StatsController.php

controllers/StatsController.php

PHP

```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/Attendance.php';
4 require_once BASE_PATH . '/models/Course.php';
5 require_once BASE_PATH . '/models/User.php';
6
7 class StatsController {
8     private $attendanceModel;
9     private $courseModel;
10    private $userModel;
11
12    public function __construct() {
13        Security::requireLogin();
14        if (!Security::hasRole('autoridad')) {
15            die('Acceso denegado');
16        }
17
18        $db = new Database();
19        $this->attendanceModel = new Attendance($db);
20        $this->courseModel = new Course($db);
21        $this->userModel = new User($db);
22    }
23
24    public function index() {
25        $startDate = $_GET['start_date'] ?? date('Y-m-01');
26        $endDate = $_GET['end_date'] ?? date('Y-m-d');
27
28        $stats = [
29            'overall' => $this->getOverallStats($startDate, $endDate),
30            'by_course' => $this->getStatsByCourse($startDate, $endDate),
31            'by_student' => $this->getTopAbsentStudents($startDate, $endDate),
32            'by_day' => $this->getStatsByDay($startDate, $endDate)
33        ];
34
35        include BASE_PATH . '/views/stats/index.php';
36    }
37
38    private function getOverallStats($startDate, $endDate) {
39        $sql = "SELECT
40            COUNT(*) as total_records,
41            COUNT(DISTINCT student_id) as total_students,
42            SUM(CASE WHEN status = 'presente' THEN 1 ELSE 0 END) as presente,
43            SUM(CASE WHEN status = 'ausente' THEN 1 ELSE 0 END) as ausente,
44            SUM(CASE WHEN status = 'tardanza' THEN 1 ELSE 0 END) as tardanza,
45            SUM(CASE WHEN status = 'justificado' THEN 1 ELSE 0 END) as justificado
46        FROM attendances
47        WHERE date BETWEEN :start_date AND :end_date";
48
49        $db = new Database();
50        $stmt = $db->connect()->prepare($sql);
51        $stmt->execute([':start_date' => $startDate, ':end_date' => $endDate]);
52        return $stmt->fetch();
53    }
54
55    private function getStatsByCourse($startDate, $endDate) {
56        $sql = "SELECT
57            c.name as course_name,
58            COUNT(*) as total,
59            SUM(CASE WHEN a.status = 'presente' THEN 1 ELSE 0 END) as presente,
60            SUM(CASE WHEN a.status = 'ausente' THEN 1 ELSE 0 END) as ausente,

```

```

61 ROUND(SUM(CASE WHEN a.status = 'presente' THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as percentage
62 FROM attendances a
63 INNER JOIN courses c ON a.course_id = c.id
64 WHERE a.date BETWEEN :start_date AND :end_date
65 GROUP BY c.id, c.name
66 ORDER BY percentage DESC";
67
68 $db = new Database();
69 $stmt = $db->connect()->prepare($sql);
70 $stmt->execute([':start_date' => $startDate, ':end_date' => $endDate]);
71 return $stmt->fetchAll();
72 }
73
74 private function getTopAbsentStudents($startDate, $endDate, $limit = 10) {
75 $sql = "SELECT
76 CONCAT(u.last_name, ' ', u.first_name) as student_name,
77 c.name as course_name,
78 COUNT(*) as total_absences
79 FROM attendances a
80 INNER JOIN users u ON a.student_id = u.id
81 INNER JOIN courses c ON a.course_id = c.id
82 WHERE a.status = 'ausente'
83 AND a.date BETWEEN :start_date AND :end_date
84 GROUP BY a.student_id, u.last_name, u.first_name, c.name
85 ORDER BY total_absences DESC
86 LIMIT :limit";
87
88 $db = new Database();
89 $stmt = $db->connect()->prepare($sql);
90 $stmt->bindValue(':start_date', $startDate);
91 $stmt->bindValue(':end_date', $endDate);
92 $stmt->bindValue(':limit', $limit, PDO::PARAM_INT);
93 $stmt->execute();
94 return $stmt->fetchAll();
95 }
96
97 private function getStatsByDay($startDate, $endDate) {
98 $sql = "SELECT
99 date,
100 COUNT(*) as total,
101 SUM(CASE WHEN status = 'presente' THEN 1 ELSE 0 END) as presente,
102 ROUND(SUM(CASE WHEN status = 'presente' THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as percentage
103 FROM attendances
104 WHERE date BETWEEN :start_date AND :end_date
105 GROUP BY date
106 ORDER BY date ASC";
107
108 $db = new Database();
109 $stmt = $db->connect()->prepare($sql);
110 $stmt->execute([':start_date' => $startDate, ':end_date' => $endDate]);
111 return $stmt->fetchAll();
112 }
113 }

```

## 4.17 — TutorController.php

controllers/TutorController.php

PHP

```

1 <?php
2 // ARCHIVO COMPLETO - Reemplaza TODO el archivo
3 // controllers/TutorController.php
4
5 require_once BASE_PATH . '/config/config.php';
6 require_once BASE_PATH . '/models/Attendance.php';
7 require_once BASE_PATH . '/models/Course.php';
8
9 class TutorController {
10 private $attendanceModel;
11 private $courseModel;
12
13 public function __construct() {
14 Security::requireLogin();
15 if (!Security::hasRole('docente')) {
16 die('Acceso denegado');
17 }
18 $db = new Database();
19 $this->attendanceModel = new Attendance($db);
20 $this->courseModel = new Course($db);
21 }
22
23 public function courseAttendance() {
24 $teacherId = $_SESSION['user_id'];
25 $courseId = $this->attendanceModel->getTutorCourseId($teacherId);
26
27 if (!$courseId) {
28 include BASE_PATH . '/views/tutor/no_tutor.php';
29 return;
30 }
31
32 $course = $this->courseModel->findById($courseId);
33 $subjects = $this->attendanceModel->getSubjectsByCourse($courseId);
34 $students = $this->attendanceModel->getStudentsByCourse($courseId);
35 $topAbsences = $this->attendanceModel->getTutorTopAbsences($courseId, 5);
36
37 // Leer filtros del GET
38 $filters = [];
39 if (!empty($_GET['subject_id'])) $filters['subject_id'] = (int)$_GET['subject_id'];
40 if (!empty($_GET['student_id'])) $filters['student_id'] = (int)$_GET['student_id'];
41 if (!empty($_GET['status'])) $filters['status'] = $_GET['status'];
42 if (!empty($_GET['start_date'])) $filters['start_date'] = $_GET['start_date'];
43 if (!empty($_GET['end_date'])) $filters['end_date'] = $_GET['end_date'];
44
45 $attendances = $this->attendanceModel->getTutorCourseAttendance($courseId, $filters);
46 $stats = $this->attendanceModel->getTutorCourseStats($courseId, $filters);
47
48 include BASE_PATH . '/views/tutor/course_attendance.php';
49 }
50
51 // Endpoint AJAX
52 public function ajax() {
53 header('Content-Type: application/json');
54
55 $teacherId = $_SESSION['user_id'];
56 $courseId = $this->attendanceModel->getTutorCourseId($teacherId);
57
58 if (!$courseId) {
59 echo json_encode(['error' => 'No es tutor']);
60 exit;

```

[illegible]

[illegible]

## 4.18 — UserController.php

**controllers/UserController.php**

PHP



```

1 <?php
2 require_once BASE_PATH . '/config/config.php';
3 require_once BASE_PATH . '/models/User.php';
4 require_once BASE_PATH . '/models/Role.php';
5
6 class UserController {
7     private $userModel;
8     private $roleModel;
9
10    public function __construct() {
11        Security::requireLogin();
12        if (!Security::hasRole('autoridad')) {
13            die('Acceso denegado');
14        }
15
16        $db = new Database();
17        $this->userModel = new User($db);
18        $this->roleModel = new Role($db);
19    }
20
21    public function index() {
22        $users = $this->userModel->getAll();
23        $roles = $this->roleModel->getAll();
24        include BASE_PATH . '/views/users/index.php';
25    }
26
27    public function assignRole() {
28        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
29            $userId = (int)$_POST['user_id'];
30            $roleId = (int)$_POST['role_id'];
31
32            $this->userModel->assignRole($userId, $roleId);
33            $filter = isset($_GET['filter_role']) ? '&filter_role=' . $_GET['filter_role'] : '';
34            header('Location: ?action=users&success=1' . $filter);
35            exit;
36        }
37    }
38
39    public function removeRole() {
40        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
41            $userId = (int)$_POST['user_id'];
42            $roleId = (int)$_POST['role_id'];
43
44            // Verificar si el rol es "docente"
45            $db = new Database();
46            $stmt = $db->connect()->prepare("SELECT name FROM roles WHERE id = :role_id");
47            $stmt->execute([':role_id' => $roleId]);
48            $role = $stmt->fetch();
49
50            if ($role && $role['name'] === 'docente') {
51                // Eliminar asignaciones de materias
52                $stmt = $db->connect()->prepare("DELETE FROM teacher_assignments WHERE teacher_id = :user_id");
53                $stmt->execute([':user_id' => $userId]);
54            }
55
56            $this->userModel->removeRole($userId, $roleId);
57            $filter = isset($_GET['filter_role']) ? '&filter_role=' . $_GET['filter_role'] : '';
58            header('Location: ?action=users&removed=1' . $filter);
59            exit;
60        }

```

```

61 }
62
63 public function create() {
64 if ($_SERVER['REQUEST_METHOD'] === 'GET') {
65 $roles = $this->roleModel->getAll();
66 include BASE_PATH . '/views/users/create.php';
67 return;
68 }
69
70 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
71 $errors = [];
72
73 // Validaciones
74 if (empty($_POST['username'])) {
75 $errors[] = "El nombre de usuario es obligatorio";
76 }
77 if (empty($_POST['email']) || !filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
78 $errors[] = "Email inválido";
79 }
80 if (empty($_POST['password']) || strlen($_POST['password']) < 6) {
81 $errors[] = "La contraseña debe tener al menos 6 caracteres";
82 }
83 if ($_POST['password'] !== $_POST['confirm_password']) {
84 $errors[] = "Las contraseñas no coinciden";
85 }
86 if (empty($_POST['first_name'])) {
87 $errors[] = "El nombre es obligatorio";
88 }
89 if (empty($_POST['last_name'])) {
90 $errors[] = "El apellido es obligatorio";
91 }
92
93 // Verificar username único
94 if ($this->userModel->findByUsername($_POST['username'])) {
95 $errors[] = "El nombre de usuario ya está registrado";
96 }
97
98 // Verificar email único
99 if ($this->userModel->findByEmail($_POST['email'])) {
100 $errors[] = "El email ya está registrado";
101 }
102
103 if (!empty($errors)) {
104 $roles = $this->roleModel->getAll();
105 include BASE_PATH . '/views/users/create.php';
106 return;
107 }
108
109 // Crear usuario
110 $userData = [
111 'institution_id' => $_SESSION['institution_id'],
112 'username' => Security::sanitize($_POST['username']),
113 'email' => Security::sanitize($_POST['email']),
114 'password' => $_POST['password'],
115 'first_name' => Security::sanitize($_POST['first_name']),
116 'last_name' => Security::sanitize($_POST['last_name']),
117 'dni' => !empty($_POST['dni']) ? Security::sanitize($_POST['dni']) : null,
118 'phone' => !empty($_POST['phone']) ? Security::sanitize($_POST['phone']) : null
119 ];
120

```

```

121 if ($this->userModel->create($userData)) {
122 // Obtener ID del usuario recién creado
123 $newUserId = $this->userModel->db->lastInsertId();
124
125 // Asignar roles si se seleccionaron
126 if (!empty($_POST['roles'])) {
127 foreach ($_POST['roles'] as $roleId) {
128 $this->userModel->assignRole($newUserId, (int)$roleId);
129 }
130 }
131
132 header('Location: ?action=users&created=1');
133 exit;
134 } else {
135 $errors[] = "Error al crear el usuario";
136 $roles = $this->roleModel->getAll();
137 include BASE_PATH . '/views/users/create.php';
138 }
139 }
140 }
141
142 public function edit() {
143 $userId = (int)$_GET['id'];
144 $user = $this->userModel->findById($userId);
145
146 if (!$user) {
147 header('Location: ?action=users&error=not_found');
148 exit;
149 }
150
151 // Verificar que pertenece a la misma institución
152 if ($user['institution_id'] != $_SESSION['institution_id']) {
153 die('Acceso denegado');
154 }
155
156 if ($_SERVER['REQUEST_METHOD'] === 'GET') {
157 $roles = $this->roleModel->getAll();
158 $userRoles = $this->userModel->getUserRoleIds($userId);
159 include BASE_PATH . '/views/users/edit.php';
160 return;
161 }
162
163 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
164 $errors = [];
165
166 // Validaciones
167 if (empty($_POST['username'])) {
168 $errors[] = "El nombre de usuario es obligatorio";
169 }
170 if (empty($_POST['email']) || !filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
171 $errors[] = "Email inválido";
172 }
173 if (empty($_POST['first_name'])) {
174 $errors[] = "El nombre es obligatorio";
175 }
176 if (empty($_POST['last_name'])) {
177 $errors[] = "El apellido es obligatorio";
178 }
179
180 // Verificar username único (excepto el actual)

```

```

181 $existingUser = $this->userModel->findByUsername($_POST['username']);
182 if ($existingUser && $existingUser['id'] != $userId) {
183 $errors[] = "El nombre de usuario ya está registrado";
184 }
185
186 // Verificar email único (excepto el actual)
187 $existingEmail = $this->userModel->findByEmail($_POST['email']);
188 if ($existingEmail && $existingEmail['id'] != $userId) {
189 $errors[] = "El email ya está registrado";
190 }
191
192 // Si hay nueva contraseña, validarla
193 if (!empty($_POST['password'])) {
194 if (strlen($_POST['password']) < 6) {
195 $errors[] = "La contraseña debe tener al menos 6 caracteres";
196 }
197 if ($_POST['password'] != $_POST['confirm_password']) {
198 $errors[] = "Las contraseñas no coinciden";
199 }
200 }
201
202 if (!empty($errors)) {
203 $roles = $this->roleModel->getAll();
204 $userRoles = $this->userModel->getUserRoleIds($userId);
205 include BASE_PATH . '/views/users/edit.php';
206 return;
207 }
208
209 // Actualizar usuario
210 $updateData = [
211 'id' => $userId,
212 'username' => Security::sanitize($_POST['username']),
213 'email' => Security::sanitize($_POST['email']),
214 'first_name' => Security::sanitize($_POST['first_name']),
215 'last_name' => Security::sanitize($_POST['last_name']),
216 'dni' => !empty($_POST['dni']) ? Security::sanitize($_POST['dni']) : null,
217 'phone' => !empty($_POST['phone']) ? Security::sanitize($_POST['phone']) : null
218 ];
219
220 // Solo actualizar contraseña si se proporcionó una nueva
221 if (!empty($_POST['password'])) {
222 $updateData['password'] = $_POST['password'];
223 }
224
225 if ($this->userModel->update($updateData)) {
226 header('Location: ?action=users&updated=1');
227 exit;
228 } else {
229 $errors[] = "Error al actualizar el usuario";
230 $roles = $this->roleModel->getAll();
231 $userRoles = $this->userModel->getUserRoleIds($userId);
232 include BASE_PATH . '/views/users/edit.php';
233 }
234 }
235 }
236
237 public function delete() {
238 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
239 $userId = (int)$_POST['user_id'];
240 $user = $this->userModel->findById($userId);

```

```

241
242 if (!$user) {
243 header('Location: ?action=users&error=not_found');
244 exit;
245 }
246
247 // Verificar que pertenece a la misma institución
248 if ($user['institution_id'] != $_SESSION['institution_id']) {
249 die('Acceso denegado');
250 }
251
252 // No permitir eliminar al propio usuario
253 if ($userId == $_SESSION['user_id']) {
254 header('Location: ?action=users&error=self_delete');
255 exit;
256 }
257
258 // Verificar si tiene asistencias registradas
259 $db = new Database();
260 $stmt = $db->connect()->prepare("SELECT COUNT(*) as count FROM attendances WHERE student_id = :user_id");
261 $stmt->execute([':user_id' => $userId]);
262 $result = $stmt->fetch();
263
264 if ($result['count'] > 0) {
265 // Soft delete - desactivar en lugar de eliminar
266 if ($this->userModel->deactivate($userId)) {
267 header('Location: ?action=users&deactivated=1');
268 exit;
269 }
270 } else {
271 // Hard delete - eliminar completamente
272 if ($this->userModel->delete($userId)) {
273 header('Location: ?action=users&deleted=1');
274 exit;
275 }
276 }
277
278 header('Location: ?action=users&error=delete_failed');
279 exit;
280 }
281 }
282 }

```

## 5. MODULO MODELS (14 archivos)

### 5.1 — Attendance.php

models/Attendance.php

PHP

```

1 <?php
2 // ARCHIVO COMPLETO - Reemplaza TODO el archivo
3 // models/Attendance.php
4
5 class Attendance {
6     private $db;
7
8     public function __construct($database) {
9         $this->db = $database->connect();
10    }
11
12    public function create($data) {
13        $checkSql = "SELECT id FROM attendances
14        WHERE student_id = :student_id
15        AND course_id = :course_id
16        AND subject_id = :subject_id
17        AND date = :date
18        AND hour_period = :hour_period";
19        $checkStmt = $this->db->prepare($checkSql);
20        $checkStmt->execute([
21            ':student_id' => $data[':student_id'],
22            ':course_id' => $data[':course_id'],
23            ':subject_id' => $data[':subject_id'],
24            ':date' => $data[':date'],
25            ':hour_period' => $data[':hour_period']
26        ]);
27        $existing = $checkStmt->fetch();
28
29        if ($existing) {
30            $updateSql = "UPDATE attendances
31            SET status = :status, observation = :observation, updated_at = NOW()
32            WHERE id = :id";
33            $updateStmt = $this->db->prepare($updateSql);
34            return $updateStmt->execute([
35                ':status' => $data[':status'],
36                ':observation' => $data[':observation'],
37                ':id' => $existing['id']
38            ]);
39        }
40
41        $sql = "INSERT INTO attendances
42        (student_id, course_id, subject_id, teacher_id, school_year_id, shift_id, date, hour_period, status,
43        observation)
44        VALUES (:student_id, :course_id, :subject_id, :teacher_id, :school_year_id, :shift_id, :date, :hour_period,
45        :status, :observation)";
46        $stmt = $this->db->prepare($sql);
47        return $stmt->execute($data);
48    }
49
50    public function canEdit($attendanceId) {
51        $sql = "SELECT created_at FROM attendances WHERE id = :id";
52        $stmt = $this->db->prepare($sql);
53        $stmt->execute([':id' => $attendanceId]);
54        $record = $stmt->fetch();
55        if (!$record) return false;
56        return (time() - strtotime($record['created_at'])) <= (EDIT_ATTENDANCE_HOURS * 3600);
57    }
58
59    public function update($id, $data) {
60        $sql = "UPDATE attendances
61        SET status = :status, observation = :observation, updated_at = NOW()
62        WHERE id = :id";

```

```

61 $stmt = $this->db->prepare($sql);
62 return $stmt->execute([':id' => $id, ':status' => $data['status'], ':observation' => $data['observation']]);
63 }
64
65 public function getByCourse($courseId, $date) {
66 $sql = "SELECT a.*, u.first_name, u.last_name, s.name as subject_name,
67 CONCAT(t.first_name, ' ', t.last_name) as teacher_name
68 FROM attendances a
69 INNER JOIN users u ON a.student_id = u.id
70 INNER JOIN subjects s ON a.subject_id = s.id
71 INNER JOIN users t ON a.teacher_id = t.id
72 WHERE a.course_id = :course_id AND a.date = :date
73 ORDER BY u.last_name, u.first_name";
74 $stmt = $this->db->prepare($sql);
75 $stmt->execute([':course_id' => $courseId, ':date' => $date]);
76 return $stmt->fetchAll();
77 }
78
79 public function getByStudent($studentId) {
80 $sql = "SELECT a.*, s.name as subject_name, c.name as course_name, sh.name as shift_name
81 FROM attendances a
82 INNER JOIN subjects s ON a.subject_id = s.id
83 INNER JOIN courses c ON a.course_id = c.id
84 INNER JOIN shifts sh ON a.shift_id = sh.id
85 WHERE a.student_id = :student_id
86 ORDER BY a.date DESC, a.hour_period
87 LIMIT 100";
88 $stmt = $this->db->prepare($sql);
89 $stmt->execute([':student_id' => $studentId]);
90 return $stmt->fetchAll();
91 }
92
93 public function getReportData($courseId, $startDate, $endDate) {
94 $sql = "SELECT a.*,
95 CONCAT(u.last_name, ' ', u.first_name) as student_name,
96 s.name as subject_name
97 FROM attendances a
98 INNER JOIN users u ON a.student_id = u.id
99 INNER JOIN subjects s ON a.subject_id = s.id
100 WHERE a.course_id = :course_id
101 AND a.date BETWEEN :start_date AND :end_date
102 ORDER BY a.date, u.last_name, u.first_name";
103 $stmt = $this->db->prepare($sql);
104 $stmt->execute([':course_id' => $courseId, ':start_date' => $startDate, ':end_date' => $endDate]);
105 return $stmt->fetchAll();
106 }
107
108 public function getStatsByCourse($courseId, $startDate, $endDate) {
109 $sql = "SELECT
110 COUNT(*) as total,
111 SUM(CASE WHEN status = 'presente' THEN 1 ELSE 0 END) as presente,
112 SUM(CASE WHEN status = 'ausente' THEN 1 ELSE 0 END) as ausente,
113 SUM(CASE WHEN status = 'tardanza' THEN 1 ELSE 0 END) as tardanza,
114 SUM(CASE WHEN status = 'justificado' THEN 1 ELSE 0 END) as justificado
115 FROM attendances
116 WHERE course_id = :course_id AND date BETWEEN :start_date AND :end_date";
117 $stmt = $this->db->prepare($sql);
118 $stmt->execute([':course_id' => $courseId, ':start_date' => $startDate, ':end_date' => $endDate]);
119 return $stmt->fetch();
120 }

```



[illegible]

```

181 return $stmt->fetchAll();
182 }
183
184 // Estudiantes del curso (para filtro)
185 public function getStudentsByCourse($courseId) {
186 $sql = "SELECT u.id, CONCAT(u.last_name, ' ', u.first_name) as full_name
187 FROM users u
188 INNER JOIN course_students cs ON u.id = cs.student_id
189 WHERE cs.course_id = :course_id
190 ORDER BY u.last_name, u.first_name";
191 $stmt = $this->db->prepare($sql);
192 $stmt->execute([':course_id' => $courseId]);
193 return $stmt->fetchAll();
194 }
195
196 // Asistencias del curso con filtros (vista del tutor)
197 public function getTutorCourseAttendance($courseId, $filters = []) {
198 $sql = "SELECT
199 a.id, a.date, a.hour_period, a.status, a.observation,
200 CONCAT(u.last_name, ' ', u.first_name) as student_name,
201 u.id as student_id,
202 s.name as subject_name,
203 s.id as subject_id,
204 CONCAT(t.last_name, ' ', t.first_name) as teacher_name
205 FROM attendances a
206 INNER JOIN users u ON a.student_id = u.id
207 INNER JOIN subjects s ON a.subject_id = s.id
208 INNER JOIN users t ON a.teacher_id = t.id
209 WHERE a.course_id = :course_id";
210 $params = [':course_id' => $courseId];
211
212 if (!empty($filters['subject_id'])) { $sql .= " AND a.subject_id = :subject_id"; $params[':subject_id'] =
213 $filters['subject_id']; }
214 if (!empty($filters['student_id'])) { $sql .= " AND a.student_id = :student_id"; $params[':student_id'] =
215 $filters['student_id']; }
216 if (!empty($filters['status'])) { $sql .= " AND a.status = :status"; $params[':status'] =
217 $filters['status']; }
218 if (!empty($filters['start_date'])) { $sql .= " AND a.date >= :start_date"; $params[':start_date'] =
219 $filters['start_date']; }
220 if (!empty($filters['end_date'])) { $sql .= " AND a.date <= :end_date"; $params[':end_date'] =
221 $filters['end_date']; }
222
223 $sql .= " ORDER BY a.date DESC, u.last_name, u.first_name, s.name";
224 $stmt = $this->db->prepare($sql);
225 $stmt->execute($params);
226 return $stmt->fetchAll();
227 }
228
229 // Estadísticas globales del curso (vista del tutor)
230 public function getTutorCourseStats($courseId, $filters = []) {
231 $sql = "SELECT
232 COUNT(*) as total,
233 SUM(CASE WHEN status = 'presente' THEN 1 ELSE 0 END) as presente,
234 SUM(CASE WHEN status = 'ausente' THEN 1 ELSE 0 END) as ausente,
235 SUM(CASE WHEN status = 'tardanza' THEN 1 ELSE 0 END) as tardanza,
236 SUM(CASE WHEN status = 'justificado' THEN 1 ELSE 0 END) as justificado
237 FROM attendances a
238 WHERE a.course_id = :course_id";
239 $params = [':course_id' => $courseId];
240
241 if (!empty($filters['subject_id'])) { $sql .= " AND a.subject_id = :subject_id"; $params[':subject_id'] =
242 $filters['subject_id']; }
243 if (!empty($filters['student_id'])) { $sql .= " AND a.student_id = :student_id"; $params[':student_id'] =
244 $filters['student_id']; }
245 if (!empty($filters['start_date'])) { $sql .= " AND a.date >= :start_date"; $params[':start_date'] =
246 $filters['start_date']; }
247 if (!empty($filters['end_date'])) { $sql .= " AND a.date <= :end_date"; $params[':end_date'] =
248 $filters['end_date']; }
249
250 $stmt = $this->db->prepare($sql);
251 $stmt->execute($params);
252 return $stmt->fetch();
253 }

```

```

241 $stmt = $this->db->prepare($sql);
242 $stmt->execute($params);
243 return $stmt->fetch();
244 }
245
246 // Top estudiantes con más ausencias (vista del tutor)
247 public function getTutorTopAbsences($courseId, $limit = 10) {
248     $sql = "SELECT CONCAT(u.last_name, ' ', u.first_name) as student_name,
249     COUNT(*) as total_ausencias
250     FROM attendances a
251     INNER JOIN users u ON a.student_id = u.id
252     WHERE a.course_id = :course_id AND a.status = 'ausente'
253     GROUP BY a.student_id, u.last_name, u.first_name
254     ORDER BY total_ausencias DESC
255     LIMIT :lim";
256     $stmt = $this->db->prepare($sql);
257     $stmt->bindValue(':course_id', $courseId, PDO::PARAM_INT);
258     $stmt->bindValue(':lim', $limit, PDO::PARAM_INT);
259     $stmt->execute();
260     return $stmt->fetchAll();
261 }
262
263 // Ausencias sin justificación pendiente o aprobada (para el formulario de justificación)
264 public function getUnjustifiedAbsences($studentId) {
265     $sql = "SELECT a.id, a.date, a.hour_period, a.status,
266     s.name as subject_name,
267     c.name as course_name
268     FROM attendances a
269     INNER JOIN subjects s ON a.subject_id = s.id
270     INNER JOIN courses c ON a.course_id = c.id
271     WHERE a.student_id = :student_id
272     AND a.status = 'ausente'
273     AND a.id NOT IN (
274     SELECT attendance_id FROM justifications
275     WHERE attendance_id IS NOT NULL
276     AND status IN ('pendiente','aprobado')
277     )
278     ORDER BY a.date DESC, a.hour_period ASC";
279     $stmt = $this->db->prepare($sql);
280     $stmt->execute([':student_id' => $studentId]);
281     return $stmt->fetchAll();
282 }
283 }

```

## 5.2 — ClassSchedule.php

models/ClassSchedule.php

PHP

```

1 <?php
2 class ClassSchedule {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function create($data) {
10        // Verificar si ya existe una clase en ese horario
11        $checkSql = "SELECT s.name as subject_name,
12        CONCAT(u.last_name, ' ', u.first_name) as teacher_name
13        FROM class_schedule cs
14        INNER JOIN subjects s ON cs.subject_id = s.id
15        INNER JOIN users u ON cs.teacher_id = u.id
16        WHERE cs.course_id = :course_id
17        AND cs.day_of_week = :day_of_week
18        AND cs.period_number = :period_number
19        AND cs.school_year_id = :school_year_id";
20
21        $checkStmt = $this->db->prepare($checkSql);
22        $checkStmt->execute([
23            ':course_id' => $data[':course_id'],
24            ':day_of_week' => $data[':day_of_week'],
25            ':period_number' => $data[':period_number'],
26            ':school_year_id' => $data[':school_year_id']
27        ]);
28
29        $existing = $checkStmt->fetch();
30
31        if ($existing) {
32            return [
33                'success' => false,
34                'message' => 'Ya existe ' . $existing['subject_name'] . ' con ' . $existing['teacher_name'] . ' en este
horario'
35            ];
36        }
37
38        $sql = "INSERT INTO class_schedule
39        (course_id, subject_id, teacher_id, school_year_id, day_of_week, period_number)
40        VALUES (:course_id, :subject_id, :teacher_id, :school_year_id, :day_of_week, :period_number)";
41
42        $stmt = $this->db->prepare($sql);
43        $result = $stmt->execute($data);
44
45        return [
46            'success' => $result,
47            'message' => 'Clase agregada correctamente'
48        ];
49    }
50
51    public function getByCourse($courseId, $schoolYearId) {
52        $sql = "SELECT cs.*,
53        s.name as subject_name,
54        CONCAT(u.last_name, ' ', u.first_name) as teacher_name
55        FROM class_schedule cs
56        INNER JOIN subjects s ON cs.subject_id = s.id
57        INNER JOIN users u ON cs.teacher_id = u.id
58        WHERE cs.course_id = :course_id
59        AND cs.school_year_id = :school_year_id
60        ORDER BY cs.day_of_week, cs.period_number";

```

```

61
62 $stmt = $this->db->prepare($sql);
63 $stmt->execute([
64 ':course_id' => $courseId,
65 ':school_year_id' => $schoolYearId
66 ]);
67 return $stmt->fetchAll();
68 }
69
70 public function getTeacherScheduleToday($teacherId, $schoolYearId) {
71 $dayName = $this->getCurrentDayName();
72
73 $sql = "SELECT cs.*,
74 c.name as course_name,
75 c.grade_level,
76 s.name as subject_name
77 FROM class_schedule cs
78 INNER JOIN courses c ON cs.course_id = c.id
79 INNER JOIN subjects s ON cs.subject_id = s.id
80 WHERE cs.teacher_id = :teacher_id
81 AND cs.school_year_id = :school_year_id
82 AND cs.day_of_week = :day_of_week
83 ORDER BY cs.period_number";
84
85 $stmt = $this->db->prepare($sql);
86 $stmt->execute([
87 ':teacher_id' => $teacherId,
88 ':school_year_id' => $schoolYearId,
89 ':day_of_week' => $dayName
90 ]);
91 return $stmt->fetchAll();
92 }
93
94 public function delete($id) {
95 $sql = "DELETE FROM class_schedule WHERE id = :id";
96 $stmt = $this->db->prepare($sql);
97 return $stmt->execute([':id' => $id]);
98 }
99
100 private function getCurrentDayName() {
101 // Forzar zona horaria Ecuador
102 $tz = new DateTimeZone('America/Guayaquil');
103 $now = new DateTime('now', $tz);
104 $dayNumber = (int)$now->format('N'); // 1=Lunes, 7=Domingo
105
106 $days = [
107 1 => 'lunes',
108 2 => 'martes',
109 3 => 'miercoles',
110 4 => 'jueves',
111 5 => 'viernes',
112 6 => 'sabado'
113 ];
114
115 return $days[$dayNumber] ?? 'lunes';
116 }
117 }

```

## 5.3 — Course.php

models/Course.php

PHP

```

1 <?php
2 class Course {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function getAll() {
10        $sql = "SELECT c.*, s.name as shift_name, sy.name as year_name
11        FROM courses c
12        INNER JOIN shifts s ON c.shift_id = s.id
13        INNER JOIN school_years sy ON c.school_year_id = sy.id
14        WHERE c.institution_id = :institution_id
15        ORDER BY c.name";
16
17        $stmt = $this->db->prepare($sql);
18        $stmt->execute([':institution_id' => $_SESSION['institution_id']]);
19        return $stmt->fetchAll();
20    }
21
22    public function create($data) {
23        $sql = "INSERT INTO courses (institution_id, school_year_id, name, grade_level, parallel, shift_id)
24        VALUES (:institution_id, :school_year_id, :name, :grade_level, :parallel, :shift_id)";
25
26        $stmt = $this->db->prepare($sql);
27        return $stmt->execute($data);
28    }
29
30    public function getStudents($courseId) {
31        $sql = "SELECT u.id, u.first_name, u.last_name, u.dni
32        FROM users u
33        INNER JOIN course_students cs ON u.id = cs.student_id
34        WHERE cs.course_id = :course_id
35        ORDER BY u.last_name, u.first_name";
36
37        $stmt = $this->db->prepare($sql);
38        $stmt->execute([':course_id' => $courseId]);
39        return $stmt->fetchAll();
40    }
41
42    public function enrollStudent($courseId, $studentId, $schoolYearId) {
43        // Primero verificar si ya está matriculado en otro curso del mismo año
44        $checkSql = "SELECT COUNT(*) as count FROM course_students
45        WHERE student_id = :student_id AND school_year_id = :school_year_id";
46        $checkStmt = $this->db->prepare($checkSql);
47        $checkStmt->execute([
48            ':student_id' => $studentId,
49            ':school_year_id' => $schoolYearId
50        ]);
51        $result = $checkStmt->fetch();
52
53        if ($result['count'] > 0) {
54            return false; // Ya está matriculado
55        }
56
57        $sql = "INSERT INTO course_students (student_id, course_id, school_year_id, enrollment_date)
58        VALUES (:student_id, :course_id, :school_year_id, CURDATE())";
59
60        $stmt = $this->db->prepare($sql);

```

```

61 return $stmt->execute([
62   ':student_id' => $studentId,
63   ':course_id' => $courseId,
64   ':school_year_id' => $schoolYearId
65 ]);
66 }
67
68 public function getEnrolledStudents($courseId) {
69   $sql = "SELECT u.id, u.first_name, u.last_name, u.dni, cs.enrollment_date
70 FROM users u
71 INNER JOIN course_students cs ON u.id = cs.student_id
72 WHERE cs.course_id = :course_id
73 ORDER BY u.last_name, u.first_name";
74
75   $stmt = $this->db->prepare($sql);
76   $stmt->execute([':course_id' => $courseId]);
77   return $stmt->fetchAll();
78 }
79
80 public function findById($id) {
81   $sql = "SELECT c.*, s.name as shift_name
82 FROM courses c
83 INNER JOIN shifts s ON c.shift_id = s.id
84 WHERE c.id = :id";
85
86   $stmt = $this->db->prepare($sql);
87   $stmt->execute([':id' => $id]);
88   return $stmt->fetch();
89 }
90
91 public function update($data) {
92   $sql = "UPDATE courses SET
93 name = :name,
94 grade_level = :grade_level,
95 parallel = :parallel,
96 shift_id = :shift_id,
97 updated_at = NOW()
98 WHERE id = :id";
99
100   $stmt = $this->db->prepare($sql);
101   return $stmt->execute($data);
102 }
103
104 public function delete($id) {
105   // Eliminar relaciones primero
106   $sql = "DELETE FROM course_students WHERE course_id = :id";
107   $stmt = $this->db->prepare($sql);
108   $stmt->execute([':id' => $id]);
109
110   // Eliminar horarios
111   $sql = "DELETE FROM class_schedule WHERE course_id = :id";
112   $stmt = $this->db->prepare($sql);
113   $stmt->execute([':id' => $id]);
114
115   // Eliminar curso
116   $sql = "DELETE FROM courses WHERE id = :id";
117   $stmt = $this->db->prepare($sql);
118   return $stmt->execute([':id' => $id]);
119 }
120

```

```
121 public function unenrollStudent($studentId, $schoolYearId) {
122     $sql = "DELETE FROM course_students
123     WHERE student_id = :student_id AND school_year_id = :school_year_id";
124
125     $stmt = $this->db->prepare($sql);
126     return $stmt->execute([
127         ':student_id' => $studentId,
128         ':school_year_id' => $schoolYearId
129     ]);
130 }
131 }
```

## 5.4 — Institution.php

models/Institution.php

PHP



```

1 <?php
2 class Institution {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function getById($id) {
10        $sql = "SELECT * FROM institutions WHERE id = :id";
11        $stmt = $this->db->prepare($sql);
12        $stmt->execute([':id' => $id]);
13        return $stmt->fetch();
14    }
15
16    public function getAll() {
17        $sql = "SELECT * FROM institutions ORDER BY name";
18        $stmt = $this->db->query($sql);
19        return $stmt->fetchAll();
20    }
21
22    public function update($id, $data) {
23        $sql = "UPDATE institutions SET
24            name = :name,
25            address = :address,
26            province = :province,
27            city = :city,
28            phone = :phone,
29            email = :email,
30            director_name = :director_name,
31            amie_code = :amie_code,
32            website = :website,
33            logo_path = :logo_path,
34            updated_at = NOW()
35            WHERE id = :id";
36
37        // Obtener logo actual si no se proporciona uno nuevo
38        $currentInstitution = $this->getById($id);
39        $logoPath = $data['logo_path'] ?? $currentInstitution['logo_path'];
40
41        $stmt = $this->db->prepare($sql);
42        return $stmt->execute([
43            ':id' => $id,
44            ':name' => $data['name'],
45            ':address' => $data['address'],
46            ':province' => $data['province'],
47            ':city' => $data['city'],
48            ':phone' => $data['phone'],
49            ':email' => $data['email'],
50            ':director_name' => $data['director_name'],
51            ':amie_code' => $data['amie_code'],
52            ':website' => $data['website'],
53            ':logo_path' => $logoPath
54        ]);
55    }
56
57    public function getShifts($institutionId) {
58        $sql = "SELECT s.id, s.name,
59            GROUP_CONCAT(s.name ORDER BY s.name) as shift_names
60        FROM shifts s

```

```

61 INNER JOIN institution_shifts ins ON s.id = ins.shift_id
62 WHERE ins.institution_id = :institution_id
63 GROUP BY ins.institution_id";
64
65 $stmt = $this->db->prepare($sql);
66 $stmt->execute([':institution_id' => $institutionId]);
67 return $stmt->fetch();
68 }
69 }

```

## 5.5 — InstitutionShift.php

models/InstitutionShift.php

PHP

```

1 <?php
2 class InstitutionShift {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function assign($data) {
10        $sql = "INSERT INTO institution_shifts (institution_id, shift_id)
11        VALUES (:institution_id, :shift_id)
12        ON DUPLICATE KEY UPDATE institution_id = institution_id";
13
14        $stmt = $this->db->prepare($sql);
15        return $stmt->execute($data);
16    }
17
18    public function remove($institutionId, $shiftId) {
19        $sql = "DELETE FROM institution_shifts
20        WHERE institution_id = :institution_id AND shift_id = :shift_id";
21
22        $stmt = $this->db->prepare($sql);
23        return $stmt->execute([
24            ':institution_id' => $institutionId,
25            ':shift_id' => $shiftId
26        ]);
27    }
28
29    public function getByInstitution($institutionId) {
30        $sql = "SELECT s.id, s.name
31        FROM shifts s
32        INNER JOIN institution_shifts ins ON s.id = ins.shift_id
33        WHERE ins.institution_id = :institution_id
34        ORDER BY s.name";
35
36        $stmt = $this->db->prepare($sql);
37        $stmt->execute([':institution_id' => $institutionId]);
38        return $stmt->fetchAll();
39    }
40
41    public function getInstitutionShiftIds($institutionId) {
42        $sql = "SELECT shift_id FROM institution_shifts WHERE institution_id = :institution_id";
43        $stmt = $this->db->prepare($sql);
44        $stmt->execute([':institution_id' => $institutionId]);
45        return $stmt->fetchAll(PDO::FETCH_COLUMN);
46    }
47 }

```

## 5.6 — Justification.php

models/Justification.php

PHP

```

1 <?php
2 class Justification {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function create($data) {
10        $sql = "INSERT INTO justifications
11        (attendance_id, student_id, submitted_by, reason, document_path)
12        VALUES (:attendance_id, :student_id, :submitted_by, :reason, :document_path)";
13
14        $stmt = $this->db->prepare($sql);
15        return $stmt->execute($data);
16    }
17
18    public function getPending() {
19        $sql = "SELECT j.*,
20        CONCAT(u.last_name, ' ', u.first_name) as student_name,
21        CONCAT(s.last_name, ' ', s.first_name) as submitted_by_name,
22        a.date as attendance_date,
23        sub.name as subject_name,
24        c.name as course_name
25        FROM justifications j
26        INNER JOIN users u ON j.student_id = u.id
27        INNER JOIN users s ON j.submitted_by = s.id
28        INNER JOIN attendances a ON j.attendance_id = a.id
29        INNER JOIN subjects sub ON a.subject_id = sub.id
30        LEFT JOIN course_students cs ON u.id = cs.student_id
31        LEFT JOIN courses c ON cs.course_id = c.id
32        WHERE j.status = 'pendiente'
33        ORDER BY j.created_at DESC";
34
35        $stmt = $this->db->query($sql);
36        return $stmt->fetchAll();
37    }
38
39    public function getByStudent($studentId) {
40        $sql = "SELECT j.*,
41        a.date as attendance_date,
42        sub.name as subject_name,
43        j.created_at as submitted_at,
44        j.updated_at as reviewed_at,
45        CONCAT(r.last_name, ' ', r.first_name) as reviewer_name
46        FROM justifications j
47        INNER JOIN attendances a ON j.attendance_id = a.id
48        INNER JOIN subjects sub ON a.subject_id = sub.id
49        LEFT JOIN users r ON j.reviewed_by = r.id
50        WHERE j.student_id = :student_id
51        ORDER BY j.created_at DESC";
52
53        $stmt = $this->db->prepare($sql);
54        $stmt->execute([':student_id' => $studentId]);
55        return $stmt->fetchAll();
56    }
57
58    public function approve($justificationId, $reviewerId, $notes = '') {
59        $sql = "UPDATE justifications
60        SET status = 'aprobado',

```

```

61 reviewed_by = :reviewed_by,
62 review_notes = :notes,
63 updated_at = NOW()
64 WHERE id = :id";
65
66 $stmt = $this->db->prepare($sql);
67 $result = $stmt->execute([
68 ':reviewed_by' => $reviewerId,
69 ':notes' => $notes,
70 ':id' => $justificationId
71 ]);
72
73 if ($result) {
74 // Actualizar asistencia a justificado
75 $sql2 = "UPDATE attendances a
76 INNER JOIN justifications j ON a.id = j.attendance_id
77 SET a.status = 'justificado'
78 WHERE j.id = :id";
79 $stmt2 = $this->db->prepare($sql2);
80 $stmt2->execute([':id' => $justificationId]);
81 }
82
83 return $result;
84 }
85
86 public function reject($justificationId, $reviewerId, $notes) {
87 $sql = "UPDATE justifications
88 SET status = 'rechazado',
89 reviewed_by = :reviewed_by,
90 review_notes = :notes,
91 updated_at = NOW()
92 WHERE id = :id";
93
94 $stmt = $this->db->prepare($sql);
95 return $stmt->execute([
96 ':reviewed_by' => $reviewerId,
97 ':notes' => $notes,
98 ':id' => $justificationId
99 ]);
100 }
101
102 public function existsByAttendance($attendanceId) {
103 $sql = "SELECT COUNT(*) as count FROM justifications WHERE attendance_id = :attendance_id";
104 $stmt = $this->db->prepare($sql);
105 $stmt->execute([':attendance_id' => $attendanceId]);
106 $result = $stmt->fetch();
107 return $result['count'] > 0;
108 }
109
110 public function getById($id) {
111 $sql = "SELECT j.*, j.student_id, j.submitted_by,
112 CONCAT(u.last_name, ' ', u.first_name) as student_name
113 FROM justifications j
114 INNER JOIN users u ON j.student_id = u.id
115 WHERE j.id = :id";
116 $stmt = $this->db->prepare($sql);
117 $stmt->execute([':id' => $id]);
118 return $stmt->fetch();
119 }
120

```

[illegible]

[illegible]

```

241 if (!empty($j['attendance_id'])) {
242     $this->db->prepare(
243         "UPDATE attendances SET status='justificado' WHERE id=:id"
244     )->execute([':id' => $j['attendance_id']]);
245 }
246 // Si tiene rango de fechas, actualizar todas las ausencias del estudiante en ese rango
247 if (!empty($j['date_from']) && !empty($j['date_to'])) {
248     $this->db->prepare(
249         "UPDATE attendances SET status='justificado'
250         WHERE student_id=:sid AND date BETWEEN :df AND :dt AND status='ausente'"
251     )->execute([
252         ':sid' => $j['student_id'],
253         ':df' => $j['date_from'],
254         ':dt' => $j['date_to'],
255     ]);
256 }
257 }
258 return $result;
259 }
260
261 // Crear una justificación por cada asistencia seleccionada
262 public function createForAttendances($attendanceIds, $data) {
263     $sql = "INSERT INTO justifications
264     (attendance_id, student_id, submitted_by,
265     date_from, date_to, working_days,
266     reason_type, reason, document_path, can_approve, status)
267     VALUES
268     (:attendance_id, :student_id, :submitted_by,
269     :date_from, :date_to, :working_days,
270     :reason_type, :reason, :document_path, :can_approve, 'pendiente')";
271     $stmt = $this->db->prepare($sql);
272     $ok = true;
273     foreach ($attendanceIds as $attId) {
274         $row = $data;
275         $row[':attendance_id'] = (int)$attId;
276         if (!$stmt->execute($row)) $ok = false;
277     }
278     return $ok;
279 }
280
281 // Contar ausencias ya justificadas (pendiente o aprobada) para un estudiante en un rango
282 public function countPendingJustifications($studentId) {
283     $sql = "SELECT COUNT(*) FROM justifications
284     WHERE student_id = :sid AND status = 'pendiente'";
285     $stmt = $this->db->prepare($sql);
286     $stmt->execute([':sid' => $studentId]);
287     return (int)$stmt->fetchColumn();
288 }
289 }

```

## 5.7 — Notification.php

models/Notification.php

PHP

```

1 <?php
2 class Notification {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     // Crear notificación
10    // $link: URL relativa destino, ej: '?action=pending_justifications'
11    public function create($userId, $title, $message, $type = 'info', $link = null) {
12        $sql = "INSERT INTO notifications (user_id, title, message, type, link, is_read, created_at)
13        VALUES (:user_id, :title, :message, :type, :link, 0, NOW())";
14        $stmt = $this->db->prepare($sql);
15        return $stmt->execute([
16            ':user_id' => $userId,
17            ':title' => $title,
18            ':message' => $message,
19            ':type' => $type,
20            ':link' => $link
21        ]);
22    }
23
24    // Obtener notificaciones de un usuario (con paginación)
25    public function getByUser($userId, $limit = 20, $offset = 0) {
26        $sql = "SELECT * FROM notifications
27        WHERE user_id = :user_id
28        ORDER BY created_at DESC
29        LIMIT :limit OFFSET :offset";
30        $stmt = $this->db->prepare($sql);
31        $stmt->bindValue(':user_id', $userId, PDO::PARAM_INT);
32        $stmt->bindValue(':limit', $limit, PDO::PARAM_INT);
33        $stmt->bindValue(':offset', $offset, PDO::PARAM_INT);
34        $stmt->execute();
35        return $stmt->fetchAll();
36    }
37
38    // Solo no leídas (para el dropdown del navbar)
39    public function getUnread($userId, $limit = 10) {
40        $sql = "SELECT * FROM notifications
41        WHERE user_id = :user_id AND is_read = 0
42        ORDER BY created_at DESC
43        LIMIT :limit";
44        $stmt = $this->db->prepare($sql);
45        $stmt->bindValue(':user_id', $userId, PDO::PARAM_INT);
46        $stmt->bindValue(':limit', $limit, PDO::PARAM_INT);
47        $stmt->execute();
48        return $stmt->fetchAll();
49    }
50
51    // Contar no leídas
52    public function getUnreadCount($userId) {
53        $sql = "SELECT COUNT(*) as count FROM notifications
54        WHERE user_id = :user_id AND is_read = 0";
55        $stmt = $this->db->prepare($sql);
56        $stmt->execute([':user_id' => $userId]);
57        return (int)$stmt->fetch()['count'];
58    }
59
60    // Marcar una como leída

```



```

61 public function markAsRead($notificationId, $userId) {
62     $sql = "UPDATE notifications SET is_read = 1
63     WHERE id = :id AND user_id = :user_id";
64     $stmt = $this->db->prepare($sql);
65     return $stmt->execute([':id' => $notificationId, ':user_id' => $userId]);
66 }
67
68 // Marcar TODAS como leídas
69 public function markAllAsRead($userId) {
70     $sql = "UPDATE notifications SET is_read = 1
71     WHERE user_id = :user_id AND is_read = 0";
72     $stmt = $this->db->prepare($sql);
73     return $stmt->execute([':user_id' => $userId]);
74 }
75
76 // Eliminar una notificación
77 public function delete($notificationId, $userId) {
78     $sql = "DELETE FROM notifications WHERE id = :id AND user_id = :user_id";
79     $stmt = $this->db->prepare($sql);
80     return $stmt->execute([':id' => $notificationId, ':user_id' => $userId]);
81 }
82
83 // Eliminar todas las leídas
84 public function deleteRead($userId) {
85     $sql = "DELETE FROM notifications WHERE user_id = :user_id AND is_read = 1";
86     $stmt = $this->db->prepare($sql);
87     return $stmt->execute([':user_id' => $userId]);
88 }
89
90 // Total de notificaciones del usuario (para paginación)
91 public function countByUser($userId) {
92     $sql = "SELECT COUNT(*) as count FROM notifications WHERE user_id = :user_id";
93     $stmt = $this->db->prepare($sql);
94     $stmt->execute([':user_id' => $userId]);
95     return (int)$stmt->fetch()['count'];
96 }
97 }

```

## 5.8 — Representative.php

models/Representative.php

PHP

```

1 <?php
2 class Representative {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function assignStudent($representativeId, $studentId, $relationship, $isPrimary = 0) {
10        $sql = "INSERT INTO representatives
11        (representative_id, student_id, relationship, is_primary)
12        VALUES
13        (:rep_id, :stu_id, :rel_ins, :prim_ins)
14        ON DUPLICATE KEY UPDATE
15        relationship = :rel_upd,
16        is_primary = :prim_upd";
17
18        $stmt = $this->db->prepare($sql);
19
20        return $stmt->execute([
21            ':rep_id' => $representativeId,
22            ':stu_id' => $studentId,
23            ':rel_ins' => $relationship,
24            ':prim_ins' => $isPrimary,
25            ':rel_upd' => $relationship,
26            ':prim_upd' => $isPrimary
27        ]);
28    }
29
30    public function getStudentsByRepresentative($representativeId) {
31        $sql = "SELECT u.*, r.relationship, r.is_primary, c.name as course_name, sh.name as shift_name
32        FROM users u
33        INNER JOIN representatives r ON u.id = r.student_id
34        LEFT JOIN course_students cs ON u.id = cs.student_id
35        LEFT JOIN courses c ON cs.course_id = c.id
36        LEFT JOIN shifts sh ON c.shift_id = sh.id
37        WHERE r.representative_id = :representative_id
38        ORDER BY r.is_primary DESC, u.last_name, u.first_name";
39
40        $stmt = $this->db->prepare($sql);
41        $stmt->execute([':representative_id' => $representativeId]);
42        return $stmt->fetchAll();
43    }
44
45    public function getRepresentativesByStudent($studentId) {
46        $sql = "SELECT u.*, r.relationship, r.is_primary
47        FROM users u
48        INNER JOIN representatives r ON u.id = r.representative_id
49        WHERE r.student_id = :student_id
50        ORDER BY r.is_primary DESC";
51
52        $stmt = $this->db->prepare($sql);
53        $stmt->execute([':student_id' => $studentId]);
54        return $stmt->fetchAll();
55    }
56
57    public function removeStudent($representativeId, $studentId) {
58        $sql = "DELETE FROM representatives
59        WHERE representative_id = :representative_id AND student_id = :student_id";
60
61        $stmt = $this->db->prepare($sql);
62        return $stmt->execute([
63            ':representative_id' => $representativeId,
64            ':student_id' => $studentId
65        ]);
66    }
67 }

```

## 5.9 — Role.php

models/Role.php

PHP

```
1 <?php
2 class Role {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function getAll() {
10        $sql = "SELECT * FROM roles ORDER BY name";
11        $stmt = $this->db->query($sql);
12        return $stmt->fetchAll();
13    }
14
15    public function findById($id) {
16        $sql = "SELECT * FROM roles WHERE id = :id";
17        $stmt = $this->db->prepare($sql);
18        $stmt->execute([':id' => $id]);
19        return $stmt->fetch();
20    }
21 }
```

## 5.10 — SchoolYear.php

models/SchoolYear.php

PHP

```

1 <?php
2 class SchoolYear {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function getActive() {
10        $sql = "SELECT * FROM school_years
11        WHERE institution_id = :institution_id AND is_active = 1
12        LIMIT 1";
13
14        $stmt = $this->db->prepare($sql);
15        $stmt->execute([':institution_id' => $_SESSION['institution_id']]);
16        return $stmt->fetch();
17    }
18
19    public function getAll() {
20        $sql = "SELECT * FROM school_years
21        WHERE institution_id = :institution_id
22        ORDER BY start_date DESC";
23
24        $stmt = $this->db->prepare($sql);
25        $stmt->execute([':institution_id' => $_SESSION['institution_id']]);
26        return $stmt->fetchAll();
27    }
28
29
30    public function findById($id) {
31        $sql = "SELECT * FROM school_years
32        WHERE id = :id AND institution_id = :institution_id";
33
34        $stmt = $this->db->prepare($sql);
35        $stmt->execute([
36            ':id' => $id,
37            ':institution_id' => $_SESSION['institution_id']
38        ]);
39
40        return $stmt->fetch();
41    }
42
43    public function create($data) {
44        $sql = "INSERT INTO school_years
45        (institution_id, name, start_date, end_date, is_active)
46        VALUES
47        (:institution_id, :name, :start_date, :end_date, :is_active)";
48
49        $stmt = $this->db->prepare($sql);
50        $ok = $stmt->execute([
51            ':institution_id' => $data['institution_id'],
52            ':name' => $data['name'],
53            ':start_date' => $data['start_date'],
54            ':end_date' => $data['end_date'],
55            ':is_active' => $data['is_active']
56        ]);
57        // Retorna el ID insertado para poder activarlo en la misma conexión
58        return $ok ? (int)$this->db->lastInsertId() : 0;
59    }
60

```

```

61 public function update($data) {
62     $sql = "UPDATE school_years
63     SET name=:name,
64     start_date=:start_date,
65     end_date=:end_date
66     WHERE id=:id
67     AND institution_id=:institution_id";
68
69     $stmt = $this->db->prepare($sql);
70
71     return $stmt->execute([
72         ':id'=>$data['id'],
73         ':name'=>$data['name'],
74         ':start_date'=>$data['start_date'],
75         ':end_date'=>$data['end_date'],
76         ':institution_id'=>$_SESSION['institution_id']
77     ]);
78 }
79
80 public function delete($id) {
81     $stmt = $this->db->prepare(
82     "DELETE FROM school_years
83     WHERE id=:id AND institution_id=:institution_id"
84     );
85
86     return $stmt->execute([
87         ':id'=>$id,
88         ':institution_id'=>$_SESSION['institution_id']
89     ]);
90 }
91
92 public function activate($id) {
93
94     // desactiva todos
95     $this->db->prepare(
96     "UPDATE school_years
97     SET is_active=0
98     WHERE institution_id=:institution_id"
99     )->execute([':institution_id'=>$_SESSION['institution_id']]);
100
101     if ($id == 0) return true;
102
103     // activa uno
104     $stmt = $this->db->prepare(
105     "UPDATE school_years
106     SET is_active=1
107     WHERE id=:id AND institution_id=:institution_id"
108     );
109
110     return $stmt->execute([
111         ':id'=>$id,
112         ':institution_id'=>$_SESSION['institution_id']
113     ]);
114 }
115
116 public function deactivate($id) {
117     $stmt = $this->db->prepare(
118     "UPDATE school_years
119     SET is_active=0
120     WHERE id=:id AND institution_id=:institution_id"

```

```

121 );
122
123 return $stmt->execute([
124 ':id'=>$id,
125 ':institution_id'=>$_SESSION['institution_id']
126 ]);
127 }
128
129 public function checkOverlap($start, $end, $excludeId = null) {
130
131 $sql = "SELECT COUNT(*)
132 FROM school_years
133 WHERE institution_id = :institution_id
134 AND (start_date <= :end AND end_date >= :start)";
135
136 if ($excludeId) {
137 $sql .= " AND id != :excludeId";
138 }
139
140 $stmt = $this->db->prepare($sql);
141
142 $params = [
143 ':institution_id' => $_SESSION['institution_id'],
144 ':start' => $start,
145 ':end' => $end
146 ];
147
148 if ($excludeId) {
149 $params[':excludeId'] = $excludeId;
150 }
151
152 $stmt->execute($params);
153 return $stmt->fetchColumn() > 0;
154 }
155
156 }

```

## 5.11 — Shift.php

models/Shift.php

PHP

```

1 <?php
2 class Shift {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function getAll() {
10        $sql = "SELECT * FROM shifts ORDER BY id";
11        $stmt = $this->db->query($sql);
12        return $stmt->fetchAll();
13    }
14 }

```

## 5.12 — Subject.php

models/Subject.php

PHP

```

1 <?php
2 class Subject {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function getAll() {
10        $sql = "SELECT * FROM subjects
11        WHERE institution_id = :institution_id
12        ORDER BY name";
13
14        $stmt = $this->db->prepare($sql);
15        $stmt->execute([':institution_id' => $_SESSION['institution_id']]);
16        return $stmt->fetchAll();
17    }
18
19    public function create($data) {
20        $sql = "INSERT INTO subjects (institution_id, name, code)
21        VALUES (:institution_id, :name, :code)";
22
23        $stmt = $this->db->prepare($sql);
24        return $stmt->execute($data);
25    }
26
27    public function findById($id) {
28        $sql = "SELECT * FROM subjects WHERE id = :id";
29        $stmt = $this->db->prepare($sql);
30        $stmt->execute([':id' => $id]);
31        return $stmt->fetch();
32    }
33
34    public function update($data) {
35        $sql = "UPDATE subjects SET
36        name = :name,
37        code = :code,
38        updated_at = NOW()
39        WHERE id = :id";
40
41        $stmt = $this->db->prepare($sql);
42        return $stmt->execute($data);
43    }
44
45    public function delete($id) {
46        $sql = "DELETE FROM subjects WHERE id = :id";
47        $stmt = $this->db->prepare($sql);
48        return $stmt->execute([':id' => $id]);
49    }
50 }

```

## 5.13 — TeacherAssignment.php

models/TeacherAssignment.php

PHP

```

1 <?php
2 class TeacherAssignment {
3     private $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function assign($data) {
10         // Verificar si ya existe asignación curso-materia
11         $checkSql = "SELECT ta.id, CONCAT(u.last_name, ' ', u.first_name) as teacher_name
12         FROM teacher_assignments ta
13         INNER JOIN users u ON ta.teacher_id = u.id
14         WHERE ta.course_id = :course_id
15         AND ta.subject_id = :subject_id
16         AND ta.school_year_id = :school_year_id";
17
18         $checkStmt = $this->db->prepare($checkSql);
19         $checkStmt->execute([
20             ':course_id' => $data[':course_id'],
21             ':subject_id' => $data[':subject_id'],
22             ':school_year_id' => $data[':school_year_id']
23         ]);
24
25         $existing = $checkStmt->fetch();
26
27         if ($existing) {
28             return [
29                 'success' => false,
30                 'message' => 'La materia ya está asignada a: ' . $existing['teacher_name']
31             ];
32         }
33
34         $sql = "INSERT INTO teacher_assignments
35         (teacher_id, course_id, subject_id, school_year_id, is_tutor)
36         VALUES (:teacher_id, :course_id, :subject_id, :school_year_id, :is_tutor)";
37
38         $stmt = $this->db->prepare($sql);
39         $result = $stmt->execute($data);
40
41         return [
42             'success' => $result,
43             'message' => 'Asignación creada correctamente'
44         ];
45     }
46
47     public function getByTeacher($teacherId) {
48         $sql = "SELECT ta.*, c.name as course_name, s.name as subject_name,
49         sh.name as shift_name, sy.name as year_name
50         FROM teacher_assignments ta
51         INNER JOIN courses c ON ta.course_id = c.id
52         INNER JOIN subjects s ON ta.subject_id = s.id
53         INNER JOIN shifts sh ON ta.shift_id = sh.id
54         INNER JOIN school_years sy ON ta.school_year_id = sy.id
55         WHERE ta.teacher_id = :teacher_id
56         ORDER BY c.name, s.name";
57
58         $stmt = $this->db->prepare($sql);
59         $stmt->execute([':teacher_id' => $teacherId]);
60         return $stmt->fetchAll();

```



```

61 }
62
63 public function getByCourse($courseId) {
64 $sql = "SELECT ta.*,
65 CONCAT(u.last_name, ' ', u.first_name) as teacher_name,
66 s.name as subject_name
67 FROM teacher_assignments ta
68 INNER JOIN users u ON ta.teacher_id = u.id
69 INNER JOIN subjects s ON ta.subject_id = s.id
70 WHERE ta.course_id = :course_id
71 ORDER BY s.name";
72
73 $stmt = $this->db->prepare($sql);
74 $stmt->execute([':course_id' => $courseId]);
75 return $stmt->fetchAll();
76 }
77
78 public function getTutorByCourse($courseId, $schoolYearId) {
79 $sql = "SELECT u.*, ta.id as assignment_id
80 FROM teacher_assignments ta
81 INNER JOIN users u ON ta.teacher_id = u.id
82 WHERE ta.course_id = :course_id
83 AND ta.school_year_id = :school_year_id
84 AND ta.is_tutor = 1
85 LIMIT 1";
86
87 $stmt = $this->db->prepare($sql);
88 $stmt->execute([
89 ':course_id' => $courseId,
90 ':school_year_id' => $schoolYearId
91 ]);
92 return $stmt->fetch();
93 }
94
95 public function setTutor($courseId, $teacherId, $schoolYearId) {
96 // Verificar si el docente ya es tutor de otro curso en el mismo año lectivo
97 $checkSql = "SELECT c.name
98 FROM teacher_assignments ta
99 INNER JOIN courses c ON ta.course_id = c.id
100 WHERE ta.teacher_id = :teacher_id
101 AND ta.school_year_id = :school_year_id
102 AND ta.is_tutor = 1
103 AND ta.course_id != :course_id";
104
105 $checkStmt = $this->db->prepare($checkSql);
106 $checkStmt->execute([
107 ':teacher_id' => $teacherId,
108 ':school_year_id' => $schoolYearId,
109 ':course_id' => $courseId
110 ]);
111
112 $existingTutor = $checkStmt->fetch();
113
114 if ($existingTutor) {
115 return [
116 'success' => false,
117 'message' => 'Este docente ya es tutor del curso: ' . $existingTutor['name']
118 ];
119 }
120

```

```

121 // Quitar tutor actual del curso
122 $sql1 = "UPDATE teacher_assignments
123 SET is_tutor = 0
124 WHERE course_id = :course_id AND school_year_id = :school_year_id";
125 $stmt1 = $this->db->prepare($sql1);
126 $stmt1->execute([
127 ':course_id' => $courseId,
128 ':school_year_id' => $schoolYearId
129 ]);
130
131 // Verificar si docente tiene asignación en el curso
132 $checkAssignment = "SELECT id FROM teacher_assignments
133 WHERE course_id = :course_id
134 AND teacher_id = :teacher_id
135 AND school_year_id = :school_year_id
136 LIMIT 1";
137 $stmt = $this->db->prepare($checkAssignment);
138 $stmt->execute([
139 ':course_id' => $courseId,
140 ':teacher_id' => $teacherId,
141 ':school_year_id' => $schoolYearId
142 ]);
143 $existing = $stmt->fetch();
144
145 if ($existing) {
146 // Actualizar asignación existente
147 $sql2 = "UPDATE teacher_assignments
148 SET is_tutor = 1
149 WHERE id = :id";
150 $stmt2 = $this->db->prepare($sql2);
151 $result = $stmt2->execute([':id' => $existing['id']]);
152 } else {
153 return [
154 'success' => false,
155 'message' => 'El docente debe tener al menos una asignatura en el curso antes de ser tutor'
156 ];
157 }
158
159 return [
160 'success' => $result,
161 'message' => 'Tutor asignado correctamente'
162 ];
163 }
164
165 public function remove($assignmentId) {
166 // Verificar si es tutor antes de eliminar
167 $checkSql = "SELECT is_tutor FROM teacher_assignments WHERE id = :id";
168 $checkStmt = $this->db->prepare($checkSql);
169 $checkStmt->execute([':id' => $assignmentId]);
170 $assignment = $checkStmt->fetch();
171
172 if ($assignment && $assignment['is_tutor'] == 1) {
173 return [
174 'success' => false,
175 'message' => 'No se puede eliminar: este docente es tutor del curso. Quite primero la tutoría.'
176 ];
177 }
178
179 $sql = "DELETE FROM teacher_assignments WHERE id = :id";
180 $stmt = $this->db->prepare($sql);

```

```

181 $result = $stmt->execute([':id' => $assignmentId]);
182
183 return [
184 'success' => $result,
185 'message' => 'Asignación eliminada correctamente'
186 ];
187 }
188
189 public function getAll() {
190 $sql = "SELECT ta.*,
191 CONCAT(u.last_name, ' ', u.first_name) as teacher_name,
192 c.name as course_name,
193 s.name as subject_name,
194 sy.name as year_name
195 FROM teacher_assignments ta
196 INNER JOIN users u ON ta.teacher_id = u.id
197 INNER JOIN courses c ON ta.course_id = c.id
198 INNER JOIN subjects s ON ta.subject_id = s.id
199 INNER JOIN school_years sy ON ta.school_year_id = sy.id
200 ORDER BY c.name, s.name";
201
202 $stmt = $this->db->query($sql);
203 return $stmt->fetchAll();
204 }
205 }

```

## 5.14 — User.php

models/User.php

PHP

```

1 <?php
2 class User {
3     public $db;
4
5     public function __construct($database) {
6         $this->db = $database->connect();
7     }
8
9     public function create($data) {
10        $sql = "INSERT INTO users (institution_id, username, email, password, first_name, last_name, dni, phone)
11        VALUES (:institution_id, :username, :email, :password, :first_name, :last_name, :dni, :phone)";
12
13        $stmt = $this->db->prepare($sql);
14        return $stmt->execute([
15            ':institution_id' => $data['institution_id'],
16            ':username' => $data['username'],
17            ':email' => $data['email'],
18            ':password' => Security::hashPassword($data['password']),
19            ':first_name' => $data['first_name'],
20            ':last_name' => $data['last_name'],
21            ':dni' => $data['dni'] ?? null,
22            ':phone' => $data['phone'] ?? null
23        ]);
24    }
25
26    public function findByEmail($email) {
27        $sql = "SELECT * FROM users WHERE email = :email AND is_active = 1";
28        $stmt = $this->db->prepare($sql);
29        $stmt->execute([':email' => $email]);
30        return $stmt->fetch();
31    }
32
33    public function findById($id) {
34        $sql = "SELECT * FROM users WHERE id = :id";
35        $stmt = $this->db->prepare($sql);
36        $stmt->execute([':id' => $id]);
37        return $stmt->fetch();
38    }
39
40    public function getUserRoles($userId) {
41        $sql = "SELECT r.name FROM roles r
42        INNER JOIN user_roles ur ON r.id = ur.role_id
43        WHERE ur.user_id = :user_id";
44        $stmt = $this->db->prepare($sql);
45        $stmt->execute([':user_id' => $userId]);
46        return $stmt->fetchAll(PDO::FETCH_COLUMN);
47    }
48
49    public function assignRole($userId, $roleId) {
50        $sql = "INSERT IGNORE INTO user_roles (user_id, role_id) VALUES (:user_id, :role_id)";
51        $stmt = $this->db->prepare($sql);
52        return $stmt->execute([':user_id' => $userId, ':role_id' => $roleId]);
53    }
54
55    public function setResetToken($email, $token) {
56        $sql = "UPDATE users SET reset_token = :token, reset_expires = DATE_ADD(NOW(), INTERVAL 1 HOUR)
57        WHERE email = :email";
58        $stmt = $this->db->prepare($sql);
59        return $stmt->execute([':token' => $token, ':email' => $email]);
60    }

```

```

61
62 public function validateResetToken($token) {
63 $sql = "SELECT * FROM users WHERE reset_token = :token AND reset_expires > NOW()";
64 $stmt = $this->db->prepare($sql);
65 $stmt->execute([':token' => $token]);
66 return $stmt->fetch();
67 }
68
69 public function resetPassword($userId, $newPassword) {
70 $sql = "UPDATE users SET password = :password, reset_token = NULL, reset_expires = NULL
71 WHERE id = :id";
72 $stmt = $this->db->prepare($sql);
73 return $stmt->execute([
74 ':password' => Security::hashPassword($newPassword),
75 ':id' => $userId
76 ]);
77 }
78
79 public function getAll() {
80 $sql = "SELECT u.*, GROUP_CONCAT(r.name) as roles
81 FROM users u
82 LEFT JOIN user_roles ur ON u.id = ur.user_id
83 LEFT JOIN roles r ON ur.role_id = r.id
84 WHERE u.institution_id = :institution_id
85 GROUP BY u.id
86 ORDER BY u.last_name, u.first_name";
87
88 $stmt = $this->db->prepare($sql);
89 $stmt->execute([':institution_id' => $_SESSION['institution_id']]);
90 return $stmt->fetchAll();
91 }
92
93 public function removeRole($userId, $roleId) {
94 $sql = "DELETE FROM user_roles WHERE user_id = :user_id AND role_id = :role_id";
95 $stmt = $this->db->prepare($sql);
96 return $stmt->execute([':user_id' => $userId, ':role_id' => $roleId]);
97 }
98
99 public function getUserWithRoles($userId) {
100 $user = $this->findById($userId);
101 if ($user) {
102 $user['roles'] = $this->getUserRoles($userId);
103 }
104 return $user;
105 }
106
107 /* public function getByRole($roleName) {
108 $sql = "SELECT u.* FROM users u
109 INNER JOIN user_roles ur ON u.id = ur.user_id
110 INNER JOIN roles r ON ur.role_id = r.id
111 WHERE r.name = :role_name AND u.institution_id = :institution_id
112 ORDER BY u.last_name, u.first_name";
113
114 $stmt = $this->db->prepare($sql);
115 $stmt->execute([
116 ':role_name' => $roleName,
117 ':institution_id' => $_SESSION['institution_id']
118 ]);
119 return $stmt->fetchAll();
120 } */

```

```

121
122 // Agregar al final de la clase User
123
124 public function getByRole($roleName) {
125     $sql = "SELECT u.* FROM users u
126     INNER JOIN user_roles ur ON u.id = ur.user_id
127     INNER JOIN roles r ON ur.role_id = r.id
128     WHERE r.name = :role_name AND u.institution_id = :institution_id
129     ORDER BY u.last_name, u.first_name";
130
131     $stmt = $this->db->prepare($sql);
132     $stmt->execute([
133         ':role_name' => $roleName,
134         ':institution_id' => $_SESSION['institution_id']
135     ]);
136     return $stmt->fetchAll();
137 }
138
139 public function getStudentsNotEnrolled($schoolYearId) {
140     $sql = "SELECT u.* FROM users u
141     INNER JOIN user_roles ur ON u.id = ur.user_id
142     INNER JOIN roles r ON ur.role_id = r.id
143     WHERE r.name = 'estudiante'
144     AND u.institution_id = :institution_id
145     AND u.id NOT IN (
146     SELECT student_id FROM course_students
147     WHERE school_year_id = :school_year_id
148     )
149     ORDER BY u.last_name, u.first_name";
150
151     $stmt = $this->db->prepare($sql);
152     $stmt->execute([
153         ':institution_id' => $_SESSION['institution_id'],
154         ':school_year_id' => $schoolYearId
155     ]);
156     return $stmt->fetchAll();
157 }
158
159 public function getStudentCourse($studentId, $schoolYearId) {
160     $sql = "SELECT c.*, s.name as shift_name
161     FROM courses c
162     INNER JOIN course_students cs ON c.id = cs.course_id
163     INNER JOIN shifts s ON c.shift_id = s.id
164     WHERE cs.student_id = :student_id
165     AND cs.school_year_id = :school_year_id";
166
167     $stmt = $this->db->prepare($sql);
168     $stmt->execute([
169         ':student_id' => $studentId,
170         ':school_year_id' => $schoolYearId
171     ]);
172     return $stmt->fetch();
173 }
174
175 public function updateProfile($userId, $data) {
176     $sql = "UPDATE users
177     SET first_name = :first_name,
178     last_name = :last_name,
179     phone = :phone,
180     email = :email,

```

```

181 updated_at = NOW()
182 WHERE id = :id";
183
184 $stmt = $this->db->prepare($sql);
185 return $stmt->execute([
186 ':first_name' => $data['first_name'],
187 ':last_name' => $data['last_name'],
188 ':phone' => $data['phone'],
189 ':email' => $data['email'],
190 ':id' => $userId
191 ]);
192 }
193
194 public function updatePassword($userId, $newPassword) {
195 $sql = "UPDATE users SET password = :password WHERE id = :id";
196 $stmt = $this->db->prepare($sql);
197 return $stmt->execute([
198 ':password' => Security::hashPassword($newPassword),
199 ':id' => $userId
200 ]);
201 }
202
203 public function findByEmailOrUsername($emailOrUsername) {
204 $sql = "SELECT * FROM users
205 WHERE (email = :email OR username = :username)
206 AND is_active = 1";
207
208 $stmt = $this->db->prepare($sql);
209 $stmt->execute([
210 ':email' => $emailOrUsername,
211 ':username' => $emailOrUsername
212 ]);
213 return $stmt->fetch();
214 }
215
216 public function getUserRoleIds($userId) {
217 $sql = "SELECT r.id FROM roles r
218 INNER JOIN user_roles ur ON r.id = ur.role_id
219 WHERE ur.user_id = :user_id
220 ORDER BY r.name";
221 $stmt = $this->db->prepare($sql);
222 $stmt->execute([':user_id' => $userId]);
223 return $stmt->fetchAll(PDO::FETCH_COLUMN);
224 }
225
226 public function findByUsername($username) {
227 $sql = "SELECT * FROM users WHERE username = :username";
228 $stmt = $this->db->prepare($sql);
229 $stmt->execute([':username' => $username]);
230 return $stmt->fetch();
231 }
232
233 public function update($data) {
234 // Si se incluye password, actualizarla
235 if (isset($data['password'])) {
236 $sql = "UPDATE users SET
237 username = :username,
238 email = :email,
239 password = :password,
240 first_name = :first_name,

```

```

241 last_name = :last_name,
242 dni = :dni,
243 phone = :phone,
244 updated_at = NOW()
245 WHERE id = :id";
246
247 $params = [
248   ':id' => $data['id'],
249   ':username' => $data['username'],
250   ':email' => $data['email'],
251   ':password' => Security::hashPassword($data['password']),
252   ':first_name' => $data['first_name'],
253   ':last_name' => $data['last_name'],
254   ':dni' => $data['dni'],
255   ':phone' => $data['phone']
256 ];
257 } else {
258   // No actualizar password
259   $sql = "UPDATE users SET
260     username = :username,
261     email = :email,
262     first_name = :first_name,
263     last_name = :last_name,
264     dni = :dni,
265     phone = :phone,
266     updated_at = NOW()
267     WHERE id = :id";
268
269   $params = [
270     ':id' => $data['id'],
271     ':username' => $data['username'],
272     ':email' => $data['email'],
273     ':first_name' => $data['first_name'],
274     ':last_name' => $data['last_name'],
275     ':dni' => $data['dni'],
276     ':phone' => $data['phone']
277   ];
278 }
279
280 $stmt = $this->db->prepare($sql);
281 return $stmt->execute($params);
282 }
283
284 public function delete($userId) {
285   // Primero eliminar roles
286   $sql = "DELETE FROM user_roles WHERE user_id = :user_id";
287   $stmt = $this->db->prepare($sql);
288   $stmt->execute([':user_id' => $userId]);
289
290   // Eliminar representaciones (tanto como representante como estudiante)
291   $sql = "DELETE FROM representatives
292     WHERE representative_id = :rep_id OR student_id = :stu_id";
293   $stmt = $this->db->prepare($sql);
294   $stmt->execute([
295     ':rep_id' => $userId,
296     ':stu_id' => $userId
297   ]);
298
299   // Eliminar de course_students
300   $sql = "DELETE FROM course_students WHERE student_id = :user_id";

```



```

301 $stmt = $this->db->prepare($sql);
302 $stmt->execute([':user_id' => $userId]);
303
304 // Eliminar asignaciones docentes
305 $sql = "DELETE FROM teacher_assignments WHERE teacher_id = :user_id";
306 $stmt = $this->db->prepare($sql);
307 $stmt->execute([':user_id' => $userId]);
308
309 // Finalmente eliminar usuario
310 $sql = "DELETE FROM users WHERE id = :id";
311 $stmt = $this->db->prepare($sql);
312 return $stmt->execute([':id' => $userId]);
313 }
314
315 public function deactivate($userId) {
316 $sql = "UPDATE users SET is_active = 0, updated_at = NOW() WHERE id = :id";
317 $stmt = $this->db->prepare($sql);
318 return $stmt->execute([':id' => $userId]);
319 }
320 }

```

## 6. MODULO HELPERS (4 archivos)

### 6.1 — Backup.php

helpers/Backup.php

PHP

```
1 <?php
2 class Backup {
3     private $dbHost = 'localhost';
4     private $dbUser = 'root';
5     private $dbPass = '';
6     private $dbName = 'ecuasistencia2026_db';
7     private $backupDir;
8
9     public function __construct() {
10         $this->backupDir = BASE_PATH . '/backups/';
11         if (!file_exists($this->backupDir)) {
12             mkdir($this->backupDir, 0755, true);
13         }
14     }
15
16     public function createBackup() {
17         $filename = 'backup_' . date('Y-m-d_H-i-s') . '.sql';
18         $filepath = $this->backupDir . $filename;
19
20         // Verificar que mysqldump esté disponible
21         $mysqldumpPath = 'C:\\xampp\\mysql\\bin\\mysqldump.exe'; // XAMPP Windows
22         if (!file_exists($mysqldumpPath)) {
23             $mysqldumpPath = 'mysqldump'; // Linux/Mac o PATH configurado
24         }
25
26         // Comando mejorado con opciones
27         if (empty($this->dbPass)) {
28             $command = sprintf(
29                 "%s --host=%s --user=%s --skip-password %s > \"%s\" 2>&1",
30                 $mysqldumpPath,
31                 $this->dbHost,
32                 $this->dbUser,
33                 $this->dbName,
34                 $filepath
35             );
36         } else {
37             $command = sprintf(
38                 "%s --host=%s --user=%s --password=%s %s > \"%s\" 2>&1",
39                 $mysqldumpPath,
40                 $this->dbHost,
41                 $this->dbUser,
42                 $this->dbPass,
43                 $this->dbName,
44                 $filepath
45             );
46         }
47
48         exec($command, $output, $result);
49
50         // Verificar que el archivo existe y tiene contenido
51         if (file_exists($filepath) && filesize($filepath) > 0) {
52             return $filename;
53         }
54
55         // Si falló, intentar eliminar archivo vacío
56         if (file_exists($filepath)) {
57             unlink($filepath);
58         }
59
60         return false;
61     }
62 }
```

```

61 }
62
63 public function getBackups() {
64 $files = glob($this->backupDir . 'backup_*.sql');
65 $backups = [];
66
67 foreach ($files as $file) {
68 $size = filesize($file);
69 $backups[] = [
70 'filename' => basename($file),
71 'path' => $file,
72 'size' => $this->formatBytes($size),
73 'date' => date('d/m/Y H:i:s', filemtime($file)),
74 'timestamp' => filemtime($file)
75 ];
76 }
77
78 usort($backups, function($a, $b) {
79 return $b['timestamp'] - $a['timestamp'];
80 });
81
82 return $backups;
83 }
84
85 private function formatBytes($bytes, $precision = 2) {
86 $units = ['B', 'KB', 'MB', 'GB'];
87 $bytes = max($bytes, 0);
88 $pow = floor(($bytes ? log($bytes) : 0) / log(1024));
89 $pow = min($pow, count($units) - 1);
90 $bytes /= (1 << (10 * $pow));
91 return round($bytes, $precision) . ' ' . $units[$pow];
92 }
93
94 public function deleteOldBackups($daysToKeep = 30) {
95 $files = glob($this->backupDir . 'backup_*.sql');
96 $deleted = 0;
97
98 foreach ($files as $file) {
99 if (time() - filemtime($file) > ($daysToKeep * 24 * 60 * 60)) {
100 unlink($file);
101 $deleted++;
102 }
103 }
104
105 return $deleted;
106 }
107
108 public function downloadBackup($filename) {
109 $filepath = $this->backupDir . $filename;
110
111 if (!file_exists($filepath)) {
112 return false;
113 }
114
115 header('Content-Type: application/octet-stream');
116 header('Content-Disposition: attachment; filename="' . $filename . '"');
117 header('Content-Length: ' . filesize($filepath));
118 readfile($filepath);
119 exit;
120 }

```

```
121
122 public function deleteBackup($filename) {
123 // Validar que sea un archivo de backup válido
124 if (!preg_match('/^backup_[\d\-_]+\\.sql$/', $filename)) {
125 return false;
126 }
127
128 $filepath = $this->backupDir . $filename;
129
130 if (file_exists($filepath)) {
131 return unlink($filepath);
132 }
133
134 return false;
135 }
136 }
```

## 6.2 — Logger.php

helpers/Logger.php

PHP

```

1 <?php
2 class Logger {
3     private $db;
4
5     public function __construct() {
6         $database = new Database();
7         $this->db = $database->connect();
8     }
9
10    public function log($action, $entityType = null, $entityId = null, $description = null) {
11        if (!isset($_SESSION['user_id'])) {
12            return false;
13        }
14
15        $sql = "INSERT INTO activity_logs
16        (user_id, action, entity_type, entity_id, description, ip_address, user_agent)
17        VALUES (:user_id, :action, :entity_type, :entity_id, :description, :ip_address, :user_agent)";
18
19        $stmt = $this->db->prepare($sql);
20        return $stmt->execute([
21            ':user_id' => $_SESSION['user_id'],
22            ':action' => $action,
23            ':entity_type' => $entityType,
24            ':entity_id' => $entityId,
25            ':description' => $description,
26            ':ip_address' => $_SERVER['REMOTE_ADDR'] ?? null,
27            ':user_agent' => $_SERVER['HTTP_USER_AGENT'] ?? null
28        ]);
29    }
30
31    public function getLogs($limit = 100, $filters = []) {
32        $sql = "SELECT l.*, CONCAT(u.last_name, ' ', u.first_name) as user_name
33        FROM activity_logs l
34        INNER JOIN users u ON l.user_id = u.id
35        WHERE l=1";
36
37        $params = [];
38
39        if (!empty($filters['user_id'])) {
40            $sql .= " AND l.user_id = :user_id";
41            $params[':user_id'] = $filters['user_id'];
42        }
43
44        if (!empty($filters['action'])) {
45            $sql .= " AND l.action = :action";
46            $params[':action'] = $filters['action'];
47        }
48
49        if (!empty($filters['start_date'])) {
50            $sql .= " AND l.created_at >= :start_date";
51            $params[':start_date'] = $filters['start_date'];
52        }
53
54        if (!empty($filters['end_date'])) {
55            $sql .= " AND l.created_at <= :end_date";
56            $params[':end_date'] = $filters['end_date'] . ' 23:59:59';
57        }
58
59        $sql .= " ORDER BY l.created_at DESC LIMIT :limit";
60

```

```

61 $stmt = $this->db->prepare($sql);
62 foreach ($params as $key => $value) {
63 $stmt->bindValue($key, $value);
64 }
65 $stmt->bindValue(':limit', $limit, PDO::PARAM_INT);
66 $stmt->execute();
67
68 return $stmt->fetchAll();
69 }
70 }

```

## 6.3 — Mailer.php

helpers/Mailer.php

PHP

```

1 <?php
2 use PHPMailer\PHPMailer\PHPMailer;
3 use PHPMailer\PHPMailer\Exception;
4
5 require BASE_PATH . '/vendor/autoload.php';
6
7 class Mailer {
8     public static function send($to, $subject, $body) {
9         $mail = new PHPMailer(true);
10
11         try {
12             $mail->isSMTP();
13             $mail->Host = SMTP_HOST;
14             $mail->SMTPAuth = true;
15             $mail->Username = SMTP_USER;
16             $mail->Password = SMTP_PASS;
17             $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
18             $mail->Port = SMTP_PORT;
19             $mail->CharSet = 'UTF-8';
20
21             $mail->setFrom(SMTP_FROM, SMTP_NAME);
22             $mail->addAddress($to);
23
24             $mail->isHTML(true);
25             $mail->Subject = $subject;
26             $mail->Body = $body;
27
28             $mail->send();
29             return true;
30         } catch (Exception $e) {
31             error_log("Error de correo: {$mail->ErrorInfo}");
32             return false;
33         }
34     }
35 }

```

## 6.4 — Security.php

helpers/Security.php

PHP

```

1 <?php
2 class Security {
3     public static function generateToken() {
4         if (!isset($_SESSION['csrf_token'])) {
5             $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
6         }
7         return $_SESSION['csrf_token'];
8     }
9
10    public static function validateToken($token) {
11        return isset($_SESSION['csrf_token']) && hash_equals($_SESSION['csrf_token'], $token);
12    }
13
14    public static function sanitize($data) {
15        return htmlspecialchars(strip_tags(trim($data)), ENT_QUOTES, 'UTF-8');
16    }
17
18    public static function validateEmail($email) {
19        return filter_var($email, FILTER_VALIDATE_EMAIL);
20    }
21
22    public static function hashPassword($password) {
23        return password_hash($password, PASSWORD_BCRYPT);
24    }
25
26    public static function verifyPassword($password, $hash) {
27        return password_verify($password, $hash);
28    }
29
30    public static function requireLogin() {
31        if (!isset($_SESSION['user_id'])) {
32            header('Location: ' . BASE_URL . '/public/index.php?action=login');
33            exit;
34        }
35    }
36
37    public static function hasRole($roleNames) {
38        // Superusuario siempre tiene acceso
39        if (!empty($_SESSION['is_superuser'])) return true;
40        if (!isset($_SESSION['roles'])) return false;
41        $roleNames = (array) $roleNames;
42        return count(array_intersect($roleNames, $_SESSION['roles'])) > 0;
43    }
44
45    public static function isSuperAdmin() {
46        return !empty($_SESSION['is_superuser']);
47    }
48
49    public static function startSession($rememberMe = false) {
50        if ($rememberMe) {
51            // Sesión de 7 días
52            $lifetime = 7 * 24 * 60 * 60; // 7 días
53        } else {
54            // Sesión expira al cerrar navegador
55            $lifetime = 0;
56        }
57
58        session_set_cookie_params([
59            'lifetime' => $lifetime,
60            'path' => '/',

```

```
61 'httponly' => true,  
62 'samesite' => 'Lax'  
63 ]];  
64  
65 session_start();  
66  
67 // Regenerar ID por seguridad  
68 if (!isset($_SESSION['initiated'])) {  
69 session_regenerate_id(true);  
70 $_SESSION['initiated'] = true;  
71 }  
72  
73 // Timeout de inactividad (30 minutos)  
74 if (isset($_SESSION['last_activity']) &&  
75 (time() - $_SESSION['last_activity'] > 1800)) {  
76 session_unset();  
77 session_destroy();  
78 return false;  
79 }  
80  
81 $_SESSION['last_activity'] = time();  
82 return true;  
83 }  
84 }
```