**Canvas LTI Attendance System - Complete Code Documentation**

**Table of Contents**

1. [System Architecture](https://claude.ai/chat/a08c2866-0087-4219-bd29-4ab39b592337#system-architecture)
2. [Authentication Flow](https://claude.ai/chat/a08c2866-0087-4219-bd29-4ab39b592337#authentication-flow)
3. [Database Structure](https://claude.ai/chat/a08c2866-0087-4219-bd29-4ab39b592337#database-structure)
4. [API Routes](https://claude.ai/chat/a08c2866-0087-4219-bd29-4ab39b592337#api-routes)
5. [Frontend Components](https://claude.ai/chat/a08c2866-0087-4219-bd29-4ab39b592337#frontend-components)
6. [Data Flow](https://claude.ai/chat/a08c2866-0087-4219-bd29-4ab39b592337#data-flow)

**1. System Architecture**

**Overview**

This is a Next.js application that integrates with Canvas LMS using LTI 1.3 protocol for secure launch and OAuth 2.0 for API access.

Canvas LMS

↓

LTI 1.3 Launch → Next.js App → SQL Server Database

↓ ↓

OAuth 2.0 Flow Attendance Management

↓ ↓

Canvas API Student/Instructor Views

**Technology Stack**

* **Frontend**: Next.js 15 (React 19)
* **Backend**: Next.js API Routes
* **Database**: SQL Server (via mssql package)
* **Authentication**: LTI 1.3 + Canvas OAuth 2.0
* **Encryption**: AES-256-GCM for token storage
* **Web Server**: Nginx (reverse proxy)
* **Process Manager**: PM2

**2. Authentication Flow**

**2.1 LTI 1.3 Launch Sequence**

**Step 1: Login Initiation** (/api/lti/login/route.js)

Canvas → POST /api/lti/login

Parameters:

- iss: Canvas platform URL

- login\_hint: User identifier

- target\_link\_uri: Where to redirect after auth

- lti\_message\_hint: Session data

- client\_id: Your LTI client ID

What happens:

1. Receives login request from Canvas

2. Validates required parameters

3. Generates state and nonce for security

4. Redirects to Canvas authorization endpoint

**Code Breakdown**:

// /api/lti/login/route.js

export async function handleLogin(request) {

// 1. Parse parameters (handle both GET/POST)

// 2. Validate login\_hint and target\_link\_uri

// 3. Build authorization URL with:

// - response\_type: 'id\_token'

// - client\_id: from environment

// - redirect\_uri: target\_link\_uri

// - state & nonce: random security tokens

// 4. Redirect to Canvas for authentication

}

**Step 2: Launch Handler** (/api/lti/launch/route.js)

Canvas → POST /api/lti/launch (with id\_token)

What happens:

1. Receives JWT id\_token from Canvas

2. Decodes token to extract user/course data

3. Extracts SIS IDs from custom claims

4. Identifies user role (instructor/student)

5. Redirects to dashboard with user context

**Code Breakdown**:

// /api/lti/launch/route.js

export async function POST(request) {

// 1. Get id\_token from form data

const formData = await request.formData();

const id\_token = formData.get('id\_token');

// 2. Decode JWT (without verification for now)

const decoded = jwt.decode(id\_token);

// 3. Extract user information

const userInfo = {

user\_id: decoded?.sub,

user\_sis\_id: decoded?.['custom']?.user\_sis\_id, // SIS ID from Canvas

user\_name: decoded?.name,

course\_id: decoded?.['context']?.id,

course\_sis\_id: decoded?.['custom']?.course\_sis\_id, // Course SIS ID

course\_name: decoded?.['context']?.title,

roles: decoded?.['roles'], // Array of role URIs

nrps\_url: decoded?.['namesroleservice']?.context\_memberships\_url

};

// 4. Determine if instructor

const isInstructor = userInfo.roles.some(role =>

role.includes('Instructor') || role.includes('Teacher')

);

// 5. Redirect to dashboard

const redirectUrl = `/dashboard?token=${encodeURIComponent(JSON.stringify(userInfo))}`;

return new NextResponse(`<script>window.location.href='${redirectUrl}';</script>`);

}

**2.2 Canvas OAuth Flow**

**Purpose**: Get API access token to fetch roster data

**Step 1: Authorization Request** (/api/auth/canvas/route.js)

User clicks "Authorize" → GET /api/auth/canvas

What happens:

1. Receives user\_id, course\_id, SIS IDs

2. Creates state object with context

3. Redirects to Canvas OAuth authorization

**Code Breakdown**:

// /api/auth/canvas/route.js

export async function GET(request) {

const { searchParams } = new URL(request.url);

// 1. Get context from query params

const stateData = {

user\_id: searchParams.get('user\_id'),

user\_sis\_id: searchParams.get('user\_sis\_id'),

course\_id: searchParams.get('course\_id'),

course\_sis\_id: searchParams.get('course\_sis\_id')

};

// 2. Build OAuth URL

const params = new URLSearchParams({

client\_id: process.env.CANVAS\_API\_CLIENT\_ID,

response\_type: 'code',

redirect\_uri: 'https://yourdomain.com/api/auth/callback',

state: encodeURIComponent(JSON.stringify(stateData)),

scope: 'url:GET|/api/v1/courses/:course\_id/enrollments'

});

// 3. Redirect to Canvas

return NextResponse.redirect(`https://aui.instructure.com/login/oauth2/auth?${params}`);

}

**Step 2: Callback Handler** (/api/auth/callback/route.js)

Canvas → GET /api/auth/callback?code=xxx&state=xxx

What happens:

1. Receives authorization code

2. Exchanges code for access token

3. Encrypts and stores token in database

4. Closes popup window

**Code Breakdown**:

// /api/auth/callback/route.js

export async function GET(request) {

const { searchParams } = new URL(request.url);

const code = searchParams.get('code');

const state = searchParams.get('state');

// 1. Parse state to get context

const stateData = JSON.parse(decodeURIComponent(state));

// 2. Exchange code for token

const tokenResponse = await fetch('https://aui.instructure.com/login/oauth2/token', {

method: 'POST',

body: new URLSearchParams({

grant\_type: 'authorization\_code',

client\_id: process.env.CANVAS\_API\_CLIENT\_ID,

client\_secret: process.env.CANVAS\_API\_CLIENT\_SECRET,

code: code,

redirect\_uri: 'https://yourdomain.com/api/auth/callback'

})

});

const tokenData = await tokenResponse.json();

// 3. Store encrypted token

await storeCanvasToken(

stateData.user\_id,

stateData.course\_id,

tokenData.access\_token,

tokenData.refresh\_token,

tokenData.expires\_in // Usually 3600 seconds (1 hour)

);

// 4. Close popup and reload parent

return new NextResponse(`

<script>

if (window.opener) {

window.opener.location.reload();

window.close();

}

</script>

`);

}

**3. Database Structure**

**3.1 Token Encryption System**

**Why?** Access tokens are sensitive and must be encrypted at rest.

// /lib/db-mssql.js

class TokenEncryption {

constructor() {

this.algorithm = 'aes-256-gcm'; // Authenticated encryption

this.keyLength = 32; // 256 bits

this.ivLength = 16; // 128 bits

this.tagLength = 16; // 128 bits

this.saltLength = 64; // 512 bits

this.iterations = 100000; // PBKDF2 iterations

}

encrypt(text) {

// 1. Generate random salt

const salt = crypto.randomBytes(this.saltLength);

// 2. Derive key from password + salt

const key = crypto.pbkdf2Sync(

process.env.ENCRYPTION\_KEY,

salt,

this.iterations,

this.keyLength,

'sha256'

);

// 3. Generate random IV

const iv = crypto.randomBytes(this.ivLength);

// 4. Encrypt

const cipher = crypto.createCipheriv(this.algorithm, key, iv);

const encrypted = Buffer.concat([

cipher.update(text, 'utf8'),

cipher.final()

]);

// 5. Get authentication tag

const tag = cipher.getAuthTag();

// 6. Combine: salt + iv + tag + encrypted

const combined = Buffer.concat([salt, iv, tag, encrypted]);

return combined.toString('base64');

}

decrypt(encryptedText) {

// 1. Decode from base64

const combined = Buffer.from(encryptedText, 'base64');

// 2. Extract components

const salt = combined.slice(0, this.saltLength);

const iv = combined.slice(this.saltLength, this.saltLength + this.ivLength);

const tag = combined.slice(this.saltLength + this.ivLength, this.saltLength + this.ivLength + this.tagLength);

const encrypted = combined.slice(this.saltLength + this.ivLength + this.tagLength);

// 3. Derive same key

const key = crypto.pbkdf2Sync(

process.env.ENCRYPTION\_KEY,

salt,

this.iterations,

this.keyLength,

'sha256'

);

// 4. Decrypt

const decipher = crypto.createDecipheriv(this.algorithm, key, iv);

decipher.setAuthTag(tag);

const decrypted = Buffer.concat([

decipher.update(encrypted),

decipher.final()

]);

return decrypted.toString('utf8');

}

}

**3.2 Database Functions**

**Store Token**:

export async function storeCanvasToken(userId, courseId, accessToken, refreshToken, expiresIn) {

// 1. Encrypt tokens

const encryptedAccess = tokenEncryption.encrypt(accessToken);

const encryptedRefresh = refreshToken ? tokenEncryption.encrypt(refreshToken) : null;

// 2. Calculate expiry

const expiresAt = new Date(Date.now() + (expiresIn \* 1000));

// 3. Upsert into database

await pool.request()

.input('userId', sql.VarChar, userId)

.input('courseId', sql.VarChar, courseId)

.input('accessToken', sql.NVarChar(sql.MAX), encryptedAccess)

.input('refreshToken', sql.NVarChar(sql.MAX), encryptedRefresh)

.input('expiresAt', sql.DateTime2, expiresAt)

.query(`

MERGE canvas\_tokens AS target

USING (SELECT @userId AS user\_id, @courseId AS course\_id) AS source

ON target.user\_id = source.user\_id AND target.course\_id = source.course\_id

WHEN MATCHED THEN UPDATE SET ...

WHEN NOT MATCHED THEN INSERT ...

`);

}

**Get Token**:

export async function getCanvasToken(userId, courseId) {

// 1. Query for non-expired token

const result = await pool.request()

.input('userId', sql.VarChar, userId)

.input('courseId', sql.VarChar, courseId)

.query(`

SELECT access\_token, refresh\_token, expires\_at

FROM canvas\_tokens

WHERE user\_id = @userId

AND course\_id = @courseId

AND expires\_at > GETDATE() -- Only non-expired

`);

if (result.recordset.length === 0) return null;

// 2. Decrypt and return

return {

accessToken: tokenEncryption.decrypt(result.recordset[0].access\_token),

refreshToken: result.recordset[0].refresh\_token

? tokenEncryption.decrypt(result.recordset[0].refresh\_token)

: null

};

}

**Refresh Token**:

export async function refreshCanvasToken(userId, courseId) {

// 1. Get existing token data

const tokenData = await getCanvasToken(userId, courseId);

if (!tokenData?.refreshToken) return null;

// 2. Request new token from Canvas

const response = await fetch('https://aui.instructure.com/login/oauth2/token', {

method: 'POST',

body: new URLSearchParams({

grant\_type: 'refresh\_token',

client\_id: process.env.CANVAS\_API\_CLIENT\_ID,

client\_secret: process.env.CANVAS\_API\_CLIENT\_SECRET,

refresh\_token: tokenData.refreshToken

})

});

if (!response.ok) {

await deleteCanvasToken(userId, courseId);

return null;

}

const newTokenData = await response.json();

// 3. Store new token

await storeCanvasToken(

userId,

courseId,

newTokenData.access\_token,

newTokenData.refresh\_token || tokenData.refreshToken,

newTokenData.expires\_in

);

return newTokenData.access\_token;

}

**4. API Routes**

**4.1 Roster Fetching (/api/lti/roster/route.js)**

**Purpose**: Get list of students enrolled in a course

export async function POST(request) {

const { course\_id, user\_id, nrps\_url } = await request.json();

// 1. Extract numeric Canvas course ID

let numericCourseId;

if (nrps\_url) {

const match = nrps\_url.match(/courses\/(\d+)/);

numericCourseId = match ? match[1] : null;

}

// 2. Get or refresh access token

let accessToken = null;

const tokenData = await getCanvasToken(user\_id, course\_id);

if (tokenData) {

accessToken = tokenData.accessToken;

} else {

// Try to refresh

accessToken = await refreshCanvasToken(user\_id, course\_id);

}

// 3. If no token, return mock data and flag for auth

if (!accessToken) {

return NextResponse.json({

members: getMockRoster().members,

needsAuth: true

});

}

// 4. Fetch enrollments from Canvas API

const response = await fetch(

`https://aui.instructure.com/api/v1/courses/${numericCourseId}/enrollments?per\_page=100`,

{

headers: {

'Authorization': `Bearer ${accessToken}`,

'Accept': 'application/json'

}

}

);

// 5. Handle errors

if (response.status === 401) {

await deleteCanvasToken(user\_id, course\_id);

return NextResponse.json({ needsAuth: true });

}

// 6. Parse and format members

const enrollments = await response.json();

const members = enrollments

.filter(e => ['StudentEnrollment', 'TaEnrollment', 'TeacherEnrollment'].includes(e.type))

.map(enrollment => ({

user\_id: enrollment.user\_id?.toString(),

sis\_user\_id: enrollment.sis\_user\_id, // SIS ID!

name: enrollment.user?.name || 'Unknown',

email: enrollment.user?.email || '',

roles: [enrollment.type === 'TeacherEnrollment' ? 'Instructor' : 'Student'],

status: enrollment.enrollment\_state

}));

return NextResponse.json({ members, success: true });

}

**4.2 Mark Attendance (/api/attendance/mark/route.js)**

**Purpose**: Record or update a student's attendance status

**Flow**:

1. Check if record exists

2. Insert or Update attendance

3. Log change in audit table

4. Return success

export async function POST(request) {

const pool = await getPool();

const {

course\_id, course\_sis\_id, student\_id, student\_sis\_id,

status, date, session\_type, instructor\_id,

instructor\_sis\_id, course\_name, instructor\_name, marked\_time

} = await request.json();

// 1. Check for existing record

const existing = await pool.request()

.input('courseId', sql.VarChar, course\_id)

.input('date', sql.Date, date)

.input('sessionType', sql.VarChar, session\_type)

.input('studentId', sql.VarChar, student\_id)

.query(`

SELECT id, status FROM attendance

WHERE course\_id = @courseId

AND session\_date = @date

AND session\_type = @sessionType

AND student\_id = @studentId

`);

const isUpdate = existing.recordset.length > 0;

const oldStatus = isUpdate ? existing.recordset[0].status : null;

// 2. Insert or Update using MERGE

const result = await pool.request()

.input('courseId', sql.VarChar, course\_id)

.input('courseSisId', sql.VarChar, course\_sis\_id)

.input('courseName', sql.VarChar, course\_name)

.input('date', sql.Date, date)

.input('sessionType', sql.VarChar, session\_type)

.input('studentId', sql.VarChar, student\_id)

.input('studentSisId', sql.VarChar, student\_sis\_id)

.input('status', sql.VarChar, status)

.input('markedTime', sql.Time, marked\_time)

.input('instructorId', sql.VarChar, instructor\_id)

.input('instructorSisId', sql.VarChar, instructor\_sis\_id)

.input('instructorName', sql.VarChar, instructor\_name)

.query(`

MERGE attendance AS target

USING (

SELECT @courseId AS course\_id, @date AS session\_date,

@sessionType AS session\_type, @studentId AS student\_id

) AS source

ON target.course\_id = source.course\_id

AND target.session\_date = source.session\_date

AND target.session\_type = source.session\_type

AND target.student\_id = source.student\_id

WHEN MATCHED THEN

UPDATE SET

status = @status,

marked\_time = @markedTime,

marked\_at = GETDATE(),

marked\_by = @instructorId,

marked\_by\_sis\_id = @instructorSisId

WHEN NOT MATCHED THEN

INSERT (course\_id, course\_sis\_id, course\_name, session\_date, session\_type,

student\_id, student\_sis\_id, status, marked\_time, marked\_by,

marked\_by\_sis\_id, instructor\_name)

VALUES (@courseId, @courseSisId, @courseName, @date, @sessionType,

@studentId, @studentSisId, @status, @markedTime, @instructorId,

@instructorSisId, @instructorName)

OUTPUT INSERTED.id;

`);

const attendanceId = result.recordset[0].id;

// 3. Log to audit table

await pool.request()

.input('sessionId', sql.Int, attendanceId)

.input('studentId', sql.VarChar, student\_id)

.input('studentSisId', sql.VarChar, student\_sis\_id)

.input('courseSisId', sql.VarChar, course\_sis\_id)

.input('sessionType', sql.VarChar, session\_type)

.input('oldStatus', sql.VarChar, oldStatus)

.input('newStatus', sql.VarChar, status)

.input('changedBy', sql.VarChar, instructor\_id)

.input('changedBySisId', sql.VarChar, instructor\_sis\_id)

.input('classDate', sql.Date, date)

.input('markedTime', sql.Time, marked\_time)

.input('changeType', sql.VarChar, isUpdate ? 'update' : 'initial')

.query(`

INSERT INTO attendance\_audit (

session\_id, student\_id, course\_sis\_id, student\_sis\_id, session\_type,

old\_status, new\_status, changed\_by, changed\_by\_sis\_id,

class\_date, marked\_time, change\_type

)

VALUES (

@sessionId, @studentId, @courseSisId, @studentSisId, @sessionType,

@oldStatus, @newStatus, @changedBy, @changedBySisId,

@classDate, @markedTime, @changeType

)

`);

return NextResponse.json({ success: true, wasUpdate: isUpdate });

}

**4.3 Get Attendance (/api/attendance/get/route.js)**

**Purpose**: Retrieve attendance records for a specific date/session

export async function POST(request) {

const pool = await getPool();

const { course\_id, date, session\_type } = await request.json();

const result = await pool.request()

.input('courseId', sql.VarChar, course\_id)

.input('date', sql.Date, date)

.input('sessionType', sql.VarChar, session\_type)

.query(`

SELECT

student\_id,

student\_sis\_id,

status,

marked\_time,

marked\_by\_sis\_id

FROM attendance

WHERE course\_id = @courseId

AND session\_date = @date

AND session\_type = @sessionType

`);

return NextResponse.json({ records: result.recordset });

}

**4.4 Student Attendance (/api/attendance/student/route.js)**

**Purpose**: Get a student's complete attendance history

export async function POST(request) {

const pool = await getPool();

const { course\_sis\_id, student\_sis\_id } = await request.json();

// 1. Query by SIS IDs (important!)

const result = await pool.request()

.input('courseSisId', sql.VarChar, course\_sis\_id)

.input('studentSisId', sql.VarChar, student\_sis\_id)

.query(`

SELECT

session\_date,

session\_type,

status,

marked\_time,

marked\_at

FROM attendance

WHERE course\_sis\_id = @courseSisId

AND student\_sis\_id = @studentSisId

ORDER BY session\_date DESC, session\_type

`);

// 2. Calculate statistics

const records = result.recordset;

const stats = {

total: records.length,

present: records.filter(r => r.status === 'present').length,

absent: records.filter(r => r.status === 'absent').length,

late: records.filter(r => r.status === 'late').length,

excused: records.filter(r => r.status === 'excused').length

};

return NextResponse.json({ records, stats });

}

**4.5 Export Attendance (/api/attendance/export/route.js)**

**Purpose**: Export attendance data as CSV

export async function GET(request) {

const { searchParams } = new URL(request.url);

const course\_id = searchParams.get('course\_id');

const date\_from = searchParams.get('date\_from');

const date\_to = searchParams.get('date\_to');

const pool = await getPool();

// 1. Query attendance records

const result = await pool.request()

.input('courseId', sql.VarChar, course\_id)

.input('dateFrom', sql.Date, date\_from)

.input('dateTo', sql.Date, date\_to)

.query(`

SELECT

course\_sis\_id as SIS\_Course\_ID,

student\_sis\_id as SIS\_Student\_ID,

status as Attendance,

CONVERT(VARCHAR, session\_date, 23) as Class\_Date,

marked\_by\_sis\_id as SIS\_Teacher\_ID,

COALESCE(course\_name, course\_sis\_id) as Course\_Code,

COALESCE(instructor\_name, marked\_by\_sis\_id) as Teacher\_Name

FROM attendance

WHERE course\_id = @courseId

AND session\_date >= @dateFrom

AND session\_date <= @dateTo

AND student\_sis\_id IS NOT NULL

ORDER BY session\_date, student\_sis\_id

`);

// 2. Format as CSV

const headers = ['SIS\_Course\_ID', 'SIS\_Student\_ID', 'Attendance', 'Class\_Date', 'SIS\_Teacher\_ID', 'Course\_Code', 'Teacher\_Name'];

const csvRows = [

headers.join(','),

...result.recordset.map(row =>

`${row.SIS\_Course\_ID},${row.SIS\_Student\_ID},${row.Attendance},${row.Class\_Date},${row.SIS\_Teacher\_ID},"${row.Course\_Code}","${row.Teacher\_Name}"`

)

];

const csv = csvRows.join('\n');

const filename = `attendance\_${course\_id}\_${date\_from}.csv`;

// 3. Return as downloadable file

return new NextResponse(csv, {

headers: {

'Content-Type': 'text/csv; charset=utf-8',

'Content-Disposition': `attachment; filename="${filename}"`

}

});

}

**5. Frontend Components**

**5.1 Dashboard (/app/dashboard/page.js)**

**Purpose**: Main entry point after LTI launch, shows different views for instructors vs students

function DashboardContent() {

const [userInfo, setUserInfo] = useState(null);

const [activeTab, setActiveTab] = useState('attendance');

useEffect(() => {

// 1. Parse token from URL

const token = searchParams.get('token');

if (token) {

const decoded = JSON.parse(decodeURIComponent(token));

setUserInfo(decoded);

localStorage.setItem('lti\_context', JSON.stringify(decoded));

}

}, [searchParams]);

// 2. Show different views based on role

if (userInfo?.isInstructor) {

return (

<>

<div className="tabs">

<button onClick={() => setActiveTab('attendance')}>Take Attendance</button>

<button onClick={() => setActiveTab('reports')}>Reports</button>

</div>

{activeTab === 'attendance' && (

<iframe src={`/attendance?token=${encodeURIComponent(JSON.stringify(userInfo))}`} />

)}

{activeTab === 'reports' && (

<ReportsView userInfo={userInfo} />

)}

</>

);

} else {

// Student view

return (

<iframe src={`/student-attendance?token=${encodeURIComponent(JSON.stringify(userInfo))}`} />

);

}

}

**5.2 Attendance Page (/app/attendance/page.js)**

**Purpose**: Instructor interface for marking attendance

**Key Features**:

* Date picker with navigation
* Morning/Evening session toggle
* Student grid with status buttons
* Real-time stats
* Authorization flow for Canvas API

function AttendanceContent() {

const [roster, setRoster] = useState([]);

const [attendance, setAttendance] = useState({});

const [sessionType, setSessionType] = useState('morning');

const [selectedDate, setSelectedDate] = useState(today);

// 1. Load roster from Canvas API

const loadRoster = async (userInfo) => {

const response = await fetch('/api/lti/roster', {

method: 'POST',

body: JSON.stringify({

course\_id: userInfo.course\_id,

user\_id: userInfo.user\_id,

user\_sis\_id: userInfo.user\_sis\_id,

nrps\_url: userInfo.nrps\_url

})

});

const data = await response.json();

if (data.needsAuth) {

// Show authorization banner

setRoster([]);

} else {

setRoster(data.members);

}

};

// 2. Load existing attendance for date/session

const loadAttendance = async (userInfo, date, type) => {

const response = await fetch('/api/attendance/get', {

method: 'POST',

body: JSON.stringify({

course\_id: userInfo.course\_id,

date: date,

session\_type: type

})

});

const data = await response.json();

// Map records to state

const attendanceMap = {};

roster.forEach(student => {

attendanceMap[student.user\_id] = null;

});

data.records.forEach(record => {

attendanceMap[record.student\_id] = record.status;

});

setAttendance(attendanceMap);

};

// 3. Mark attendance for a student

const markAttendance = async (student, status) => {

const currentTime = new Date().toTimeString().split(' ')[0];

const previousStatus = attendance[student.user\_id];

// Confirm if changing status

if (previousStatus !== null && previousStatus !== status) {

const confirmed = window.confirm(

`Change ${student.name} from "${previousStatus}" to "${status}"?`

);

if (!confirmed) return;

}

// Optimistic update

setAttendance(prev => ({ ...prev, [student.user\_id]: status }));

try {

const response = await fetch('/api/attendance/mark', {

method: 'POST',

body: JSON.stringify({

course\_id: userInfo.course\_id,

course\_sis\_id: userInfo.course\_sis\_id,

student\_id: student.user\_id,

student\_sis\_id: student.sis\_user\_id,

status: status,

date: selectedDate,

session\_type: sessionType,

instructor\_id: userInfo.user\_id,

instructor\_sis\_id: userInfo.user\_sis\_id,

course\_name: userInfo.course\_name,

instructor\_name: userInfo.user\_name,

marked\_time: currentTime

})

});

if (!response.ok) throw new Error('Failed to save');

setMessage('✓ Saved');

} catch (error) {

// Revert on error

setAttendance(prev => ({ ...prev, [student.user\_id]: previousStatus }));

setMessage('Failed to save');

}

};

// 4. Handle authorization

const handleAuthorize = () => {

const params = new URLSearchParams({

course\_id: userInfo.course\_id,

user\_id: userInfo.user\_id,

user\_sis\_id: userInfo.user\_sis\_id,

course\_sis\_id: userInfo.course\_sis\_id

});

const popup = window.open(

`/api/auth/canvas?${params}`,

'canvas\_auth',

'width=600,height=700'

);

// Monitor popup closure

const checkPopup = setInterval(() => {

if (popup && popup.closed) {

clearInterval(checkPopup);

setTimeout(() => loadRoster(userInfo), 500);

}

}, 500);

};

return (

<div className="attendance-page">

{/\* Authorization Banner \*/}

{roster.length === 0 && (

<div className="auth-banner">

<button onClick={handleAuthorize}>Authorize Canvas Access</button>

</div>

)}

{/\* Date Controls \*/}

<div className="controls">

<input type="date" value={selectedDate} onChange={e => setSelectedDate(e.target.value)} />

<button onClick={() => setSessionType('morning')}>Morning</button>

<button onClick={() => setSessionType('evening')}>Evening</button>

</div>

{/\* Stats \*/}

<div className="stats-grid">

<div>Present: {stats.present}</div>

<div>Absent: {stats.absent}</div>

<div>Late: {stats.late}</div>

<div>Excused: {stats.excused}</div>

</div>

{/\* Student Grid \*/}

<div className="students-grid">

{roster.map(student => (

<div key={student.user\_id} className="student-card">

<h3>{student.name}</h3>

<div className="attendance-buttons">

<button

className={attendance[student.user\_id] === 'present' ? 'active' : ''}

onClick={() => markAttendance(student, 'present')}

>

✓ Present

</button>

<button

className={attendance[student.user\_id] === 'absent' ? 'active' : ''}

onClick={() => markAttendance(student, 'absent')}

>

✗ Absent

</button>

<button

className={attendance[student.user\_id] === 'late' ? 'active' : ''}

onClick={() => markAttendance(student, 'late')}

>

⏰ Late

</button>

<button

className={attendance[student.user\_id] === 'excused' ? 'active' : ''}

onClick={() => markAttendance(student, 'excused')}

>

✉ Excused

</button>

</div>

</div>

))}

</div>

</div>

);

}

**5.3 Student Attendance View (/app/student-attendance/page.js)**

**Purpose**: Show student their own attendance history

function StudentAttendanceContent() {

const [records, setRecords] = useState([]);

const [stats, setStats] = useState(null);

useEffect(() => {

const token = searchParams.get('token');

if (token) {

const userInfo = JSON.parse(decodeURIComponent(token));

loadAttendance(userInfo);

}

}, [searchParams]);

const loadAttendance = async (userInfo) => {

const response = await fetch('/api/attendance/student', {

method: 'POST',

body: JSON.stringify({

course\_sis\_id: userInfo.course\_sis\_id,

student\_sis\_id: userInfo.user\_sis\_id // Use SIS ID!

})

});

const data = await response.json();

setRecords(data.records);

setStats(data.stats);

};

const attendanceRate = stats

? Math.round(((stats.present + stats.late) / stats.total) \* 100)

: 0;

return (

<div className="student-dashboard">

{/\* Circular Progress \*/}

<div className="attendance-rate-card">

<div className="rate-circle">

<div className="rate-percentage">{attendanceRate}%</div>

<div className="rate-label">Attendance</div>

</div>

</div>

{/\* Stats Cards \*/}

<div className="stats-overview">

<div className="stat-card">

<div className="stat-label">Present</div>

<div className="stat-value">{stats.present}</div>

</div>

<div className="stat-card">

<div className="stat-label">Absent</div>

<div className="stat-value">{stats.absent}</div>

</div>

{/\* ... more stats ... \*/}

</div>

{/\* History \*/}

<div className="records-section">

<h2>Attendance History</h2>

{records.map((record, idx) => (

<div key={idx} className="record-item">

<div>{formatDate(record.session\_date)}</div>

<div>{record.session\_type}</div>

<div className={`status-${record.status}`}>{record.status}</div>

</div>

))}

</div>

</div>

);

}

**6. Data Flow**

**6.1 Complete Flow Diagram**

┌─────────────────────────────────────────────────────────────┐

│ Canvas LMS │

└──────────────────┬──────────────────────────────────────────┘

│

│ 1. LTI Launch

↓

┌─────────────────────────────────────────────────────────────┐

│ /api/lti/login (Login Initiation) │

│ - Receives: iss, login\_hint, target\_link\_uri │

│ - Generates: state, nonce │

│ - Redirects: Canvas /authorize\_redirect │

└──────────────────┬──────────────────────────────────────────┘

│

│ 2. Canvas Authorization

↓

┌─────────────────────────────────────────────────────────────┐

│ /api/lti/launch (Launch Handler) │

│ - Receives: id\_token (JWT) │

│ - Extracts: user info, course info, SIS IDs, roles │

│ - Redirects: /dashboard with token │

└──────────────────┬──────────────────────────────────────────┘

│

│ 3. Dashboard Loads

↓

┌─────────────────────────────────────────────────────────────┐

│ /dashboard │

│ If Instructor: If Student: │

│ - Show /attendance page - Show /student-attendance│

└──────────────────┬──────────────────────────────────────────┘

│

┌──────────┴──────────┐

│ │

↓ ↓

┌──────────────────┐ ┌──────────────────┐

│ Instructor │ │ Student │

│ Workflow │ │ Workflow │

└──────────────────┘ └──────────────────┘

│ │

│ 4a. Need Roster │ 4b. Load Own Data

↓ ↓

┌──────────────────┐ ┌──────────────────┐

│ /api/auth/ │ │ /api/attendance/ │

│ canvas │ │ student │

│ (OAuth) │ │ - Query by │

│ │ │ SIS IDs │

│ 5. Get Access │ │ - Return │

│ Token │ │ records + │

│ │ │ stats │

│ 6. Store in DB │ └──────────────────┘

│ (Encrypted) │

└──────────────────┘

│

│ 7. Fetch Roster

↓

┌──────────────────┐

│ /api/lti/roster │

│ - Get token │

│ - Call Canvas │

│ - Return │

│ students │

└──────────────────┘

│

│ 8. Mark Attendance

↓

┌──────────────────┐

│ /api/attendance/ │

│ mark │

│ - Check │

│ existing │

│ - MERGE into │

│ attendance │

│ - Log to audit │

└──────────────────┘

│

│ 9. Load Attendance

↓

┌──────────────────┐

│ /api/attendance/ │

│ get │

│ - Query by │

│ course, date │

│ - Return │

│ records │

└──────────────────┘

**6.2 Key Data Identifiers**

**Why SIS IDs Matter**:

* Canvas IDs change between dev/test/prod
* SIS IDs are stable institutional identifiers
* Use SIS IDs for:
  + Linking student records
  + Export to institutional systems
  + Cross-referencing with other systems

**ID Mapping**:

Canvas User ID (user\_id): "6567"

↓ Maps to

Canvas SIS User ID (student\_sis\_id): "11111" (from Banner/SIS)

Canvas Course ID (course\_id): "de967266d1282c1d4f566a48dda4047f86d81651"

↓ Maps to

Canvas SIS Course ID (course\_sis\_id): "7-test" (from Banner/SIS)

**Storage Strategy**:

attendance table:

- course\_id: Canvas internal ID (for API calls)

- course\_sis\_id: Institutional ID (for exports)

- student\_id: Canvas internal ID (for API calls)

- student\_sis\_id: Institutional ID (for exports)

**7. Security Considerations**

**7.1 Token Security**

* Access tokens encrypted with AES-256-GCM
* Salt + IV unique per encryption
* PBKDF2 key derivation (100,000 iterations)
* Tokens automatically expire (1 hour from Canvas)
* Refresh tokens used to get new access tokens

**7.2 LTI Security**

* State/nonce prevent replay attacks
* JWT validation (should be added)
* HTTPS required
* No client-side secrets

**7.3 SQL Injection Prevention**

* Parameterized queries throughout
* Input validation
* Type-safe parameters

**8. Common Issues & Solutions**

**Issue: "No SIS IDs in roster"**

**Cause**: Canvas not configured to include SIS IDs in enrollment API  
**Solution**: Admin → Settings → Feature Options → Enable "Include SIS ID in enrollment export"

**Issue: "Token expired"**

**Cause**: Canvas tokens expire after 1 hour  
**Solution**: System automatically refreshes using refresh\_token

**Issue: "Student can't see attendance"**

**Cause**: Query uses wrong ID type  
**Solution**: Ensure querying by student\_sis\_id, not student\_id

**Issue: "Dates show as Invalid Date"**

**Cause**: SQL Server returns dates in different format  
**Solution**: Use CONVERT(VARCHAR, date, 23) for ISO format

This documentation provides a complete understanding of the codebase. Each section explains not just what the code does, but why it's structured that way and how the pieces fit together.