Manual Técnico y Bitácora del Proyecto: Evalúa+

Autores: Miguel Barajas, Daniel Calvo
Asistente de IA: Gemini (Google)
Fechas del Sprint: 15 de octubre de 2025 - 24 de octubre de 2025

1. Visión General del Proyecto

Evalúa+ es una plataforma web diseñada para modernizar la evaluación educativa. El proyecto reemplaza los exámenes tradicionales en papel por un sistema interactivo, dinámico y con elementos de gamificación.

1.1. Objetivos Principales

- **Para Docentes:** Optimizar el proceso de evaluación mediante la creación de exámenes personalizados, la reutilización de preguntas (Banco de Preguntas) y la calificación automática, liberando tiempo para el análisis pedagógico.
- **Para Estudiantes:** Ofrecer una experiencia de evaluación moderna, atractiva y con retroalimentación instantánea que fomenta un ciclo de aprendizaje continuo.

1.2. Alcance del Sprint

El proyecto se ejecutó en un sprint intensivo de 10 días, desarrollando una aplicación web funcional, segura y desplegada en producción, desde cero.

2. Arquitectura de la Solución

2.1. Stack Tecnológico

Se seleccionó un stack enfocado en la simplicidad, la velocidad de desarrollo y el aprendizaje de los fundamentos de la web, evitando frameworks complejos.

- **Frontend:** HTML5, CSS3, JavaScript (Vanilla JS).
- **Backend & Base de Datos (BaaS):** Google Firebase.
  - **Firebase Authentication:** Gestión de registro e inicio de sesión por Correo/Contraseña.
  - **Firestore Database:** Base de datos NoSQL para toda la información de la aplicación.
  - **Firebase Hosting:** Despliegue y alojamiento de la aplicación en un CDN global.
- **Bibliotecas Adicionales (CDN):**
  - **Toastify JS:** Para notificaciones no intrusivas.
  - **Chart.js:** Para la visualización de datos en el panel de analíticas.

2.2. Modelo de Datos (Firestore)

La arquitectura de la base de datos se compone de las siguientes colecciones principales:

- users: Almacena la información de cada usuario (nombre, email, rol: 'docente' o 'estudiante'). El uid de Authentication se usa como ID del documento.
- salas: Contiene cada evaluación. Almacena el título, materia, docenteId (propietario), codigoAcceso (para unirse), limiteTiempo, clasesAsignadas (array de IDs de clases) y un array preguntas (que contiene los objetos de cada pregunta).
- resultados: Guarda un documento por cada evaluación completada. Incluye salaId, estudianteId, nombre, calificación, respuestas del estudiante y fecha.
- bancoPreguntas: Almacena preguntas reutilizables. Cada documento es una pregunta con su docenteId (propietario), materia y estructura de pregunta.
- clases: Permite agrupar estudiantes. Almacena el nombre, materia, docenteId (propietario), codigoClase (para inscripción) y un array estudiantesIds.

3. Metodología

El desarrollo se basó en una metodología de "Chats Secuenciales", donde cada sesión de trabajo (chat) tenía una misión específica, construyendo sobre el contexto y el código de la sesión anterior.

- **Control de Versiones:** Git y GitHub (rama main).
- **Documentación:** Bitácora continua al final de cada fase.

4. Bitácora Detallada del Desarrollo

Registro cronológico de las fases de construcción y los hitos alcanzados.

Fase 0: Cimientos del Proyecto

- **Objetivo:** Preparar el entorno de desarrollo y la estructura base.
- **Hitos Clave:**
  - Refinamiento de la index.html (Landing Page) inicial.
  - Creación de la estructura de carpetas profesional (/css, /js, /assets).
  - Configuración del repositorio en Git y GitHub, incluyendo el primer commit y la gestión de archivos .gitkeep.

Fase 1: El Arquitecto

- **Objetivo:** Configurar Firebase y construir el sistema de autenticación.
- **Hitos Clave:**
  - Creación del proyecto en Firebase y activación de Authentication (Email/Pass) y Firestore (modo de prueba).
  - Creación de login.html y register.html con sus estilos (css/auth.css).
  - Implementación de js/auth.js para manejar createUserWithEmailAndPassword y signInWithEmailAndPassword.
  - **Integración Crítica:** Al registrar un usuario, se crea un documento en la colección

users con su uid, nombre y rol (docente/estudiante).

Fase 2: El Creador

- **Objetivo:** Construir el panel de control del docente para crear salas de evaluación.
- **Hitos Clave:**
  - Creación de docente.html, css/docente.css y js/docente.js.
  - **Guardián de Ruta:** Implementación de un filtro de seguridad en js/docente.js usando onAuthStateChanged (autenticación) y getDoc (autorización por rol) para proteger el panel.
  - Implementación de la función handleCreateRoom que guarda nuevas evaluaciones en la colección salas con un codigoAcceso único y el docenteId.
  - Implementación de displayTeacherRooms usando query y where para mostrar solo las salas creadas por el docente.

Fase 3: El Participante

- **Objetivo:** Construir el panel del estudiante para unirse a salas.
- **Hitos Clave:**
  - Creación de estudiante.html, css/estudiante.css y js/estudiante.js.
  - Implementación de un Guardián de Ruta específico para el rol 'estudiante'.
  - Implementación de handleJoinRoom, que utiliza query y where("codigoAcceso", ...) para buscar y validar la sala.
  - Personalización de la interfaz ("Bienvenido, [Nombre]").
  - Implementación del botón de "Cerrar Sesión" (signOut).

Fase 4: El Analista

- **Objetivo:** Conectar los flujos: añadir preguntas, responderlas y calificarlas.
- **Hitos Clave:**
  - Modificación de salas para incluir un array preguntas (inicialmente vacío).
  - Implementación de handleAddQuestion (usando prompt()) que actualiza la sala con updateDoc y arrayUnion.
  - Implementación de startEvaluation y displayQuestion en js/estudiante.js para renderizar las preguntas y opciones (radio buttons).
  - Creación de la colección resultados.
  - Implementación de handleFinishEvaluation que calcula la puntuación y guarda el registro en la colección resultados.
  - El estudiante ve su calificación final inmediatamente.

Fase 5: El Pulidor

- **Objetivo:** Mejorar la calidad de vida: visualización de resultados e historial.
- **Hitos Clave:**
  - **Panel Docente:** Implementación de handleShowResults, que consulta la colección resultados (filtrando por salaId) para mostrar la lista de estudiantes y sus

calificaciones.
    - **Panel Estudiante:** Implementación de displayStudentHistory, que consulta resultados (filtrando por estudianteId) y realiza una consulta anidada a salas para mostrar un historial completo.
    - **Refactorización de UX:** Modificación de js/auth.js. Tras el login, ahora consulta el rol del usuario y redirige directamente al panel (docente.html o estudiante.html), mejorando la fluidez.

Fase 6: El Optimizador de Flujo

- **Objetivo:** Implementar redirección inteligente en la página de inicio.
- **Hitos Clave:**
    - Modificación de js/script.js (cargado en index.html).
    - Uso de onAuthStateChanged y getDoc para verificar si hay una sesión activa.
    - Si el usuario está autenticado, los botones de "Acceder" se actualizan dinámicamente para dirigirlo a su panel correspondiente (docente.html o estudiante.html) sin pasar por el login.

Fase 7: El Diseñador de Interfaz

- **Objetivo:** Reemplazar prompt() por un modal moderno y añadir campos de feedback.
- **Hitos Clave:**
    - Creación de una estructura modal en docente.html (inicialmente oculta).
    - Implementación de openAddQuestionModal y closeAddQuestionModal en js/docente.js.
    - **Mejora Pedagógica:** El modal incluye campos para feedbackCorrecto y feedbackIncorrecto.
    - La estructura de datos de las preguntas guardadas en Firestore ahora incluye estos campos de retroalimentación.

Fase 8: El Editor

- **Objetivo:** Implementar la funcionalidad de Editar y Eliminar preguntas (CRUD).
- **Hitos Clave:**
    - Creación de una nueva vista de gestión (manage-questions-view) en docente.html.
    - Implementación de displayQuestionsForRoom que lista las preguntas y añade botones de "Editar" y "Eliminar".
    - **Lógica de Eliminación:** handleDeleteQuestion usa confirm(), obtiene los datos de la sala, crea un nuevo array de preguntas (usando .filter()) y actualiza el documento con updateDoc.
    - **Lógica de Edición:** Reutilización inteligente del modal de la Fase 7. Se usa una variable de estado (editingQuestionIndex) para poblar el modal con los datos de la pregunta existente y guardar los cambios en el índice correcto del array.

Fase 9: El Bibliotecario

- **Objetivo:** Implementar la funcionalidad de "Banco de Preguntas".
- **Hitos Clave:**
  - Creación de la nueva colección bancoPreguntas en Firestore.
  - Modificación del modal de "Añadir Pregunta" para incluir un checkbox "Guardar en mi Banco".
  - **Lógica de Guardado Dual:** handleQuestionSubmit ahora, opcionalmente, realiza un addDoc a bancoPreguntas además de actualizar la sala.
  - Creación de un segundo modal ("Importar desde Banco") que consulta y muestra las preguntas del banco (filtradas por docenteId).
  - Implementación de la importación múltiple usando arrayUnion para añadir las preguntas seleccionadas a la evaluación actual.

Fase 10: El Tutor

- **Objetivo:** Implementar la revisión de evaluaciones para el estudiante (feedback).
- **Hitos Clave:**
  - Creación de un modal de revisión en estudiante.html.
  - Implementación de la función asíncrona showReview(resultId).
  - **Orquestación de Datos:** La función consulta resultados (para obtener las respuestas del estudiante) y anida una consulta a salas (para obtener las preguntas, opciones, respuesta correcta y *feedbacks*).
  - El modal renderiza una comparación visual (con código de colores) mostrando la respuesta del estudiante, la correcta y el feedback personalizado del docente.
  - Mejora de UX: El logo de "Evalúa+" ahora es un enlace a index.html en todas las páginas.

Fase 11: El Guardián de la Fortaleza

- **Objetivo:** Implementar Reglas de Seguridad de Firestore.
- **Hitos Clave:**
  - Reemplazo del "modo de prueba" por un archivo firestore.rules robusto.
  - Creación de funciones de ayuda (isAuth, isOwner, getUserData, isRole).
  - **Reglas implementadas:**
    - users: Solo el propietario puede leer/escribir sus propios datos.
    - salas: Solo 'docentes' pueden crear. Solo el docenteId propietario puede actualizar/borrar.
    - resultados: Solo 'estudiantes' pueden crear (para sí mismos). Docentes solo pueden leer resultados de sus salas. Estudiantes solo leen los suyos. Nadie puede actualizar/borrar.
    - bancoPreguntas: Solo el docenteId propietario puede realizar cualquier operación (CRUD).

Fase 12: El Desplegador

- **Objetivo:** Desplegar la aplicación en Firebase Hosting.

- **Hitos Clave:**
  - Instalación y autenticación de firebase-tools (Firebase CLI).
  - Ejecución de firebase init hosting.
  - Configuración clave: Se especificó . (directorio raíz) como el directorio público y se denegó la configuración de "Single-Page App (SPA)".
  - Ejecución exitosa de firebase deploy.
  - La aplicación quedó pública en la URL de Firebase Hosting.
  - Se versionaron los archivos firebase.json y .firebaserc.

Fase 13: El Pulidor de Interfaz

- **Objetivo:** Reemplazar alert() y confirm() por notificaciones "Toast" y spinners.
- **Hitos Clave:**
  - Integración de la biblioteca **Toastify JS** vía CDN en todos los archivos HTML.
  - Diseño de un indicador de carga (spinner) animado puramente con CSS.
  - Refactorización de js/auth.js para usar Toastify (verde para éxito, rojo para error) en lugar de alert().
  - Reemplazo de los "Cargando..." por el nuevo spinner de CSS.
  - Reemplazo de confirm() (al eliminar preguntas) por un modal de confirmación personalizado, mejorando la seguridad y la UX.

Fase 14: El Analista de Datos

- **Objetivo:** Construir la lógica de backend (en cliente) para un Dashboard de Analíticas.
- **Hitos Clave:**
  - Creación de la vista #analytics-view en docente.html.
  - Implementación de handleShowAnalytics. Esta función consulta salas (para la estructura de la prueba) y resultados (para todos los envíos).
  - **Motor de Cálculo:** La función procesa los datos para calcular:
    - Calificación Promedio.
    - Tasa de Aprobación (ej. > 60%).
    - Preguntas Más Difíciles/Fáciles (basado en un conteo de aciertos/fallos por pregunta).
  - Los resultados se renderizan como texto en la vista de analíticas.

Fase 15: El Visualizador

- **Objetivo:** Integrar Chart.js para visualizar las analíticas.
- **Hitos Clave:**
  - Integración de **Chart.js** vía CDN en docente.html.
  - Reemplazo de los contenedores de texto de analíticas por elementos <canvas>.
  - Implementación de gestión de estado para destruir instancias de gráficos (.destroy()) antes de dibujar nuevas, previniendo errores visuales.
  - Renderizado de un **Gráfico de Dona** (Doughnut) para la Tasa de Aprobación (Aprobados vs. Reprobados).

- Renderizado de un **Gráfico de Barras Horizontales** (Horizontal Bar) para el rendimiento por pregunta (Aciertos vs. Fallos).

Fase 16: El Cronometrador

- **Objetivo:** Implementar evaluaciones con límite de tiempo.
- **Hitos Clave:**
  - Modificación del formulario de creación de salas (docente) para incluir un campo opcional "Límite de Tiempo (minutos)".
  - El limiteTiempo (en minutos) se guarda en el documento de la sala.
  - Creación de la interfaz #timer-display en estudiante.html.
  - Implementación de startTimer(endTime) en js/estudiante.js que usa setInterval para la cuenta regresiva.
  - **Auto-Envío:** Si el temporizador llega a cero, se invoca automáticamente handleFinishEvaluation.
  - handleFinishEvaluation fue modificado para limpiar (clearInterval) cualquier temporizador activo si el estudiante envía manualmente.

Fase 17: El Binario

- **Objetivo:** Implementar un nuevo tipo de pregunta: Verdadero o Falso.
- **Hitos Clave:**
  - Modificación del modal del docente para incluir un <select> de tipo de pregunta ("Opción Múltiple", "Verdadero o Falso").
  - La interfaz del modal se vuelve dinámica: muestra/oculta los campos de opciones según el tipo seleccionado.
  - **Evolución de Datos:** Las preguntas ahora incluyen un campo tipo ('multipleChoice' o 'trueFalse').
  - Se corrigió un bug de validación deshabilitando (disabled = true) los campos required que estaban ocultos.
  - **Panel Estudiante:** displayQuestion fue refactorizado para leer el question.tipo y renderizar las opciones correctas (4 opciones o 2).
  - Se aseguró la retrocompatibilidad para preguntas antiguas (sin campo tipo).

Fase 18: El Validador

- **Objetivo:** Implementar un tercer tipo de pregunta: Respuesta Corta (Texto Exacto).
- **Hitos Clave:**
  - Extensión del <select> de tipo de pregunta para incluir "Respuesta Corta".
  - La interfaz del modal ahora muestra un único campo de texto para la respuesta correcta.
  - La estructura de datos para este tipo es: tipo: 'shortAnswer', opciones: null, correcta: "Texto exacto".
  - **Panel Estudiante:** displayQuestion renderiza un <input type="text"> para este tipo.
  - saveCurrentAnswer fue actualizado para leer el .value.trim() del input de texto si

existe, o el radio button si no.
    - La función de revisión showReview fue adaptada para mostrar la respuesta del estudiante y la respuesta correcta para este tipo.

Fase 19-20: Omitidas

- **Nota:** Las fases "El Ilustrador" (imágenes) y "El Motivador" (gamificación) fueron omitidas estratégicamente para priorizar la gestión de datos.

Fase 21: El Registrador

- **Objetivo:** Implementar la exportación de resultados a CSV.
- **Hitos Clave:**
    - Añadido el botón "Exportar a CSV" a la vista de resultados del docente.
    - Optimización: Los datos de los resultados se guardan en una variable de caché (currentResultsData) al mostrarlos, evitando consultas duplicadas.
    - Creación de dos funciones auxiliares en Vanilla JS:
        1. generateCSV(data): Convierte el array de objetos JSON a un string CSV.
        2. downloadCSV(csvContent, fileName): Crea un Blob, una URL de objeto y simula un clic en un enlace <a> para iniciar la descarga.
    - handleExportCSV orquesta el proceso y obtiene el nombre de la sala para el archivo.

Fase 22: El Arquitecto de Clases

- **Objetivo:** Implementar la infraestructura para un sistema de "Clases".
- **Hitos Clave:**
    - Creación de la nueva colección clases en Firestore (con docenteId, codigoClase, estudiantesIds).
    - Actualización de firestore.rules para la nueva colección.
    - **Panel Docente:** Implementación del formulario y lógica (handleCreateClass) para crear clases.
    - **Panel Estudiante:** Implementación del formulario y lógica (handleJoinClass) para unirse a una clase usando arrayUnion.
    - Ambos paneles ahora listan las clases a las que pertenecen (displayTeacherClasses, displayStudentClasses).

Fase 23: El Asignador

- **Objetivo:** Conectar los sistemas de Clases y Evaluaciones.
- **Hitos Clave:**
    - **Evolución de Datos:** Se añadió el campo clasesAsignadas (Array de IDs de clase) a la colección salas.
    - **Panel Docente:**
        - El formulario de "Crear Sala" ahora incluye un <select multiple> que se puebla con las clases del docente.
        - handleCreateRoom guarda los IDs de las clases seleccionadas en

clasesAsignadas.
    - **Panel Estudiante:**
        - El panel se rediseñó para mostrar "Mis Evaluaciones Asignadas".
        - Se creó displayAssignedEvaluations:
            1. Consulta las clases del estudiante.
            2. Realiza una segunda consulta a salas usando where("clasesAsignadas", "array-contains-any", ...).
        - Las evaluaciones asignadas se muestran como tarjetas con un botón "Comenzar", eliminando la necesidad de códigos.

Fase 24: Omitida

- **Nota:** La fase de rediseño visual se pospuso para priorizar la refactorización de la experiencia de evaluación.

Fase 25: El Arquitecto de Enfoque

- **Objetivo:** Mover la toma de evaluaciones a una página dedicada y libre de distracciones.
- **Hitos Clave:**
    - Creación de tres nuevos archivos: evaluacion.html, css/evaluacion.css, y js/evaluacion.js.
    - evaluacion.html se diseñó con una UI mínima (solo la prueba) para maximizar el enfoque.
    - **Refactorización:** Los puntos de entrada en js/estudiante.js (unirse por código o comenzar asignada) fueron modificados. Ahora ejecutan window.open('evaluacion.html?roomid=...'), abriendo la prueba en una nueva pestaña.
    - **Nuevo Motor:** js/evaluacion.js implementa su propio guardián de ruta (lee roomid de la URL, verifica auth y rol) y carga la evaluación.
    - Se migró toda la lógica de la prueba (timer, display, navegación) a js/evaluacion.js.
    - **Flujo de Cierre:** Al finalizar, handleFinishEvaluation en la nueva página guarda el resultado, muestra la puntuación por 3.5 segundos y ejecuta window.close(), cerrando la pestaña automáticamente.
    - Se limpió js/estudiante.js, eliminando toda la lógica de ejecución de pruebas.

Fases 26-27: El Aplicador Dinámico (Tematización)

- **Objetivo:** Implementar temas de color dinámicos basados en el rol del usuario.
- **Hitos Clave:**
    - (Fase 26 - CSS) Se definieron paletas de colores CSS (variables) para theme-docente (verde) y theme-estudiante (naranja).
    - (Fase 27 - JS) Se implementó la lógica para aplicar estas clases:
        - js/docente.js y js/estudiante.js: Los guardianes de ruta inyectan la clase de tema correspondiente en la etiqueta <body>.
        - js/evaluacion.js: Hereda el tema theme-estudiante.

    - js/script.js: La página de inicio (index.html) se tematiza si el usuario ya está logueado.
    - js/auth.js: La página de registro (register.html) cambia de tema interactivamente al seleccionar el rol.

5. Estado Final y Conclusión

En un sprint de 10 días (15 al 24 de octubre de 2025), el proyecto Evalúa+ pasó de ser una idea a una aplicación web completamente funcional, segura y desplegada en producción.

La plataforma final incluye:

- Dos roles de usuario (Docente, Estudiante) con paneles y flujos de trabajo distintos.
- Un sistema de autenticación robusto.
- Funcionalidad completa de creación, gestión (CRUD) e importación (Banco) de evaluaciones.
- Soporte para 3 tipos de preguntas (Opción Múltiple, V/F, Respuesta Corta).
- Soporte para evaluaciones con límite de tiempo.
- Calificación automática y un sistema de revisión con feedback pedagógico.
- Un sistema de "Clases" para asignación directa de evaluaciones.
- Un Dashboard de Analíticas con visualización de datos (gráficos).
- Exportación de resultados a CSV.
- Una experiencia de usuario pulida con notificaciones "Toast", spinners y tematización dinámica.
- Reglas de seguridad de Firestore granulares.
- Despliegue exitoso en Firebase Hosting.