https://github.com/jjcastro/well-col.git

http://wells-col.herokuapp.com/

Descripción de las decisiones de arquitectura tomadas para la elaboración de Well-Col

Juan Guillermo Murillo Castillo 201423979  
Juan Sebastián Gómez 201414022  
Juan José Castro 201415057  
David Mauricio Delgado Ruiz 201422209

Experimento 1

Team OverFlow

Contenido

[Decisiones de arquitectura 3](#_Toc460440054)

[Análisis 3](#_Toc460440055)

[Modelo 3](#_Toc460440056)

[Requerimientos 5](#_Toc460440057)

[Escenarios de calidad 6](#_Toc460440058)

[Restricciones 7](#_Toc460440059)

[Casos de uso 8](#_Toc460440060)

[Diseño de la arquitectura 9](#_Toc460440061)

[Patrones 9](#_Toc460440062)

[Tácticas 10](#_Toc460440063)

[Tecnologías 10](#_Toc460440064)

[Diseño de experimento 11](#_Toc460440065)

[Plan 11](#_Toc460440066)

[Tecnologías 11](#_Toc460440067)

[Implementación y ejecución de prototipos y experimentos 11](#_Toc460440068)

[Planeación resto del ciclo de desarrollo 11](#_Toc460440069)

[Depósito de las fuentes 11](#_Toc460440070)

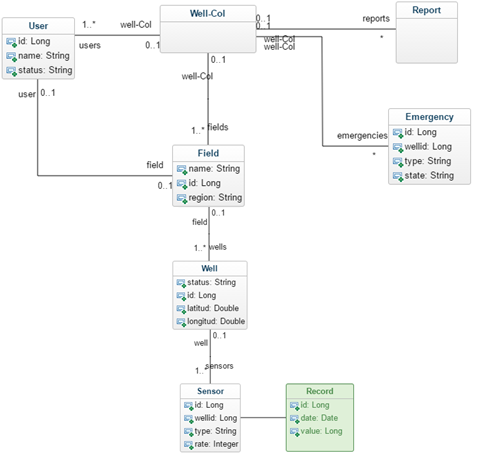
[Despliegue de la aplicación 11](#_Toc460440071)

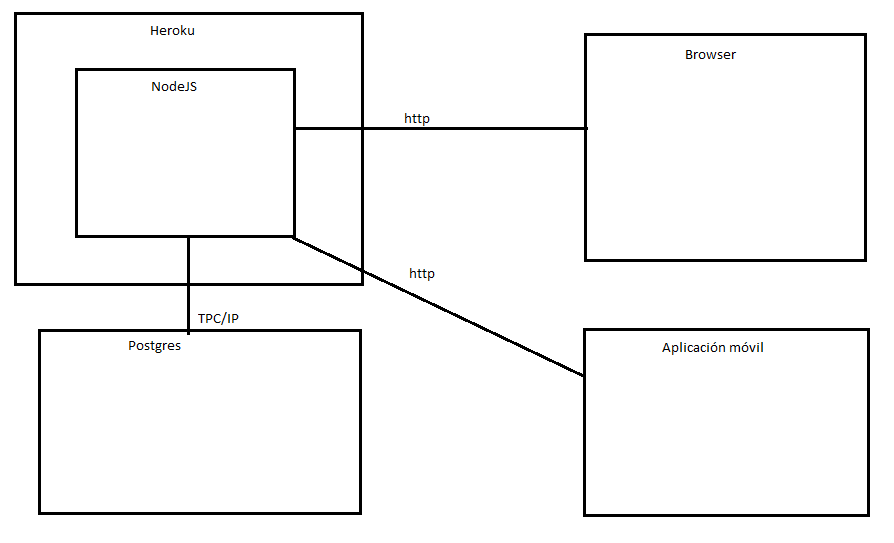
# Decisiones de arquitectura

Las decisiones de arquitectura que TeamOverFlow tomó dependieron directamente de los atributos de calidad que se consideraron críticos para el desarrollo de la aplicación. Estos atributos de calidad fueron el desempeño, la interoperabilidad, disponibilidad, seguridad y confiabilidad de la información. Para la elaboración de la aplicación se usó el método de diseño centrado en la arquitectura.

## Análisis

### Modelo





### Requerimientos

##### Requerimientos funcionales

1. Generar reportes (mensual, trimestral, semestral y anual) del consumo energético y producción de fluido (crudo y agua) de un pozo, campo o zona geográfica escogida, cuando el jefe de producción lo requiera
2. Notificar emergencias relacionadas sólo con los pozos del campo asignado al jefe. Las notificaciones llegan a los dispositivos móviles del jefe.
3. Generar reportes (diarios y semanales) de consumo energético y producción de fluido (crudo o agua) de un pozo dado, cuando el jefe de campo lo requiera.
4. CRUD para cada una de las entidades identificadas
5. Generar reportes por periodo de tiempo, pozo, campo, zona geográfica, caudal diario, consumo de energía y temperatura.
6. Recibir valores de 4 diferentes tipos de sensores

##### Requerimientos no funcionales

1. La aplicación debe ser escalable.
2. El equipo también posee una conexión satelital para comunicarse a un sistema central y transmitir toda la información de los sensores.
3. Latencia de 4800 solicitudes/segundo
4. Interfaz amigable para un fácil uso y capacitación
5. Tiempo de respuesta para generación de reportes menor a 10000 ms. (Se aclara que en este punto se tiene en cuenta que la desviación estándar de los reportes es grande, dado que un reporte semanal para un pozo no procesará la misma cantidad de datos que un reporte anual para una región.

### Escenarios de calidad

Dentro de los escenarios de calidad encontramos los mencionados anterior mente

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Tipo de atributo** | **Prioridad** |
| EC1 | Desempeño | Alta |
| **Fuente** | | |
| sensores | | |
| **Estímulo** | | |
| Registrar valor según tipo de sensor | | |
| **Ambiente** | | |
| Sobrecargado | | |
| **Medida esperada** | | |
| El rendimiento del sistema al procesar los datos debe ser 4800 solicitudes en 1000 ms | | |

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Tipo de atributo** | **Prioridad** |
| EC2 | Seguridad | Alta |
| **Fuente** | | |
| Sistema | | |
| **Estímulo** | | |
| Mantener la información segura | | |
| **Ambiente** | | |
| Normal | | |
| **Medida esperada** | | |
| La seguridad del sistema debe permitir que no se filtre la información de ningún pozo petrolero ni sensor | | |

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Tipo de atributo** | **Prioridad** |
| EC3 | Disponibilidad | Alta |
| **Fuente** | | |
| Sistema | | |
| **Estímulo** | | |
| Mantener la información disponible | | |
| **Ambiente** | | |
| Normal | | |
| **Medida esperada** | | |
| El sistema debe tener una disponibilidad de 350 días del año | | |

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Tipo de atributo** | **Prioridad** |
| EC4 | Confiabilidad de la información | Alta |
| **Fuente** | | |
| Sistema | | |
| **Estímulo** | | |
| Mantener la información persistente | | |
| **Ambiente** | | |
| Normal | | |
| **Medida esperada** | | |
| Se debe poder recuperar el 100% de los datos de cualquier sensor en el sistema | | |

### Restricciones

#### Stakeholders

* Límite de 4 meses para el desarrollo del sistema

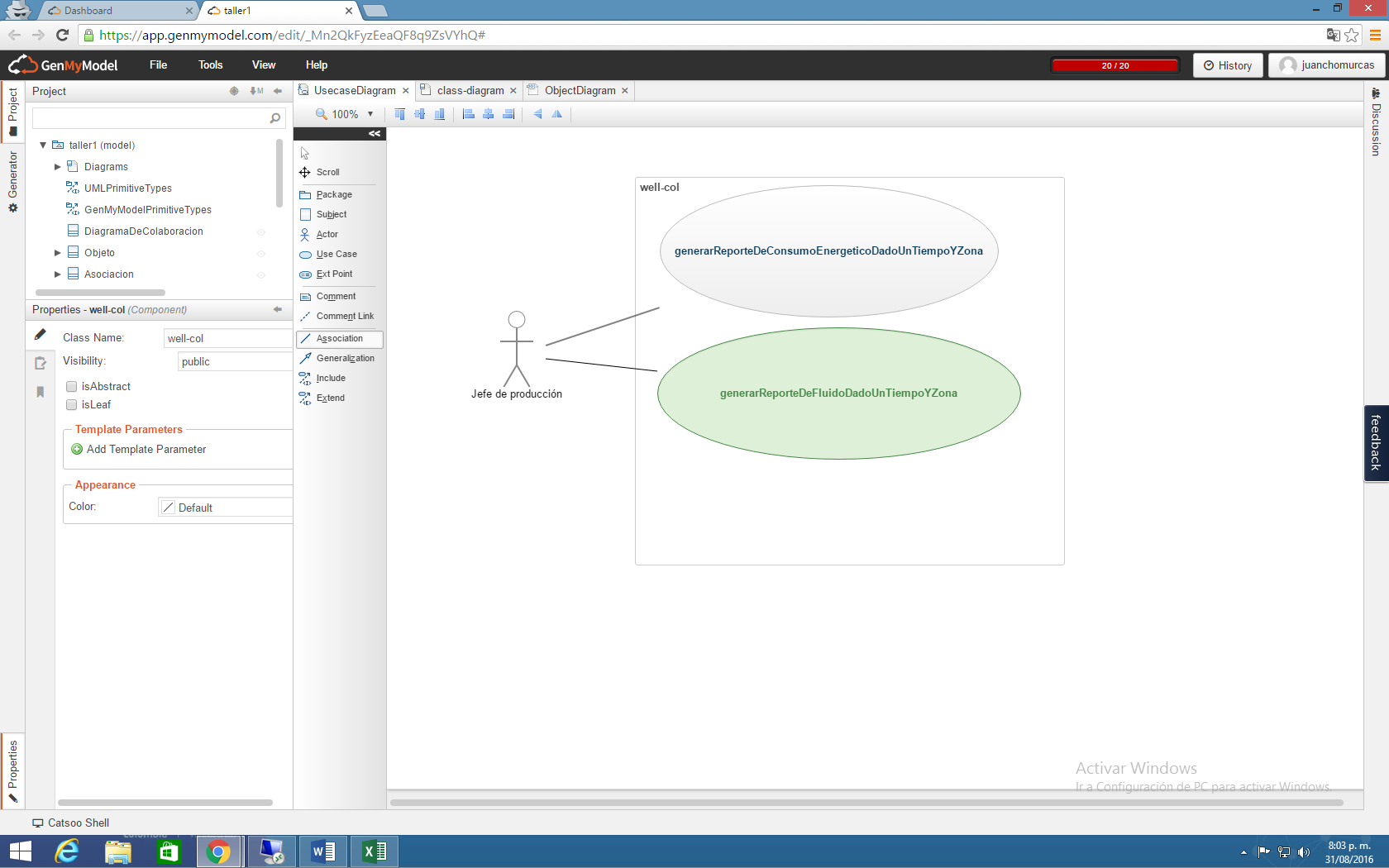
#### Tecnología

* Tecnología asíncrona, de alta disponibilidad y desempeño

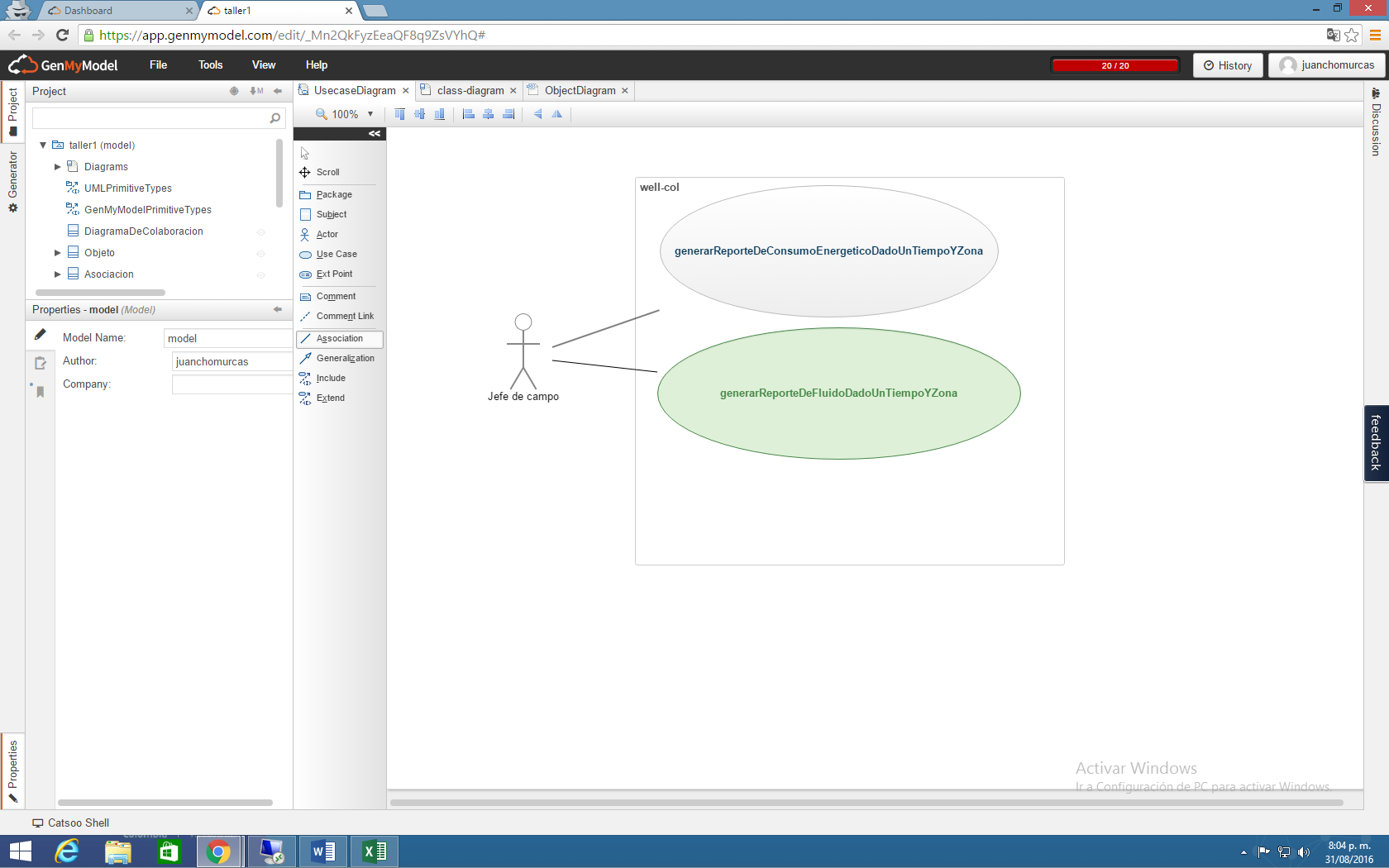
#### Negocio

* Interfaz web
* Interfaz móvil nativa
* El equipo también posee una conexión satelital para comunicarse a un sistema central y transmitir toda la información de los sensores.

### Casos de uso



* El tiempo puede ser mensual, trimestral, semestral y anual
* La zona puede ser un pozo, campo o zona geográfica escogida
* El fluido puede ser crudo o agua



* El tiempo puede ser diario o semanal
* La zona puede ser un pozo
* El fluido puede ser crudo o agua

## Diseño de la arquitectura

### Patrones

El patrón que se usó fue el patrón Modelo Vista Controlador, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y l modulo encargado de gestionar los eventos y las comunicaciones. . Para esta entrega parcial se desarrolló parte del modelo, la capa de lógica del negocio, entre estos está el CRUD de las entidades principales que se identificación como:

* Well: Representa un pozo petrolero que tiene asociado 4 sensores, uno de cada tipo, éste a su vez tiene un status que puede ser Abierto, Producción, Parado o Clausurado según la lógica de negocio. Dicho pozo tiene además un id, una latitud y longitud para saber su ubicación en el mapa.
* Sensor: un sensor tiene asociado un id propio, un id del Well al cual pertenece, un tipo (type) que indica si es de caudal, consumo, temperatura y emergencia. Tiene también el atributo rate que define cada cuantos segundos llega información de dicho sensor y un listado records que representa las un envío de información de dicho sensor que contiene un id, una fecha y un valor.
* User: Representa un usuario de la aplicación, dicho usuario tiene acceso a diferente tipo de información dependiendo de su rol, o en el contexto de la aplicación dependiendo de su estatus. Dicho usuario tiene un status que puede ser Jefe de Campo o Jefe de producción, también un id, identificador único personal que lo diferencia de los demás
* Field: Representa un campo en la aplicación y tiene atributos como nombre, id, región y a su vez tiene varios Well (o pozos).
* Report:
* Field: Un Field representa un campo que tiene un nombre, un id, un id de región a la que pertenece y un listado de pozos (Wells).
* Emergency: Esta representa una emergencia en la aplicación, tiene atributos como id, id del pozo de la emergencia, el tipo y el estado.

### Tácticas

Algunas de las tácticas que se usaron y se piensan implementar para el desempeño fueron:

* *Manage Sampling Rate*
* *Limit Event Response*
* *Prioritize Events*
* *Reduce Overhead*
* *Bound Execution Times*
* *Increase Resource Efficiency*

### Tecnologías

Las tecnologías que se usaron para el desarrollo de la aplicación fueron:

* **NodeJS**: es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.
* **ExpressJS**: es una plataforma ligera para la construcción de aplicaciones web usando NodeJS. Ayuda a organizar las aplicaciones web en el servidor. Los ExpressJS website31 describe Express como "un marco de aplicación web Node.js mínima y flexible".
* **MongoDB:** es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.
* **AngularJS:** es un *framework* de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de [Modelo Vista Whatever](https://es.wikipedia.org/w/index.php?title=Modelo_Vista_Whatever&action=edit&redlink=1) (MVW), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

## Diseño de experimento

### Plan

Se realizaron pruebas de carga sobre los métodos llevándolos hasta que se incumplieran los escenarios de calidad. Para esto primero se usó Postman para la prueba de los servicios y finalmente para la escalabilidad se usó JMeter testeando sobre todo las solicitudes de tipo POST sobre los sensores que son las que más solicitudes deben recibir.

### Tecnologías

* JMeter
* Cluster de JMeter
* Postman

## Implementación y ejecución de prototipos y experimentos

## Planeación resto del ciclo de desarrollo

Para el resto del ciclo se planea implementar la base de datos con MongoDB que es una base de datos noSQL y su implementación es sencilla. Además se tiene anexo planeado un cronograma para reuniones semanales todos los viernes de 7 de la mañana hasta las 9:30 en donde se planteará los objetivos y se mostraran todos los avances.

# Depósito de las fuentes

El depósito de preferencia para almacenar tanto el código fuente como su documentación fue GitHub debido a su facilidad para el trabajo en equipo con su sistema de versiones.

El enlace del proyecto es el siguiente:

<https://github.com/jjcastro/well-col.git>

# Despliegue de la aplicación

La aplicación se desplegó en Heroku y se encuentra disponible en el siguiente enlace:

<http://wells-col.herokuapp.com/>