UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Integrantes
Lobo, Brian
Taberna, Alexis
Bulbulian, Juan Pablo

Ayudante Asignado: Perez Albiero, Alejandro

Materia: (TACS) Tecnologías avanzadas para la construcción de software

Índice

Toma de decisiones

En este documento se habla de la toma de decisiones relacionadas con tecnologías y de negocio que se acordaron para el desarrollo de este trabajo práctico.

Requerimientos funcionales y no funcionales a más detalle:

- https://docs.google.com/document/d/1Dvydn8WtLzCuI7VqrAdAHL8FNXPYnDTZrio7c-WV_Kk/pub

Requerimientos pedidos por el TP:

- **Sistema de versionado:** Github
- **Lenguaje:** java, decidimos usar la versión 1.8 por ser la más estable y nueva.
- **Manejo de dependencias:** Maven
- **Lugar donde corre la aplicacion:** tecnologías Cloud
- **Metodología:** ágil
- **Tipo de Aplicación:** Web

Tecnologías Elegidas:

- Front-end
    - **Angular 5:** Utilizamos esta versión por ser la más reciente y conocida por el equipo, otras opciones fueron AngularJs, React y JavaScript puro, pero por las ventajas de sus herramientas, conocimientos básicos de los miembros del equipo y por la comunidad que trabaja y da soporte detrás, fuimos por esta opción.

    - **NodeJs:** Aunque posea muchas más herramientas optamos por usar su gestor de dependencias **npm** y su facilidad de uso a nivel programación e instalación, esto también nos permite tener el front separado del back, para el caso que deban estar en servidores distintos, agregando más portabilidad al producto.

    - **Bootstrap 4:** Utilizamos templates gratis sacadas de internet, lo que permite ahorrar tiempo de desarrollo en aspectos visuales y centrarnos más en la parte de seguridad y buena integración con el back de la aplicación, ya que consideramos que el *look-and-feel* es un aspecto importante, pero lo es más la seguridad que ofrece este software al trabajar directamente con dinero virtual. Como agregado este template viene con integración para idiomas así como la capacidad de ser responsive, sin cambiar mucho su uso, brindándole

de fácil aprendizaje y una sensación similar al usarlo desde un smartphone como de una PC.

- Telegram Bot
  - **Descripción:** Se trata de una aplicación que utiliza una biblioteca recomendada por la compañía Telegram para interactuar con la API que ellos mismos ofrecen al público. El bot puede ser encontrado en el directorio de Telegram bajo el nombre ***TACSCryptoBot***. Consta de cinco comandos, los cuales permiten cumplimentar las user stories encomendadas en el enunciado del Trabajo Práctico.

  - **Tipo:** se trata de un ***long polling bot***, a diferencia del tipo ***webhook***, la aplicación monitorea los servidores de Telegram en busca de updates que se hayan dado en el bot. La decisión se tomó en base a la posibilidad que daba el servicio de Cloud en Heroku, si se elegía el tipo webhook, se requería tener asignado un puerto fijo, cosa que Heroku no brinda en las cuentas gratuitas.

  - **Implementación:** es muy sencilla, desde una clase main se invoca a la clase *TACSCryptoBot.java*, la cual tiene la especificación de los comandos, los cuales están dentro de el paquete command. Todos heredan de una clase común *Command.java* y corren su metodo *exec*.

  - **Seguridad:** el Token del bot, el cual fue provisto por la compañía Telegram al crear el bot es pasado por parámetro de línea de comando al iniciar el bot. De esta manera, solo es conocido por el creador del Bot, que es quien tiene la responsabilidad también de correrlo en el entorno Cloud.

  - **Deploy:** la aplicación se encuentra actualmente corriendo en el servicio de Cloud Heroku.

  - **Tecnologías:** La aplicación está desarrollada en **Java 1.8**, la decisión se tomó al investigar que las bibliotecas y frameworks disponibles en la comunidad harían que la implementación sea bastante sencilla.
    Se utilizó el cliente web de **Spring Framework** para las request al Backend, y la biblioteca **Jackson** para el manejo de los datos en formato **JSON**.

- Backend
  - **Estructura:** se optó en una estructura de mono-aplicación la cual era más simple de crear y mantener para lo que el negocio requería, no posee una lógica muy compleja ni abundante, por lo que decidimos un modelo clásico formado por una estructura de proyectos en módulos dominados por un POM padre para su sencillo despliegue. El proyecto cuenta con una parte Core, donde se encuentran dependencias de Jars externos y objetos de

importancia alta y compartida por las demás partes, una parte Model donde se encuentra los objetos de negocio (POJO's), otra ApiWeb, con todo lo necesario para levantar la aplicación web y una estructura de servicios basada en un modelo clásico formado por una capa rest, para manejar los pedidos http, otra service, para la lógica de negocio y otra DAO para el manejo a los datos, ya sean en BD o externos por otra API.

    - **Lenguaje:** Java 1.8, la elegimos sobre la 1.7 por poseer mejores herramientas para manejo de listas, funcional y mejoras en Interfaces

    - **ORM:** Elegimos Hibernate por su robustez, amplia comunidad y facilidad a la hora de persistir objetos.

    - **Web-Service:** Optamos por una combinación entre Spring y Jersey.

    - **Server:** Fuimos por Apache Tomcat, también por su facilidad de uso y comunidad que brinda soporte en caso de alguna duda técnica

- **Programas para desarrollo:** Tales como Eclipse, Dbeaver, Insomnia, Postman, Visual Studio Code, Git Kraken, entre otros, son herramientas para desarrollo Free u Open Source, además de multiplataforma, para su disponibilidad según el SO de cada integrante.

- **Base de Datos:** Elegimos una base de datos no relacional, más específicamente MongoDB, en un principio teníamos pensado hacerla en una relacional, pero llegamos a la conclusión que no necesitamos de la estructura de relación entre datos que estas poseen, es cierto que tenemos relaciones entre datos, pero no son de tal magnitud como para ir por ese tipo de base de datos, en su lugar optamos por una relacional basada en archivos, MongoDB es simple de instalar, configurar y muy versátil, siendo una herramienta sencilla para programar y ágil a la hora de obtener datos de esta, es cierto que parte de la lógica que pensamos en stored procedures quedaron del lado del backend, pero ganamos en su lugar algo muy simple y rápido para obtener objetos por ids, así como facilidad para almacenarlos.

Administración del proyecto:

- **Herramientas:** empleamos un sistemas de tickets administrado por Trello, también gratis y de simple uso ya que es online y web.

- **Metodología**: Optamos por usar time box development, la cual deja sin cambios el calendario (Fechas de entrega) y prioriza funcionalidades a cumplir en cada entregable (Funcionalidades obligatorias y agregados), descartando ideas principalmente por falta de tiempo.

Negocio:

- **Compra y venta:** se decidió ir por comprar monedas a la aplicación, así como un “Banco”, no entre usuarios

- **Monedas Físicas:** Solo se trabaja con Dólar Estadounidense.

- **Depositos**: seran aceptados por un usuario administrador a mano

Tipo de Aplicación

- Como la aplicación era WEB decidimos hacerla responsive, para que pueda ser utilizada en varios dispositivos sin importar SO o hardware

- Permitimos que usuarios puedan manejar sus cuentas al mismo tiempo desde sus celulares o tablets al mismo tiempo que desde su PC, permitiendo retomar lo que se estaba haciendo en en el escritorio en cualquier lugar desde un smartphone sin necesidad de instalar nada

- La aplicación fue pensada para agregar clientes que la consuman, siendo diseñada de tal forma que el dia de mañana se puedan utilizar sus servicios de una app nativa, de esta forma funciona Telegram, quien consume de los mismo servicios que la aplicación WEB sin distinción

- También fue pensada para escalar, permitiendo tener instancias en varios servidores que, correctamente configurados, puedan trabajar en conjunto escalan horizontalmente

- Se tomó el trabajo de dejar la app lo más genérico posible para poder usar el proyecto para otros futuros en caso de necesitar algo funcional rápido