Ideas para el proyecto de Proyecto Software (3º Ing. Informática, EINA, UNIZAR)

INDICE

1. Objetivos de este documento

El objetivo de este documento es dar un conjunto de ideas de base para los proyectos de la asignatura. En general es buena idea que:

1. Cojáis una aplicación similar que os guste para que os sirva de inspiración.
2. Penséis en las cosas que las aplicaciones que usáis no tienen u os gustaría que tuvieran de otra manera y las consideréis también.

Es decir, que no hay que limitarse a seleccionar requisitos de los que hay aquí, podéis aportar ideas propias que tengan sentido para el tipo de aplicaciones pedidas. Tampoco entendáis necesariamente que algunas de las funciones que se proponen son obligatorias y otras no. **Vuestro trabajo es presentar una propuesta que os permita entregar un proyecto coherente** y no un conjunto de funcionalidades diversas y a medio implementar, y que sea de una complejidad y tamaño suficientes para alcanzar vuestro objetivo de nota (os ayudaremos a concretar eso).

La última sección da algunas ideas de grandes rasgos sobre tecnologías que pueden usarse en el proyecto, y de algunas limitaciones que os ponemos. Leedla antes de tomar decisiones al respecto.

2. Juegos online

- Puede estar centrado en juegos de cartas (pueden ser tipo el Cinquillo y la Brisca, o el Magic: The Gathering), de tablero (pueden ser la Oca y el Parchís, el Conecta 4, el Trivial o estilo el Colonos de Catán) u otro estilo de juegos "casual". Puede ser un único juego muy trabajado y detallado, o haber varios implementados con más sencillez. No recomendamos inventarse las reglas de un juego (mucho esfuerzo que no os compensa).
- No estamos pensando en videojuegos sino en juegos de mesa basados en turnos y que puedan jugarse online (puede ser con todas las personas en la misma habitación, cada uno en su móvil, aunque también podría haber alguno que se pudiera jugar entre varios p.ej. con una única tablet).
- Puede ser una app móvil, web y/o de escritorio.
- La funcionalidad básica depende de cada juego, pero se tiene que poder jugar una partida completa entre al menos dos jugadores humanos (pero según el juego se deberán permitir más), y determinar quién ha ganado (puede ser necesario llevar una puntuación).
- Según el juego se pueden montar partidas síncronas (los jugadores tienen que estar conectados al mismo tiempo) o se pueden montar partidas asíncronas (un jugador hace su movimiento o su elección, le llega una notificación al otro que, cuando quiere, hace el suyo etc.). En el caso de las partidas síncronas el juego debe poder ponerse en pausa (dejar la partida a medias) cuando se quiera para retomarlo más adelante (entre los mismos jugadores, claro).
- Puede haber juegos para una persona (el solitario). Lo que no buscamos es que se pueda jugar contra la máquina cuando no haya contendientes humanos (dejad eso para la versión 2).
- Si el juego lo permite, se puede dejar que los jugadores elijan el aspecto (tipos y colores de las fichas, elegir una imagen de fondo para el tablero, una imagen de fondo para los naipes etc.).
- Se debería poder guardar una partida a medias para retomarla después. También guardar un histórico de resultados de las distintas partidas, poder consultarlo, ver tu evolución en el tiempo jugando a cierto juego etc.

- Sincronización entre dispositivos: una partida dejada a medias, o bien una partida asíncrona debe poder dejarse en el móvil y retomarla desde la tablet, o desde la aplicación web o la de escritorio.
- Aspectos de red social: incluir la posibilidad de registros, amigos, comparar las puntuaciones de los juegos entre amigos, desafiarse, organizar torneos…

3. Lector de libros/comics electrónicos

- Puede estar centrado en la reproducción de libros, cómics o ambos. Soportar formatos habituales (epub, pdf, cbr/cbz, texto plano...).
- También puede incluir cierta funcionalidad de agregador de noticias (acceso a fuentes con resúmenes en formatos RSS o Atom, por ejemplo).
- Puede ser una app móvil, web y/o de escritorio.
- Tiene que incluir funcionalidad básica de lectura: ver contenidos, avanzar y retroceder páginas, acceder y navegar en la tabla de contenidos (si el formato la soporta y el documento tiene).
- Aumentar o disminuir el tamaño de la fuente (si el formato lo permite), seleccionar combinación de colores (de texto y de fondo) y tipografía (si el formato lo permite), o al menos la posibilidad de hacer zooms (para formatos como PDF o comics).
- Seleccionar fragmentos de texto y poder guardarlos o compartirlos (con otras apps en el dispositivo, enviarlos por e-mail...).
- Poder buscar palabras en un diccionario marcándolas sobre el texto.
- Puntos de lectura (*bookmarks*). Tener uno por defecto por cada libro que se esté leyendo (en la última página vista), pero poder crear los que se quieran para marcar páginas de interés.
- Buscar entre tus libros, bookmarks, fragmentos de texto guardados etc. (dentro de la app).
- Gestión de la colección. Clasificar, etiquetar, puntuar libros ya leídos etc.
- Búsqueda y descarga de libros libres disponibles en Internet.
- Sincronización entre dispositivos: poder dejar la lectura en el móvil y retomarla desde la tablet, o desde la aplicación web o la de escritorio. Acceso desde distintos dispositivos a los mismos libros, bookmarks etc.
- Aspectos de red social: incluir la posibilidad de registros, amigos, compartir textos seleccionados de libros con otros, recomendar libros...
- Buscar cosas relacionadas en Internet (p.ej. a partir de un libro buscar otros de la misma autora, o de temática similar).

4. Gestor de contraseñas

- Inspiración: LastPass
- Puede haber un cliente móvil, desktop, web, e incluso un plugin de navegador.
- Debe permitirte guardar todas tus contraseñas online, y cifradas con una única contraseña maestra. El sistema debe soportar múltiples usuarios, que se registren, almacenen online sus contraseñas y puedan recuperarlas desde los distintos clientes.
- Te puede ofrecer distintas alternativas de cifrado robusto.
- Organizadas por categorías (personal, profesional…) que puedas crear.
- Que permita buscarlas por texto libre, por fecha de creación/actualización, por categoría...

- Además de guardar pares usuario-contraseña, debe permitir guardar información adicional: texto libre con lo que quieras, URLs, archivos (p.ej. una tarjeta de claves escaneada en JPG) etc. Todo ello cifrado, claro.
- Las contraseñas pueden tener una fecha de expiración. Una vez superada la aplicación te exige cambiarlas.
- Te puede generar automáticamente contraseñas robustas (y te dice cómo de robustas son).
- Te puede ofrecer un generador de contraseñas personalizable donde tú le digas qué puede y no llevar las contraseñas que te genera (ver por ejemplo KeePass, menú Tools > Generate Password…).
- La aplicación puede recordar los cambios que hayas ido haciendo, de manera que puedas recuperar versiones antiguas de tus contraseñas si te hiciera falta.
- Se puede implementar *Two Factor Authentication* (2FA). Puede ser algo relativamente simple como que la aplicación además de pedirte un password te envíe un código por e-mail que tienes que usar para poder abrirla y descifrar el contenido. O que exija que además de un password tengas en tu posesión un cierto fichero que se generó a la vez que ese password. O se puede hacer una aplicación de móvil cuyo único papel sea el de verificar que eres tú a través de ella.

5. Tecnologías

Tenéis bastante libertad para decidir las tecnologías que vais a usar para el proyecto. Elegirlas y aceptar los problemas que tengáis con ellas son parte de las responsabilidades que tenéis que asumir al llevar a cabo un proyecto de software. Pero hay algunas cosas que chocan con los objetivos de la asignatura. Teniendo en cuenta que este tema cambia de año año y es difícil poner unas reglas fijas, vamos a ver algunas pautas generales, aunque la última palabra sobre lo que se puede y no usar la tendremos los profesores:

- Esencialmente cualquier lenguaje, *framework*, biblioteca o componente que queráis lo podéis usar. Lo que no podéis hacer es coger un proyecto ya existente y similar a lo que queréis hacer, p.ej. un proyecto de software libre, modificarlo ligeramente y presentarlo como vuestro.
- Respecto al despliegue de vuestro software (para desarrollo, integración, y para las demostraciones del mismo que nos tendréis que hacer), damos unas pautas y normas al final de este documento.
- Os recomendamos encarecidamente que no tratéis de aprender muchas tecnologías nuevas con el proyecto. El proyecto ya os va a costar bastante más esfuerzo del que pensáis sin añadir a eso factores de incertidumbre tecnológica adicionales. Ninguna tecnología es tan fácil como parece en sus tutoriales o en las charlas de alguien que la está usando, ni tan productiva como promete su página web. Esto es especialmente cierto cuando la metes en un proyecto no trivial con un equipo de tamaño mediano.
- Os recomendamos encarecidamente que no incluyáis algo innecesario[1] en el proyecto solo porque acabéis de aprender a usarlo, o porque lleváis tiempo queriendo probarlo o porque habéis oído hablar de ello y os parece interesante o porque lo-usan-no-sé-dónde y esa-gente-sí-que-sabe. Si estás empeñado en blandir un martillo, vas a ver clavos por todas partes. Esto es difícil verlo uno mismo, así que deberíais escuchar a vuestros compañeros de equipo, e incluso a vuestros profesores .

1 Un ejemplo que pasa de vez en cuando es la idea de incluir algo (p.ej. una caché de datos o escalabilidad dinámica) para solucionar unos problemas de prestaciones que nunca vais a tener en la asignatura.

- Tened en cuenta también que es un trabajo en equipo. Si solo hay una persona que controle ciertas tecnologías, esa persona acabará siendo cuello de botella y/o acabará sobrecargado de trabajo. No todo el equipo tiene que saber hacerlo todo, pero tampoco es bueno que haya cosas que solo las conozca una persona, sobre todo si son fundamentales para el éxito del proyecto.
- Si elegís algo que no conocéis, por lo menos tratad de elegir algo que lleve existiendo bastantes años: estará mejor documentado, tendrá más dudas resueltas en Stack Overflow y es más probable que en el futuro lo podáis usar para otras cosas.
- Un buen diseño de software te permite combinar tecnologías más fácilmente, porque habrás separado claramente los distintos problemas (cliente, API, persistencia, comunicaciones…) y por tanto sus implementaciones podrán separarse más fácilmente. No es buena idea elegir toda la tecnología antes de al menos haber diseñado una primera idea de la arquitectura de tu aplicación. Diseño de software y elección de tecnología deberían ir siempre de la mano.

Para los que no tengáis nada claro qué *stack* tecnológico sería adecuado para este proyecto, o para aquellos grupos cuyas únicas referencias son lo que habéis visto en la carrera hasta ahora, os podemos hacer algunas sugerencias. No son recomendaciones, son sugerencias para ayudaros a acotar vuestra búsqueda de tecnologías. **Estas sugerencias no son para que esquivéis vuestra responsabilidad a la hora de elegir tecnologías. Seguís siendo responsables de las elecciones tecnológicas que hagáis para el proyecto. Buscad, elegid, probad, leed, y aseguraos de que podéis defender vuestras elecciones tecnológicas de manera razonada.**

También queremos señalar que **estas solo son sugerencias para la asignatura de Proyecto Software**, por el tipo de proyectos que se hacen, por las cosas que se ven en asignaturas previas, y por las experiencias que tenemos de otros años. Para otros tipos de proyectos y otras circunstancias, quizás usaríamos o sugeriríamos cosas distintas.

Lenguajes de programación

Para el tipo de proyectos que proponemos, las opciones más habituales y que menos problemas os van a dar, además de que al menos algo ya sabéis o tendréis que aprender casi seguro en vuestra vida son **Java**, **Python** y **JavaScript/TypeScript**.

Algunos lenguajes como **Dart** o **Kotlin** se están usando más en la asignatura los últimos años, sobre todo porque tienen algunos frameworks para el desarrollo de aplicaciones móviles y web que son bastante potentes. Siguen siendo muy minoritarios si los comparamos con los anteriores.

Si tienes ideas muy claras sobre cuál es lenguaje-de-programación-único-para-dominarlos-a-todos, o el auténtico lenguaje-de-los-brogramadores[2]-de-verdad, o el überlenguaje que en su duodécima encarnación, esta vez sí, por fin, destronará al lenguaje-cuyo-nombre-no-debe-ser-pronunciado, esta parte del documento igual no es para ti.

¿Frameworks, bibliotecas, stacks?

Muy brevemente, si tu aplicación se basa en frameworks, estos te proporcionan un esqueleto/andamiaje (*scaffolding*) de una aplicación (o de una parte de la aplicación, p.ej. de su front-end) y tu trabajo es "rellenar los huecos". Mientras que si te basas en bibliotecas, tu trabajo es establecer la estructura de la aplicación, y llamar a las bibliotecas cuando las necesites.

Los frameworks son un mecanismo que permite disciplinar a un grupo de desarrolladores para que les sea más difícil hacer un mal diseño de ciertos aspectos del sistema, pero por eso mismo son más rígidos. Las bibliotecas te dan más libertad, pero más libertad viene acompañada de más responsabilidad. Los frameworks te proporcionan una visión uniforme de las bibliotecas que usan por debajo, pero esa visión uniforme suele hacer más complicada la depuración de fallos.

2 No es una errata: https://en.wikipedia.org/wiki/Brogrammer

Stack tecnológico es como se suele denominar al conjunto de tecnologías que necesitas para toda tu aplicación: desde la base de datos, servidor de aplicaciones, servidor web, clientes, lenguajes de programación involucrados, frameworks y bibliotecas.

Todo esto tiene límites borrosos. Algunos frameworks son para full stack (front-end y back-end), otros son modulares con lo que en la práctica puedes configurarte distintos stacks tecnológicos con ellos, y otros son solo para front-end o solo para back-end. Algunos stacks combinan distintos frameworks y/o bibliotecas. Y no todo termina con la elección del framework, stack o biblioteca, puesto que muchos de estos te ofrecen a su vez distintas alternativas para algunos de sus elementos.

Dicho todo esto, algunas elecciones comunes para los lenguajes de programación sugeridos antes son:

- **MEAN: MongoDB, Express, Angular y Node**. Con este stack puedes desarrollar una aplicación web o web móvil completa con JavaScript/TypeScript.
- Para hacer clientes web, tanto si trabajas con MEAN como con otras opciones, si **Angular** no te convence, la parte cliente puedes hacerla con **React**, **Vue.js**, o con algo más biblioteca y menos framework como **JQuery**. Angular viene con un conjunto de componentes para la GUI, **Angular Material**, mientras que para React lo más popular será seguramente **Material-UI** (aunque hay más opciones en ambos casos). Si trabajas con JQuery querrás usar componentes GUI basados en CSS como **Bootstrap** o **Semantic UI**.
- Para Python, el back-end se puede montar con **Flask** para el servidor Web y **MySQL** o **PostgreSQL** con **SQLAlchemy** por encima para la persistencia. En lugar de **Flask** puedes usar **Django**, que es más grande y anterior a Flask, aunque probablemente escribas menos código con Flask.
- Para Java (y recientemente también para Kotlin) el framework más común es **Spring con Spring Boot**, pero por muy fácil que te pongan crear una aplicación de cero, que lo ponen fácil, tiene muchas cosas y la curva de aprendizaje se puede volver empinada rápidamente si te sales de lo básico. **JHipster** puede facilitarte montar una aplicación básica (modelo de datos que establezcas, y consulta/edición de las entidades de este modelo vía interfaz web simple) con, típicamente pero puedes elegir otros, **Angular/React** en el cliente, **Spring Boot** en el back-end y la BD que prefieras. Eso sí, una vez montada la aplicación básica, el esqueleto, el resto te va a tocar a ti completarlo.
  - Hay alternativas a Spring como como **Javalin** o **Spark** que son bastante ligeras, más sencillas, y te permiten montar la parte web muy fácilmente.
  - Para conectar a la BD se puede usar **JDBC** directamente, o el driver de MongoDB si se va a usar esa BD, o se puede usar alguna implementación de **JPA** como **Spring Data** o **Hibernate** para facilitar la relacción entre objetos y BD[3].
- Si vais a hacer un cliente móvil, en general tenéis que adaptaros a la plataforma: Java, o Kotlin, para Android y Swift para iOS será lo más inmediato para aplicaciones nativas. **React Native**[4] es una alternativa basada en desarrollo con tecnología web pero que en ejecución usa componentes nativos (pero tienen que estar preparados para su uso con React, así que no tendrás acceso a tantas opciones como si trabajas directamente en nativo). **Ionic**[5] facilita el trabajo con tecnología web (sobre Angular) para hacer aplicaciones móviles híbridas, que pueden conectar con algunas de las API de la plataforma móvil que tienes por debajo, aunque siguen siendo aplicaciones web (p.ej. son más lentas que las creadas con React Native). Con Ionic

3 Aunque Spring Data te permite conectar con BD NoSQL como MongoDB, realmente lo más interesante de un mapeador objeto relacional (ORM) es mapear objetos y tablas (y por tanto BD como MySQL o PostgreSQL). Dicho esto, si para el proyecto se usa Spring, puede ser algo más fácil conectar con MongoDB con Spring Data que con el driver de MongoDB directamente.

4 Basado en React, claro.

5 Hay un Ionic Native, pero solo es un nombre para la parte de Ionic que se conecta con las API nativas de la plataforma móvil que hay debajo. Siguen siendo aplicaciones híbridas, no como las de React Native.

es más fácil que una aplicación se pueda crear una vez y ejecutar tanto en Android como en iOS que con React Native (donde habrá partes que querrás implementar de manera distinta), pero parecerá menos nativa y será más lenta. También está **Flutter**, que se programa en Dart y promete prestaciones nativas al desplegar sobre Android e iOS, aunque al parecer su despliegue sobre web no resulta tan eficiente. La mayor pega de Flutter es que tiene su propio ecosistema así que a diferencia de con las alternativas basadas en tecnologías web estándar, lo que aprendas ahí solo te servirá en el futuro si vuelves a usarlo.

- Si queréis hacer una aplicación de escritorio, sí, el escritorio también existe y usáis todos los días aplicaciones de escritorio, podéis darle un vistazo a **CEF** (**Chromium Embedded Framework**, lo que usan Spotify o el cliente de la plataforma de juegos Steam) o a **Electron** (lo que usa Atom o Slack) para desarrollar con tecnologías web pero desplegar como aplicación de escritorio. O podéis apostar por tecnología desktop pura, en cuyo caso podéis mirar las bibliotecas de componentes de GUI **wxWidgets** o **Qt** y programarla con **C++** o **Python**, o incluso usar Java con **JavaFX**.
- Base de datos podéis elegir lo que prefiráis con casi cualquier stack/framework. En el stack MEAN se puede sustituir MongoDB por otra cosa, y desde Python se puede conectar con MongoDB en lugar de con MySQL (aunque, lógicamente, no usando SQLAlchemy). Recordad que la ventaja principal, teórica, de las BD No SQL es la escalabilidad horizontal, algo que no vais a tener que hacer en la asignatura. Y que las BD SQL ofrecen más potencia de consulta y más garantías de integridad y consistencia de datos (pero con lo que ofrecen las NoSQL en eso puede ser más que suficiente para la asignatura).

Tecnologías para despliegue

Trabajar con Docker o VMs os puede facilitar mucho la vida para que todos compartáis las mismas versiones de los componentes en ejecución. P.ej. simplemente con usar una versión dockerizada de la BD, os aseguráis de que todos usáis la misma versión y que no tenéis que instalarla en vuestros equipos (instalar las BD a veces no es trivial) para hacer pruebas. Por supuesto a Docker le podéis sacar más partido que solo eso.

Para tener cosas desplegadas en la nube y que todos podáis usar tenéis los proveedores principales, AWS, Google Cloud y Azure Cloud, con sus precios y sus cuentas de estudiantes etc. (que van variando de año en año, aunque la tendencia es que cada vez es más difícil acceder a los recursos gratuitos que dan sin poner primero tu tarjeta de crédito). También podéis usar el nivel gratuito de Heroku que, aunque bastante limitado, no está mal para ser gratis y puede ser suficiente para vuestro proyecto.

Una opción bastante directa y barata, pero no gratuita, es ir por ejemplo a OVH y contratar el VPS SSD más pequeño. Sale bastante barato, especialmente repartido entre todos los integrantes del equipo, y tendréis acceso por SSH a una máquina con Linux y con su propia IP, en la que podéis instalar lo que queráis. Si estabais pensando en usar la Raspberry PI que alguien tiene en su casa, esto desde luego os dará menos problemas.

En general:

- Podéis usar contenedores Docker, con o sin algo tipo Kubernetes, máquinas virtuales, máquinas propias alojadas en vuestra casa, servidores privados virtuales en un hosting etc.
- Podéis usar soluciones *Infrastructure as a Service* (IaaS) y *Platform as a Service* (PaaS). Tanto las que venden los proveedores de *cloud* generalistas, AWS, Microsoft Azure, Google Cloud etc., como las más específicas tipo Heroku.
- Podéis usar soluciones tipo *Database as a Service*, como ElephantSQL o MongoDB Atlas.
- **No podéis** usar sistemas de tipo *Back-end as a Service* (BaaS) como Firebase o Backendless. Aunque podrían tener sentido para un proyecto similar al que propongáis en un contexto profesional, o tal vez no, este tipo de sistemas os dan resueltos demasiados aspectos que son

interesantes en el contexto de la asignatura; tendríamos que darle una buena parte de vuestra nota a los ingenieros/as que los han diseñado y que los mantienen en funcionamiento.