1.- Uso de Frameworks y ORM.

· ¿Consideras el uso de un framework una ventaja o un inconveniente? Explica los motivos.

A nivel general y en mi opinión, el uso de frameworks suele ser una ventaja mas que un inconveniente pero depende mucho del proyecto.
Me explico: usar un framework te ahorra tiempo y te proporciona una estructura más definida y clara, con una arquitectura ya establecida (como podría ser MVC) y con muchas estructuras, funciones, herramientas y componentes de seguridad ya incorporados que nos pueden ayudar mucho a la hora de desarrollar o escalar nuestra aplicación.
Por el contrario, si vamos a realizar una aplicación simple o con un objetivo muy concreto puede ser que instalar un framework nos introduzca una sobrecarga innecesaria en nuestro proyecto y con muchas dependencias que al final no vamos a utilizar. Añadiendo que, dependiendo del framework, a veces la curva de aprendizaje es un poco más alta y, en muchos casos, el framework puede ser muy 'cerrado' y te da menos flexibilidad a la hora de programar cosas como lo pueda necesitar tu proyecto.

· ¿Qué ORMs conoces y en qué crees que ayudan o dificultan en la programación de aplicaciones? Explica brevemente los motivos que justifican el hecho de usar o no un ORM.

Los ORM que conozco son:
- Entity Framework (.NET)
- Eloquent (Laravel)
- Hibernate (Java)

Creo que un ORM nos ayuda a la hora de comunicarnos con la base de datos de manera más rápida, limpia y protegiéndonos desde un inicio de errores o de ataques de inyecciones sql.
Además de ello nos ayuda a mantener un código limpio y es más fácil de leer al estar más ordenado y en un lenguaje más claro. Sin embargo, creo que cuando tenemos que tratar grandes cantidades de datos o necesitamos realizar consultas a la base de datos complejas no es de tanta utilidad así como puede ser menos eficiente.
Como en la mayoría de herramientas que tenemos hoy en día a nuestro alcance como programadores lo más importante es saber cuándo es mejor utilizarlo o realizarlo tu mismo.

2.- Integración con APIs Externas.

· ¿Cómo plantearías una nueva integración con una API externa?

Lo primero sería ver y estudiar la API externa, que endpoints tiene, que autenticación pide, que datos hay que mandar en las peticiones, si usa formato JSON o XML, así como ver que datos devuelve y que necesitamos en nuestra aplicación. Una API externa puede ser muy grande y puede llevarnos mucho tiempo de desarrollo integrarla en nuestro sistema cuando igual solo necesitamos integrar una pequeña parte de la misma.
Una vez claro lo que queremos y lo que nos proporciona la API externa prepararíamos la parte interna nuestra, que necesitamos, donde y cuándo vamos a necesitar llamar a la API, si vamos a necesitar guardar y/o mostrar los datos que recibimos, en que formato vamos a necesitar mostrarlos, si vamos a necesitar un service en medio…
En general las API's, cuando he creado alguna, las creo en una clase propia para ello como SMSOrangeService.php.

Una vez todo contemplado, ponernos a trabajar en la integración, el control y manejo de errores, la documentación, pruebas, etc..

· ¿Qué tipo de arquitectura, autenticación y otros elementos de seguridad tendrías en cuenta?

Separaría el código por capas, un controlador para recibir las solicitudes, un servicio que será el encargado de comunicarse con la API externa, un Modelo para dar formato a los datos recibidos y una clase para guardar los datos en la base de datos (en caso de que no tengamos una previa).
La autenticación dependerá un poco de la API externa pero seguramente usaría API Key guardándola en el .env o Oauth guardando el token en la base de datos para también poder controlar su expiración.
En cuanto a seguridad, siempre validar los datos que mandamos a la API externa así como hacerlo también con los datos que esta nos devuelve para evitar datos corruptos, maliciosos, errores…
Establecer un limite de llamadas para evitar abusos o colapsar el sistema.
Tener un buen control de errores para cuando la API devuelva 4/5XX o respuestas vacías poder mostrarle al usuario un error controlado.
Logs, añadir logs tanto al enviar como al recibir para poder controlar el funcionamiento de la aplicación.
Entre otros como usar HTTPS, establecer tiempo máximo de espera de petición o máximo de intentos tras un error o incluso diversas alternativas en el código para cuando falle poder realizar la tarea por otros medios.

3.- Creación de API.

· ¿De qué forma empezarías el desarrollo de una nueva API?

Al igual que cuando creamos un conector para una API externa, es importante antes de empezar una nueva API saber cuál va a ser su finalidad, que tipos de datos debe de manejar, quien la va a usar: si es para terceros, si es para sistemas internos nuestros, entre otros.
Una vez tenemos la información básica de la API empezaremos a listar los endpoints que necesitaremos y aquí también podríamos decidir que tipo de API queremos y el formato de los datos.
En todo momento nos ayudaremos de herramientas de testeo como pueden ser Postman o Swagger.
Empezaremos desarrollando la API por endpoints sencillos y conectar nuestra API a la base de datos y como va a interactuar con la misma, establecer la autenticación que solicitaremos y validar la entrada de datos.
Empezaremos la documentación y control de versiones para poder luego saber como funciona nuestra API y sus características y poder también tener una documentación para proporcionársela a los que van a consumirla.
A largo plazo completaríamos el desarrollo con diversos endpoints de mayor o menor complejidad, las conexiones a las bases de datos, los mensajes y códigos que debe devolver en caso de error o funcionamiento correcto, logs, formato de devolución de datos, etc..

· ¿De qué forma trabajarías la implementación de la misma considerando que hay parte de frontend y de backend?

Una vez hayamos definido y desarrollado la base de la API, empezaría por organizar a ambos equipos y establecer comunicación entre ellos. Es muy importante que tengamos claros los datos que necesita el frontend, en qué momento los van a solicitar y con qué estructura esperan recibirlos para así luego poder definir los endpoints del backend de la forma más eficiente posible.

Si se trata del caso en el que ambos, frontend y backend, necesitan hacer uso de la misma API y esperan recibir los datos de manera diferente y no conseguimos que se pongan de acuerdo habría que crear distintos endpoints para cada uno de ellos.

Lo mejor siempre es la comunicación, tanto con un equipo como con el otro para no desarrollar código que luego haya que cambiar o desechar porque no hemos entendido correctamente las necesidades de cada uno.

4.- Arquitectura y patrones.

· ¿Qué tipo de arquitecturas conoces?

Hay diversos tipos de arquitecturas, solamente he trabajado o, al menos visto, las siguientes:
- Arquitectura monolítica.
- Arquitectura en capas.
- Arquitectura basada en eventos (Event-Driven).

Luego hay otras arquitecturas como Microservicios, Hexagonal, Serverless, etc.. pero no he tenido el placer de trabajar con ellas.

· ¿Para qué usarías cada una de las arquitecturas?

La arquitectura monolítica la usaría para aplicaciones que busquemos poder desplegarlas rápidamente o casi desde el inicio de su desarrollo. El problema principal que tienen es que son difíciles de escalar si el proyecto se acaba haciendo muy grande y que un cambio en una de las partes puede afectar a toda la aplicación sin querer. Nos podría valer por ejemplo para desarrollar un portafolio.
La arquitectura en capas separa el código de manera que todo va a estar mas organizado y va a ser más fácil mantenerlo. La usaría para aplicaciones que sepamos que probablemente vayan a escalar o que van a necesitar de más mantenimiento como podría ser una tienda online.
La arquitectura basada en eventos nos puede venir muy bien para sistemas que requieran mucha comunicación interna o ser reactivas a las acciones del usuario. Como podría ser un sistema de pedidos de una empresa donde el pedido de un cliente activaría a logística, facturación, inventario, etc..

· En cuanto a patrones, ¿puedes enumerar patrones de diseño que conozcas y que hayas trabajado y casos de uso en los que se deben usar?

No no podría asegurar a ciencia cierta si he trabajado con mas o menos patrones de diseño. Probablemente lo haya hecho sin ‘darme cuenta’ o sin ser totalmente consciente de ello. Si que podría indicar como trabajados los patrones de MVC y Factory. MVC no se si se considera patrón como tal o si esta más considerado una arquitectura base como tal. Me gusta mucho usar MVC para las aplicaciones porque la forma de organizar la aplicación me resulta cómoda sabiendo en todo momento donde debo ir para introducir los siguientes cambios.
Factory lo he utilizado en alguna ocasión para crear notificaciones y poder hacer uso de ellas cuando fuese necesario o poder ampliar a nuevas notificaciones en función de las especificaciones.