Technical Specification

System Architecture

The proposed solution should consist of the following elements

Client Frontend

A single page web app designed for the users to interact with the booking system and any other future elements planned for the application. Consideration for a user login area along with public accessible section of this site would need to be made.

For this part of the application we may need to consider how traffic is to dealt with over a wide geographic area depending on where traffic is expected to come from.

Admin Frontend

A single page web application for the client to interact with. This system would be secured away from public access either via secure logon or ideally only accessible with internal network. This application would allow the client to view all bookings made by users on the public site, allowing for the client to make amendments where necessary. This admin interface would also be an ideal place for booking notifications to land.

Backend API

The API will act as a data layer for both the public frontend and the admin frontend, with modules for functions such as booking, users and authentication. The API should be REST compliant and use suitable authentication protocols to ensure that all communication between the frontends are secure.

Database

A relational database connected to the API that ensures efficient and secure storage of all data used within the system.

A secondary database may also be required to provide reporting functionality for the client admin system to ensure that excessive load is not placed on the main production system when provided BI data.

Monitoring

Monitoring system(s) should be integrated into the stack to ensure the smooth running of the application. This should cover logging, exception reporting and hardware/software monitoring so proactive management of performance, bugs and security risks can be done.

Hosting

Section of a cloud vendor to run the application from. The vendor must offer services suitable for all aspects of the application and offer solutions for high availability or redundancy for all aspects of the application to avoid downtime or performance bottlenecks.

Backup

We must have a suitable backup system in pace to cover the database and any additional static content. Backups must run successfully on an agreed basis and cover an agreed amount of history. We must also have in place the ability to restore backups for a specific date and time to the main production system or an agreed staging/testing location.

Staging

Staging versions of all systems should be provided to represent the current live system or development builds of future system improvements. These systems should be secured away and made accessible to only the development team and the client.

Technology Stack

Laravel is to be used as the framework for the main application API. With the main Backend language skill being PHP, Laravel is the best framework choice for this for the following reasons:

- Popularity within the PHP community. As the most used PHP framework most PHP developers will have used Laravel which is great if additional developer resource is required either in-house or via a 3rd party.
- The base documentation provided by Laravel is excellent and due to it's popularity that are plenty of other 3rd party resources and tutorials covering the framework.
- 3rd party package support. Again as the most popular framework there is plenty of support for additional packages to supplement the framework. This includes code quality tools such as specific versions of PHPCodeSniffer and PHPStan.
- Regular security and version releases supporting the new features released for PHP.
- Eloquent makes interacting with SQL easier and gives us version control of schema changes.

Database

MariaDB is supported out of the box with Laravel and fits the requirements for a relational database and is fully compatible with Eloquent. Cloud service are available to run MariaDB for production and cover the requirements for high availability, backups and data snapshots.

Frontend

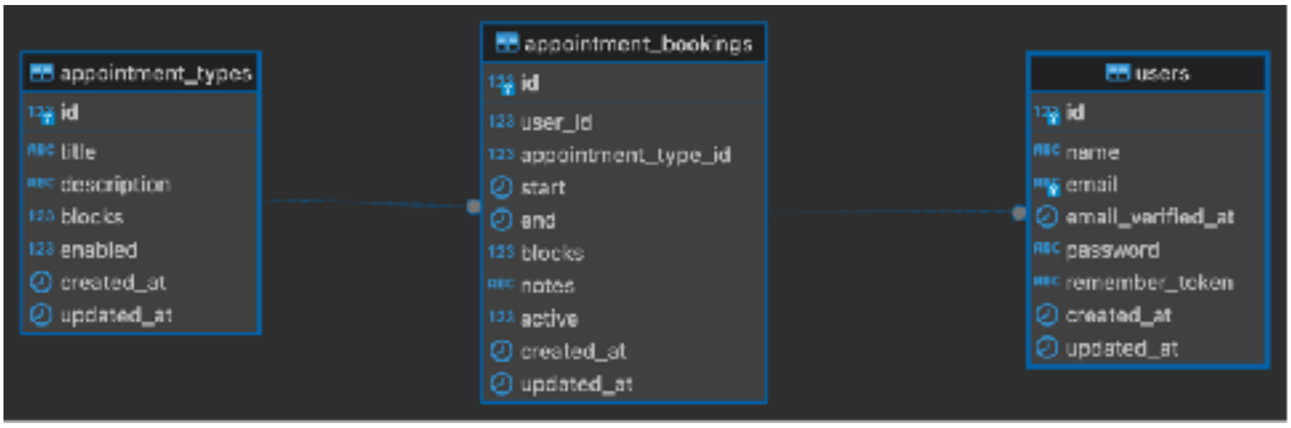
Vite + Vue.js + Tailwind

Popular Javascript and CSS frameworks to accelerate development of single page web apps for the client and admin interfaces.

Monitoring

Recommend Elastic for log and hardware/cloud system monitoring and Sentry for application logging. Both systems allow for alerts that could easily be converted into support tickets for bug fixes and DevOps tasks.

Database Schema



The above diagram represents the limited tables used in the code example for this test.

Security Considerations

The following actions should be taken to secure the application across the stack:

- Ensure that TLS encryption is used at all times when interacting with the frontend web applications and the REST API.

- Ensure that all software is kept up to date wherever possible at all times. This will included the OS running the applications, the frameworks involved in building the applications and database software.

- All data on disk should be encrypted.

- Sensitive data such as passwords should never be stored in plain text within the database.

- Firewall policies should be in place to allow only required ports/traffic across all servers and, where possible, IP addresses should be white-listed for access with all access blocked by default unless required in the case of communication between API and database, for example.

- Any keys or passwords used within the system should be rotated on an agreed basis.

- API access should be secured with API keys. The API keys should only allow the minimum level of access required.

- The scope of the API endpoints should be kept as narrow as possible.

- Have suitable alerting in place to identify potential attacks on the system and automatically blacklist suspicious IP addresses.

- Implement OAuth2 for user access within the frontend and prefer to use SSO if possible to prevent the need to store user passwords and other sensitive data.

- Sensitive data should never be used within a URL structure.

- Ensure code is written on the API layer to prevent SQL injection.

- Enable your code repository to constantly monitor dependancies for security vulnerabilities and automatically patch or recommend alternatives if a dependancy is abandoned.

- Have a well-tested disaster recovery process and ensure that security audits are performed regularly.

Performance Considerations

Ideally there should be no single point of failure across the stack. The database layer should be clustered or at least suitable replication or failover across multiple locations or data centres.

For the API and frontend applications we should consider using Kubernetes to auto scale these parts of the stack. Autoscaling can allow for peak usage time or heavy load and also give basic failover for these parts of the stack.

Other basic performance considerations are to ensure that the databases are optimised with all relevant indexes applied and any slow queries identified so that future optimisation can be done.

Any reporting or BI requirements should be shifted onto a separate database that is not running the main application.

Active monitoring and alerting should be applied to all parts of the stack highlighting any issues with excessive resource usage. It would also be useful to implement AI tools to analyse logs to help identify any peak or unusual activity so this can be fixed.