1. Showing code in action
   1. Attempt to login as unverified@domain.com
   2. Attempt to access /dashboard route without being logged in
   3. User registration with 2fa (showing validation)
   4. Login as newly registered user (showing validation)
   5. Attempt to access /users route as standard user
   6. Create a ticket (showing validation)
   7. Logout
   8. Login as [admin@domain.com](mailto:admin@domain.com)
   9. Mark a ticket as deleted
   10. Close a ticket
   11. Restore ticket
   12. Open ticket
   13. View ticket
   14. Leave a comment on ticket (showing validation)
   15. View users
   16. Lock a user
   17. Unlock a user
   18. Delete a user
   19. Restore a user
   20. Create a new support user (showing validation)
   21. Open the system in browserstack and navigate the system to show all pages are responsive
2. Run unit tests
   1. Run the postman tests
      1. Explain what each test does and why it is being tested
   2. Run React tests
      1. Open the test files to show what is being tested and explain why those tests are being run
3. URLs
   1. Go through API routes and explain they’re restful
   2. How are different HTTP verbs used
   3. Go back to postman and show the data returned from each request to the API
   4. How are requests parsed – the body-parser middleware module parses incoming requests before being passed to handlers.
   5. How are requests handed off to other code – requests are passed from the route to middleware and then to the controller. In the middleware, the request may be modified, or authentication may occur before the request is passed onto the controller’s method
4. Data structures
   1. Show the schemas for each model and explain each property’s purpose and data type
   2. Show an example document from the mongodb database for each model
   3. Show how the data access layer is implemented and explain the purpose of the data access layer – show where the connection is made (before the server starts listening for requests). Show examples in controllers where the database is being interacted with. Using the methods that the data access layer provides, removes the need to write common commands / stored procedures. The data access layer returns instances of models with relationships (where applicable) and makes querying simpler.
   4. Show requests where single and objects and sets of objects are returned - read all tickets and read single ticket. Read single ticket populates relationship of author (user model) and comments (comment) within the request’s response.
   5. Are the data structures and documents extendible?
5. Architecture
   1. Explain implementation of call backs and events – use promises to wait for a response from asynchronous requests before executing code in call back. Await is also used to stop execution of program until asynchronous request’s response has been received. I use the component life cycle events to execute code at specific times.
   2. Explain how the system’s folder structures create a coherent API. Discuss clear split between route declaration, controllers, and models.
   3. How might the framework be extended?
   4. What design or architectural patterns have been used – I have used the module design pattern to result in loose coupling and well-structured code. Show how all components are defined in separated files. I use the observer design pattern to create reactive functionality – when state is updated, dependents of that state are also updated. MVC architectural pattern has been used – this created a clear order the request goes through the system. MVC allows for loose coupling and an easier system to navigate / maintain.
6. Application
   1. Explain the user interface and user experience
   2. Explain how user accounts are managed via the user interface. Explain that when models in the system are deleted, they are only marked as deleted and not dropped from the database. This should run alongside an automated script to drop models from database that have been marked as deleted for a month.
   3. How authentication is achieved – login via user interface where credentials are compared against the hashed and salted password in the database using bcrypt. A jwt token is generated for the user and then the jwt token is returned to the frontend in the response as a http only cookie. User accounts must be verified which is achieved via SMS 2FA. If a password is entered incorrectly 5 times when logging in, the account is locked – the account can only be unlocked by an admin. A login session is created in local storage on the client side which is only used for frontend route authentication.