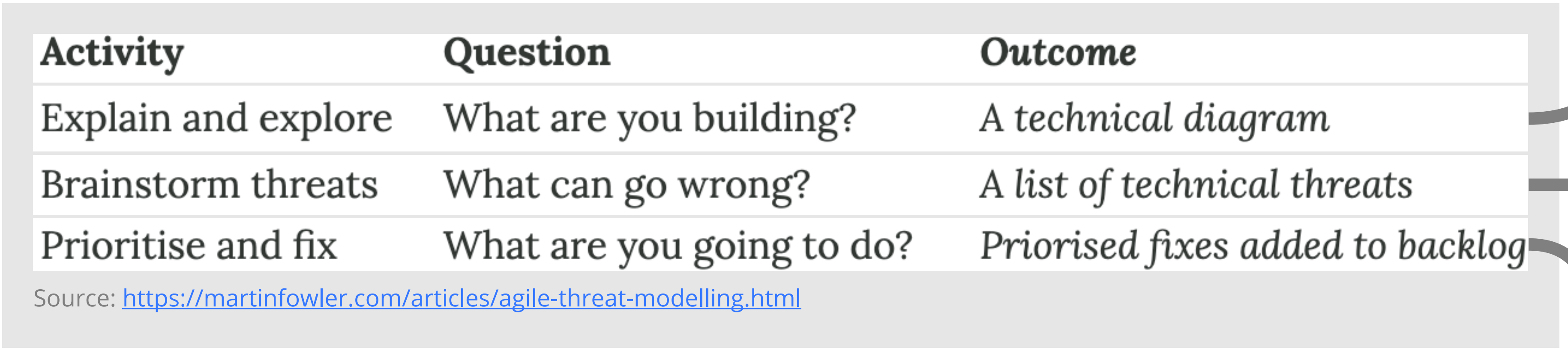
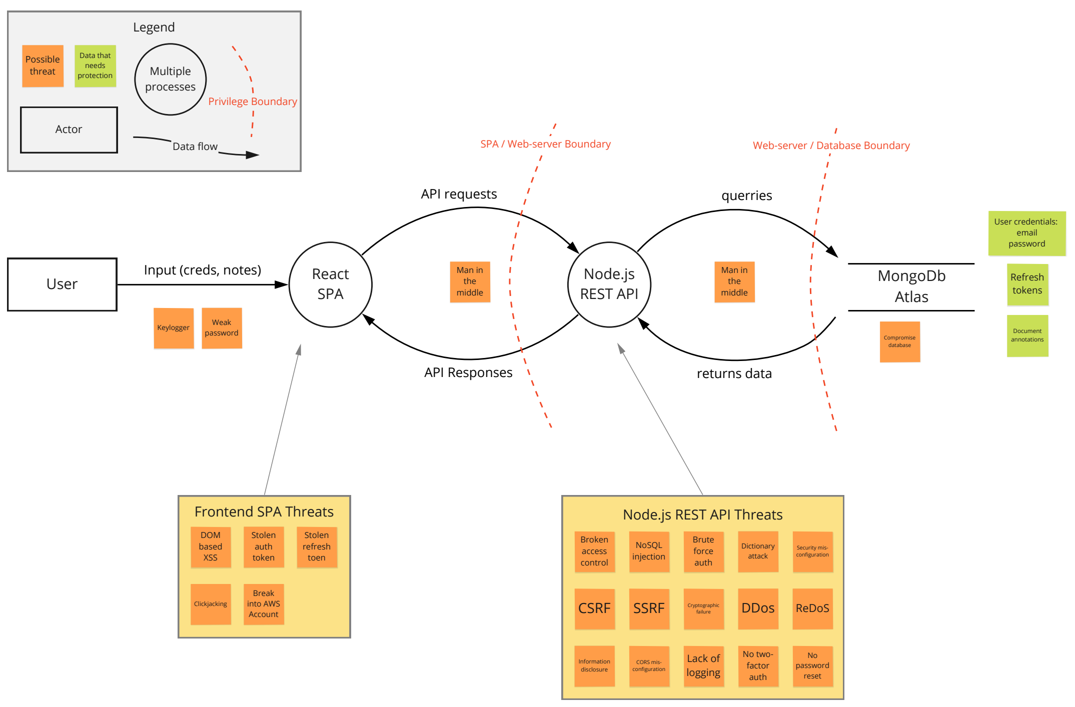


Activity	Question	Outcome
Explain and explore	What are you building?	*A technical diagram*
Brainstorm threats	What can go wrong?	*A list of technical threats*
Prioritise and fix	What are you going to do?	*Priorised fixes added to backlog*

Source: https://martinfowler.com/articles/agile-threat-modelling.html



Data Flow	Threat
User -> React SPA	• Weak password • DOM based XSS • Clickjacking • No use of https • Password based auth (no two-factor) • No password reset • Keylogger
React SPA -> Node.js REST API	• (Broken) Access control / Stolen access- or refresh-token • Man in the middle • NoSQL injection • CSRF • Brute force / Dictionary attack /login • Stored XSS • Cryptographic failures • CORS misconfiguration • Information disclosure (harvest valid user accounts) • Break into AWS account • DDos • ReDoS • Lack of logging • Security misconfiguration
Node.js REST API -> MongoDb Atlas	• Man in the middle • Compromise database

User -> React SPA	Weak password	Password strength check (Frontend + API)					
	DOM based XSS	Use React's default dynamic content escaping	Set Content-Type and X-Content-Type-Options	Don't use dangerouslySetInnerHTML	Mitigate XSS with strict CSP		
	Clickjacking	CSP frame-ancestors: none	X-Frame-Options: deny				
	Only use https	Use Strict-Transport-Security	Redirect http to https				
React SPA -> Node.js REST API	(Broken) Access control / Stolen access- or refresh-token	Reject request as unauthorised if no valid access token	Short lived access token + refresh tokens	Refresh token rotation	Automatic refresh token reuse detection	Use cookies instead of local storage to store jwt	'secure', 'sameSite' and 'httpOnly' cookies
	Man in the middle	Encrypt requests via TLS	Properly configure TLS	Only allow secure ciphers and protocols			
	NoSQL injection	Validate + sanitise input	mongoSanitize middleware				
	CSRF	'sameSite' cookies	CSRF tokens				
	Brute force / Dictionary attack /login	Password hashing + salting with Argon 2	Configure parameters for argon2 to be "slow" enough				
	Stored XSS	Validate and sanitise input	Escape dynamic content in frontend				
	Cryptographic failures	Use strong key derivation function	Finetune parameters				
	CORS misconfiguration	Check best practices and check config	Only allow required methods, headers and origins				
	Information disclosure (harvest valid user accounts)	Return default error (500 Something went wrong)	Only detailed errors which the frontend can handle	Don't disclose if an account with a certain email exists			
	Break into AWS account	Use strong generated password	Use strong two factor auth (yubikey)				
Node.js REST API -> MongoDb Atlas	Man in the middle	Only allow encrypted connections					
	Compromise database	Password hashing + salting	Encrypt database	Canary accounts			