7. Conclusion

Project Outcomes Our project successfully addresses all course objectives through the implementation of a secure, scalable, and fully functional full-stack application.

We developed the frontend using React with TypeScript, leveraging its modular, component-based architecture to build reusable UI components and manage application state. For the backend, we utilized Quarkus, a container-first Java framework, to create a REST API that handles business logic, user authentication, and persistent storage. Quarkus provided us with built-in dependency injection and Jakarta validation, streamlining development.

Application state was managed effectively on both the frontend and backend. On the frontend, we used the React Context API to maintain global state, such as user authentication and game results, without unnecessary prop drilling. On the backend, the PostgreSQL database served as the single source of truth, persisting user and game data.

To secure the backend, we implemented JWT authentication with role-based access control. Users receive short-lived access tokens and long-lived refresh tokens, ensuring secure access to protected endpoints while minimizing the need to re-enter credentials. This approach effectively supports role-based restrictions, such as limiting game endpoints to authenticated users.

In tackling the complexity of a gambling platform, we applied multiple frameworks, including Quarkus for backend development and Siren4J for implementing hypermedia-driven API responses. These tools allowed us to structure the backend effectively and dynamically guide clients through API interactions without hardcoding paths.

We used Maven to manage backend dependencies and npm for the frontend, streamlining development with tools for version management and package installations. For testing and deployment, we integrated GitHub Actions to automate CI/CD pipelines. This ensured that every code push was built, tested, and deployed automatically, reducing manual effort and maintaining high-quality code.

The backend was containerized using Docker, enabling consistent deployments across environments. The frontend was deployed on Netlify, leveraging its free tier for hosting and automated builds. These platforms facilitated a seamless deployment process and ensured scalability.

By integrating all these elements, we designed and implemented a fully functional full-stack application, meeting the course objectives and demonstrating a practical application of the knowledge gained throughout the project.

Future Improvements Future improvements should include the addition of more games and extensive frontend testing, including endpoint testing. The De-

vOps workflow should be maintained, integrating development and operations seamlessly. This approach ensures continuous integration and continuous delivery (CI/CD), allowing for rapid deployment of new features and quick resolution of issues. By fostering collaboration between development and operations teams, we can achieve higher efficiency, reliability, and scalability in our software development process. In summary, we have established a solid foundation for continuing this workflow with our existing development environment - however our code infrastructure could use some work.

Lessons Learned During the project we have learned alot about DevOps practices and collaboration. One of the biggest things we have learned is how to use pull requests efficiently in order to review and collaborate on source code. We've also learned how to use GitHub and GitHub Actions to set up build and testing pipelines that ensure the solution being pushed is able to run and pass the tests that we have written to the project, thus applying the best practices of CI/CD development. We have also learned how to set up a full stack application using two different repositories that work together when ran simultaneously. Other than that we have learned how to implement and use token based security in order to add authorization layers to our solutions in order to protect endpoints.