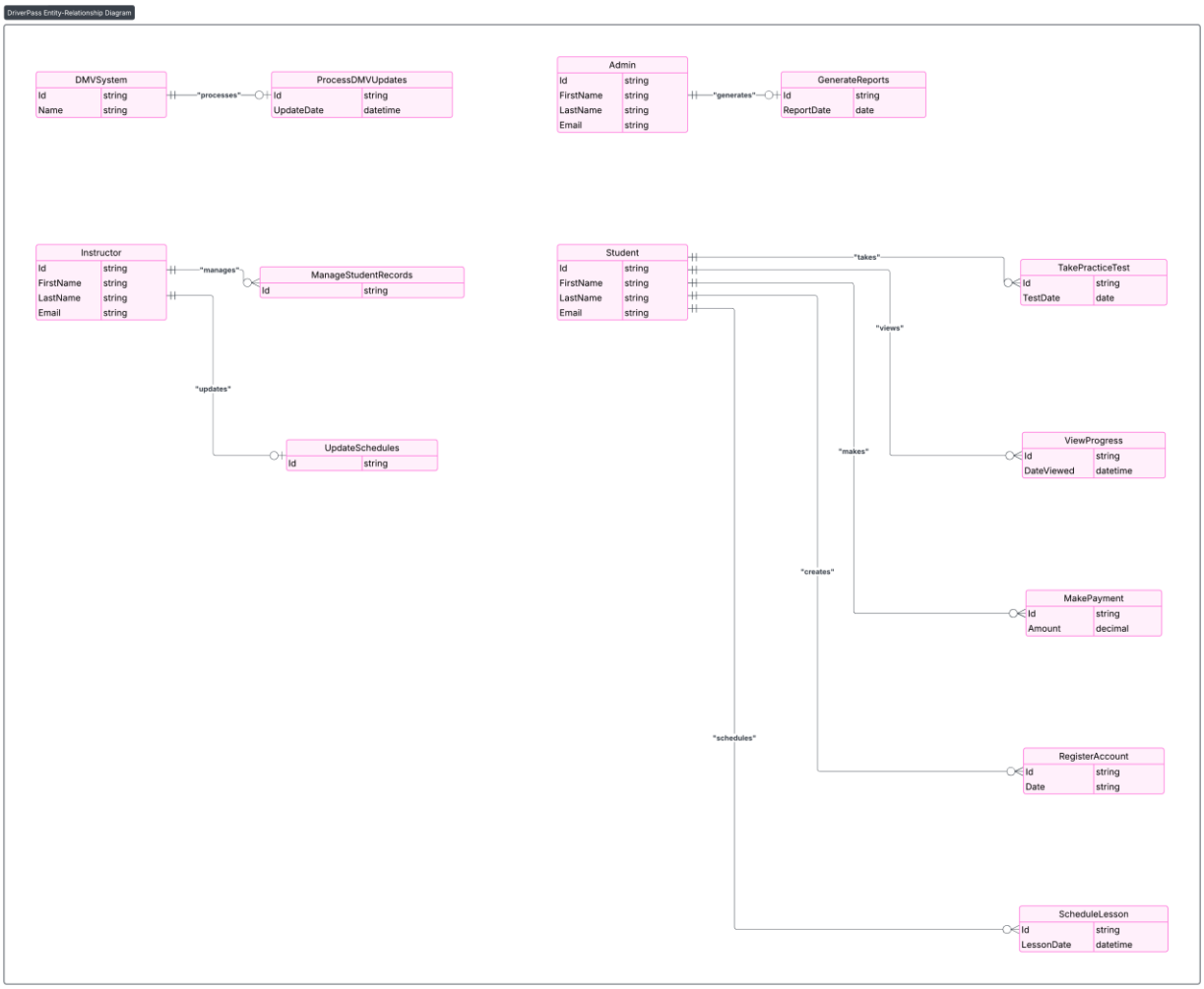
Southern New Hampshire University

CS 255 System Design Document Template

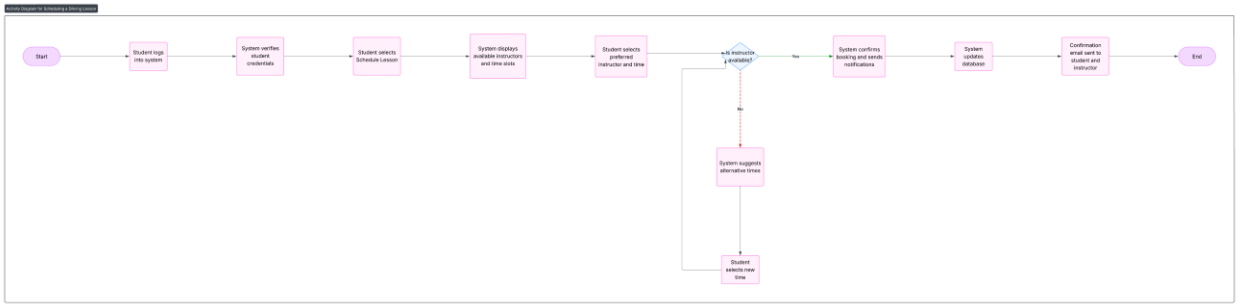
UML Diagrams

UML Use Case Diagram

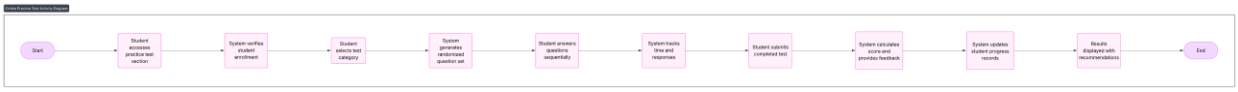


This use case diagram illustrates the key interactions between different user types and the DriverPass system. The diagram shows how students, instructors, administrators, and the DMV system each have specific functions they can perform. Students focus on learning activities like taking tests and scheduling lessons, while instructors manage their teaching responsibilities. Administrators oversee the entire system, and DMV integration ensures current information. This design directly addresses DriverPass's need to serve multiple user types efficiently while maintaining clear separation of responsibilities.

UML Activity Diagrams

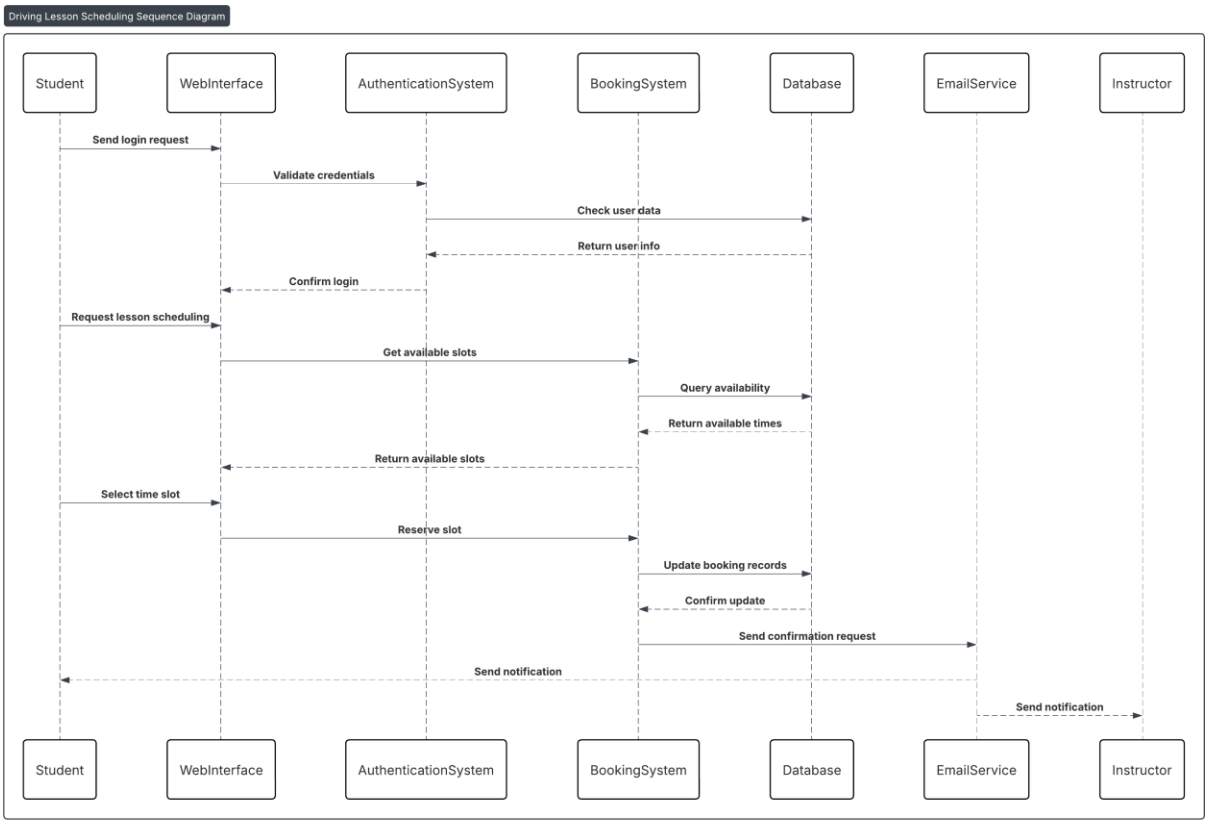


This activity diagram demonstrates the step-by-step process students follow to schedule driving lessons. The diagram shows the logical flow from login through confirmation, including decision points for instructor availability. This automated process eliminates the need for phone calls and manual scheduling, directly addressing DriverPass's need for efficient lesson management and improved customer experience.



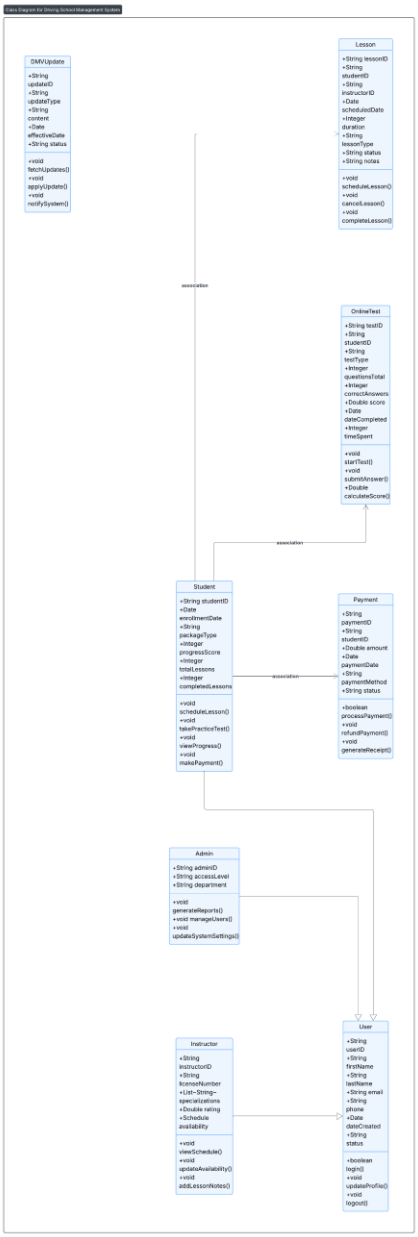
This activity diagram outlines how students take online practice tests within the system. The process flows from test selection through results and feedback, ensuring students receive immediate scoring and progress tracking. This addresses DriverPass's core need to provide comprehensive test preparation that goes beyond simple memorization of previous exams.

UML Sequence Diagram



This sequence diagram details the technical interactions between system components during lesson scheduling. It shows how the user interface, authentication, booking system, database, and email service work together to complete a booking. This design ensures reliable, secure, and efficient processing of student requests while maintaining data integrity and proper notifications.

UML Class Diagram



This class diagram represents the data structure and relationships within the DriverPass system. It shows how users, lessons, tests, and payments are organized and connected. The inheritance structure (Student, Instructor, Admin extending User) provides efficient code organization while the associations between classes support all required business functions like scheduling, testing, and payment processing.

Technical Requirements

Based on the UML diagrams I've designed for the DriverPass system, here are the technical requirements I've identified. While I'm still relatively new to the field, I've been working on production systems at my current job, so I'm drawing from both my coursework and real-world experience.

Infrastructure and Hardware Requirements

From what I've learned working with our production environment, we'll need solid server infrastructure to handle concurrent users effectively. I'd recommend starting with a web server running at least 8GB RAM and a dual-core processor with SSD storage. Based on similar applications I've worked on, 500GB should be sufficient initially, but we'll want to monitor usage patterns closely.

For the database server, I'd go with 16GB RAM and a quad-core processor. Having dealt with performance issues on our current project when the database became a bottleneck, I know this is where you don't want to cut corners. RAID configuration is essential - I learned this the hard way when we had a drive failure last month and had to restore from backups.

The load balancer is critical for handling peak usage. During my internship, I saw what happens when registration systems get overwhelmed during high-traffic periods. For DriverPass, this will probably be exam seasons and back-to-school periods.

Development Stack and Architecture

I'm comfortable with Node.js for the backend since we've been using it extensively at work, and the async nature handles concurrent requests well. The alternative would be Java Spring Boot, which I've used in school projects, but Node.js feels more natural for this type of application.

For the frontend, React is definitely the way to go. I've been working with it for about eight months now, and the component architecture will work perfectly for the different user interfaces we need - students, instructors, and admins all have different workflows. I'll pair it with Material-UI since I've gotten pretty efficient with their component library.

Database-wise, I'm leaning toward PostgreSQL over MySQL. We switched to Postgres at work last year, and the data integrity features are noticeably better, especially for financial transactions. Given that we're handling payment data and student records, those constraints and ACID compliance features are worth the slightly steeper learning curve.

Security and Payment Processing

Security is obviously critical here. I've implemented SSL/TLS in production before, and it's non-negotiable. For authentication, I'd go with JWT tokens combined with OAuth 2.0 - I just finished implementing this pattern at work and it provides good security without being overly complex for users.

For payments, Stripe is the clear choice. I integrated their API on our last project, and their documentation is excellent. Plus, they handle PCI compliance, which removes a massive regulatory

burden from our side. The last thing we want is to deal with payment card industry audits as a small team.

Third-Party Integrations

Email notifications will use SendGrid - I've integrated it before and their API is straightforward. For SMS, Twilio is the standard choice. I haven't worked with their API yet, but from what I've researched, it's well-documented and reliable.

Google Maps integration for driving routes could be a huge differentiator. I haven't implemented Maps API specifically, but I've worked with other Google APIs and they're generally well-designed. This would let instructors plan optimal routes and give students a better sense of what to expect.

Development Workflow and DevOps

Git with GitHub for version control, obviously. I'd set up CI/CD pipelines from the start - we didn't do this on my first project at work and it was painful trying to add later. Docker containers will make deployment consistent across environments, which I've learned is crucial for avoiding "it works on my machine" problems.

For monitoring, I'd start with basic application logging and add more sophisticated tools as we scale. At work, we use New Relic for performance monitoring, and it's been invaluable for identifying bottlenecks. Error tracking with Sentry has also saved me hours of debugging time.

Database Design and Performance

Looking at our class relationships, proper indexing will be crucial. I'd index on frequently queried fields like student IDs, lesson dates, and instructor availability. From experience, query performance becomes noticeable to users pretty quickly, especially for scheduling features where they expect instant results.

Backup strategy needs to be rock-solid from day one. We had a scare at work where our backup process wasn't working correctly, and it was a wake-up call about how critical this is for any system handling user data. Daily automated backups with geographic redundancy aren't optional.

Cloud Deployment and Scaling

I'd deploy on AWS since that's what we use at work and I'm familiar with their services. EC2 for compute, RDS for managed databases, and S3 for file storage. Their auto-scaling capabilities will handle traffic spikes during peak periods without requiring manual intervention.

Load testing should be part of our development process. I've seen what happens when systems get overwhelmed during high-traffic periods - user experience degrades quickly. Tools like Artillery or JMeter can simulate concurrent users taking tests and booking lessons.

DMV Integration Challenges

The DMV integration is probably the most complex technical requirement. Different states have different systems and data formats. I'd design this as a separate microservice with adapters for different state systems. This will make it easier to add new states and maintain existing integrations as DMV systems change.

Performance Optimization

Response time is critical for user experience. Students taking practice tests need immediate feedback, and lesson scheduling should feel instantaneous. I'd target sub-2-second page loads and sub-500ms API responses for most operations. Redis caching will help achieve this for frequently accessed data like test questions and instructor schedules.

Progressive Web App features would let students download practice tests for offline study. I implemented PWA features on a recent project, and while there are some complexities around data synchronization, the user experience benefits are significant.

This technical foundation should support DriverPass's requirements while providing flexibility for future enhancements. The key is building something robust enough for production use while keeping the architecture simple enough for our team to maintain and extend.

Assumptions

The following assumptions were made during the design of the DriverPass system:

From my experience working on similar projects, I'm assuming that students will have access to reliable internet connections for online practice tests and lesson scheduling. Most students today have smartphones and home internet, but this could be a limitation for some users in rural areas.

I'm also assuming that DriverPass will provide initial training to staff on system administration and management. At my current job, we learned that user adoption is much smoother when there's proper training upfront rather than trying to figure things out as you go.

For instructors, I'm assuming they're comfortable using basic computer and mobile device interfaces. Most driving instructors I've encountered seem tech-savvy enough for scheduling and basic system navigation, though we might need to provide some additional support for less tech-comfortable users.

I'm assuming students possess basic computer literacy skills for navigating web-based applications. Given that most driving students are teenagers or young adults, this seems like a safe assumption, but we should design the interface to be intuitive even for less experienced users.

For ongoing system management, I'm assuming DriverPass management will designate specific system administrators rather than having everyone try to manage different parts. This is crucial for maintaining data integrity and security based on what I've seen at work.

From a technical standpoint, I'm assuming DMV APIs and data formats will remain relatively stable for integration purposes. This is probably the riskiest assumption since government systems can change unpredictably, so we'll need to build flexible interfaces that can adapt to changes.

For payments, I'm assuming we'll need to comply with current PCI DSS standards. Having worked with payment systems before, I know this isn't optional and affects how we architect the entire payment flow.

I'm also assuming the system will initially serve the local market before potential expansion to other regions. This affects how we design the database and user management systems - we want scalability but don't need to over-engineer for national scale immediately.

Limitations

The following limitations should be considered for the DriverPass system implementation:

The biggest limitation I see is that system performance depends entirely on internet connectivity quality for all users. Students taking practice tests, instructors checking schedules, and admin staff managing the system all need consistent internet access. This could be problematic during internet outages or for users with unreliable connections.

Mobile application features will likely be limited compared to web browser functionality. From my experience developing responsive applications, some complex features just work better on larger screens. We'll prioritize the most important mobile features like basic scheduling and practice tests, but advanced reporting might require desktop access.

DMV integration will vary significantly by state and could require custom development for each region DriverPass wants to serve. Different states have completely different systems and data formats. I've worked with government APIs before, and they're often inconsistent and poorly documented, so this will probably be our biggest ongoing technical challenge.

Advanced reporting features may require additional development time beyond initial deployment. While we're including basic progress tracking and scheduling reports, complex analytics or custom reporting formats would need to be added later based on specific business needs.

System capacity will need monitoring and potential scaling during peak enrollment periods. Based on what I've seen with other seasonal businesses, driving schools probably have major traffic spikes during summer months and before major holidays when students want to get licensed. We'll need to plan for these load increases.

Offline functionality is limited to basic practice test features only. While we can cache some test questions for offline study, scheduling and payment processing require internet connectivity. This is a limitation of how online systems work, but it's worth noting for users who want to study in areas with poor coverage.

Third-party service dependencies could impact system availability. We're relying on Stripe for payments, SendGrid for emails, and potentially Twilio for SMS notifications. If any of these services have outages, it affects our system functionality. This is pretty standard for modern applications, but it's a limitation of not building everything in-house.

Initial data migration from existing systems may require temporary dual-system operation. If DriverPass currently uses spreadsheets or another system for student tracking, we'll need time to transfer that data and train staff on the new system. During this transition period, they might need to maintain both systems until everyone is comfortable with the new one.