**Justification for Candidate Architecture**

In our game AdventureLearn we decided to use 3-tier architecture as it provides many benefits for production, maintenance, security and gives great flexibility for developers. Considering rapidly changing technological environment this kind of approach will be suitable for our learning platform.

As separate modules can be distinguished:

1. **Presentation Layer**

Top layer is the closest one to the user. This component enables interaction between player and the system (and other players through it).

2. **Business Layer**

Middle layer can be also called control layer as it takes care of controlling different parts of the software. It consists of a whole logic behind the presentation layer which allow competition between players and student-teacher interaction. Business layer plays the connector role for 1st and 3rd tiers.

3. **Data access layer.**

Low-level layer is responsible for database management system and application security. Within Data layer all the objects and data needed for higher level tiers are stored.

This way of separating application modules is adequate for the game because it makes the overall structure clear and the application itself easy to maintain. Keeping in mind SOLID principles:

- 3-tier architecture make the product ready for future development and different kinds of extensions, improvements

- components are dependent just on certain elements what increases security of AdventureLearn game (User-layer has no direct access to database what can be considered as security measure and step towards efficiency with data processing)

**Other benefits from using tier-architecture for our app:**

* Gives the ability to update certain elements without interfering with other components (e.g. UI improvements will not change app logic)
* Provides scalability for the game (e.g. in further development the same logic and database can be used to deploy the game to the other platforms; on top of that other database interface easily can be added so data storage is not limited to the one instance anymore)
* Reliability is ensured due to proper dependencies usage.
* Maintenance becomes easy as it can be divided between front-end and back-end developers.

**Why not 2-tier architecture?**

Although complexity is higher in 3-tier architecture however in 2-tier application performance will be decreased upon increasing the users what makes the application also less scalable.

**Why not 4-or-more-tier architecture?**

The application has more layers the more complex it becomes. Considering time constraints for the project using more layers can be hard to achieve.

To sum up, 3-tier architecture seems to be the golden mean between less and more complex structures.